Web application development with PHP

Anis JEMMALI

EPITA Paris

April 2024

if...elseif...else Statements

```
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;
}
```

if...elseif...else Statements

```php
<?php
$t = date("H");

if ($t < "10") {
            echo "Have␣a␣good␣morning!";
} elseif ($t < "20") {
            echo "Have␣a␣good␣day!";
} else {
            echo "Have␣a␣good␣night!";
}
?>
```

Short Hand If...Else

```
<?php
$x = 6;
$z = ($x<5) ? 'Lower␣than␣5':'Upper␣than␣5';
echo $z;
?>
```

switch Statement

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

```
<?php
switch (expression) {
        case label1:
        //code block
        break;
        case label2:
        //code block;
        break;
        default:
        //code block
}
?>
```

switch Statement

```php
<?php
$grade = 'B';
switch ($grade) {
        case 'A':
                echo 'Excellent!';
                break;
        case 'B':
                echo 'Good job!';
                break;
        default:
                echo 'Work harder next time.';
}
?>
```

The PHP while Loop

```
while (//condition is true){
        //execute instruction
}
```

The PHP while Loop

```
<?php
$i = 0;
while ($i < 6) {
        $i++;fr
        if ($i == 3) continue;
        if($i == 5) break;
        echo $i.'<br>';
}
?>
```

The PHP do...while Loop

```
<?php
do {
        //execute instructions
} while (//condition is true);
?>
```

The PHP do...while Loop

```
<?php
$i=2;
do {
        $i++;
        if ($i == 3) continue;
        if($i == 5) break;
        echo $i;
} while ($i<10);
?>
```

The PHP for Loop

```
for (expression1, expression2, expression3) {
        // code block
}
```

- expression1 is evaluated once
- expression2 is evaluated before each iteration
- expression3 is evaluated after each iteration

The PHP for Loop

```php
<?php
for ($x = 0; $x <= 100; $x+=10) {
        if ($x == 50) continue;
        if ($x == 85) break;
        echo "The␣number␣is:␣$x␣<br>";
}
?>
```

PHP foreach Loop

There are two syntaxes:

```
foreach (iterable_expression as $value){
        //statements
}
```

```
foreach (iterable_expression as $key=>$value){
        //statements
}
```

The first form traverses the iterable_expression. On each iteration, the value of the current element is assigned to $value.
The second form will additionally assign the current element's key to the $key variable on each iteration.

PHP foreach Loop

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {
        echo "$x␣<br>";
}
?>
```

PHP foreach Loop

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x=>$y) {
        echo "The␣key␣is␣$x␣the␣value␣is␣$y␣<br>";
}
?>
```

PHP foreach Loop

```
<?php
$members = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"

foreach ($members as $x => $y) {
        echo "$x␣has␣$y␣year␣old␣<br>";
}
?>
```

PHP foreach Loop

```
<?php
class Car {
        public $color;
        public $model;
        public function __construct($color, $model) {
                $this->color = $color;
                $this->model = $model;
        }
}

$myCar = new Car("red", "Volvo");
foreach ($myCar as $x => $y) {
        echo "$x:␣$y␣<br>";
}
```

PHP Functions

```php
<?php
function sum($x, $y) {
        $s = $x + $y;
        return $s;
}

echo "5␣+␣10␣=␣" . sum(5, 10);
?>
```

PHP Functions

```php
<?php
function sum($x, $y, ...$z) {
        $s = $x + $y;
        if(!empty($z)){
                foreach ($z as $value) {
                        $s+=$value;
                }
        }
        return $s;
}
echo sum(1,2).'<br>';
echo sum(1,2,3,4,5);
?>
```

PHP is a Loosely Typed Language

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 mismatches.

```php
<?php
function addNumbers(int $a, int $b) {
        return $a + $b;
}
echo addNumbers(5, 3.14);
?>
```

PHP is a Loosely Typed Language

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

```php
<?php
declare(strict_types=1);

function addNumbers(int $a, int $b) {
        return $a + $b;
}
echo addNumbers(5, 3.14);
?>
```

PHP Return Type Declarations

```php
<?php
declare(strict_types=1);

function addNumbers(float $a, float $b):int {
        return (int)($a + $b);
}
echo addNumbers(5, 3.14);
?>
```

What is an Array?

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

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

You can have different data types in the same array.

```php
<?php
$myArr = array("BMW", 15, ["apple", "lemon"], myFun);
?>
```

Indexed Arrays

In indexed arrays each item has an index number.

By default, the first item has index 0, the second item has item 1, etc.

```php
<?php
$cars = array("Renault", "BMW", "Toyota");

var_dump($cars);
echo $cars[1]; //Display the second item
$cars[0] = "VW"; //Change the value of the first item
?>
```

Associative Arrays

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

```php
<?php
$car = array("brand"=>"Renault",
             "model"=>"R5",
             "year"=>1984);

var_dump($car);
echo $car["model"];
$car["year"] = 2024;
?>
```

Associative Arrays

Exercise: Create an associative array containing the information of a student

Name	Age	Country
Joel	20	France

Associative Arrays

Exercise: Create an associative table containing the information of a student

Name	Age	Country
Joel	20	France

```php
<?php
$student = array("Name"=>"Joel",
                 "Age"=>20,
                 "Country"=>"France");
?>
```

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

Two-dimensional Arrays

Example:

Name	Stock	Sold
Printer	10	15
Screens	20	5
Keyboard	25	10

```
<?php
$products=array(
        array('Printer', 10, 15),
        array('Screens', 20, 5),
        array('Keyboard', 25, 10)
);
?>
```

Two-dimensional Arrays

Exercise:

1. Display the number of sold printer
2. For each product, display the item name, the number of items in stock and the number of items sold.

Two-dimensional Arrays

Exercise:

```php
<?php
echo $products[0][2];
?>
```

Two-dimensional Arrays

Exercise:

```
<?php
echo $products[0][0]." : ".$products[0][1]
." in stock and ".$products[0][2]." sold<br>";
echo $products[1][0]." : ".$products[1][1]
." in stock and ".$products[1][2]." sold<br>";
echo $products[2][0]." : ".$products[2][1]
." in stock and ".$products[2][2]." sold<br>";
?>
```

Two-dimensional Arrays

Exercise:

```php
<?php
$productNumber = count($products);
for ($i=0; $i < $productNumber; $i++) {
        echo $products[$i][0]." : ".$products[$i][1]
        ." in stock and ".$products[$i][2]
        ." sold<br>";
}
?>
```

Two-dimensional Arrays

Exercise:

```
<?php
 foreach ($products as $p) {
        echo "$p[0]␣:␣$p[1]␣in␣stock
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣and␣$p[2]␣slod<br>";
}
?>
```

PHP Superglobals Variables

Some 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.
The PHP superglobal variables are:

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

PHP $_POST

$_POST contains an array of variables received via the HTTP POST method.
A HTML form submits information via the HTTP POST method if the form's method attribute is set to "POST".
To demonstrate this, we start by creating a simple HTML form:

```
<html>
<body>

<form method="POST" action="demo_request.php">
Name: <input type="text" name="fname">
<input type="submit">
</form>

</body>
```

PHP $_POST

When a user clicks the submit button, the form data is sent to a PHP file specified in the action attribute of the <form> tag.

In the action file we can use the $_POST variable to collect the value of the input field.

PHP file demo_request.php

```php
<?php
$name = $_POST['fname'];
echo $name;
?>
```

PHP $_GET

`$_GET` contains an array of variables received via the HTTP GET method.

There are two main ways to send variables via the HTTP GET method:

- Query strings in the URL
- HTML Forms

PHP $_GET

Query string in the URL

A query string is data added at the end of a URL. In the link below, everything after the ? sign is part of the query string:

```
<a href="demofile.php?name=Joel&age=20">Test $GET</a>
```

The query string above contains two key/value pairs:

- name=Joel
- age=20

PHP $_GET

Query string in the URL

In the demofile.php file we can use the $_GET variable to collect the value of the query string.

```php
<?php
echo "Name=" . $_GET['name'];
echo "Age=" . $_GET['age'];
?>
```

PHP $_GET

$_GET in HTML Forms

A HTML form submits information via the HTTP GET method if the form's `method` attribute is set to `"GET"`.

```
<html>
<body>

<form action="demofile.php" method="GET">
Name: <input type="text" name="name">
Age: <input type="text" name="age">
<input type="submit">
</form>

</body>
</html>
```

PHP $_GET

$_GET in HTML Forms

When a user clicks the submit button, the form fields are sent to the file `demofile.php` specified in the `action` attribute of the `<form>` tag.

Form fields are sent as query strings
`demofile.php?name=Joel&age=20`

PHP $_GET

$_GET in HTML Forms

Code inside the `demofile.php`

```php
<?php
echo "Name=" . $_GET['name'];
echo "Age=" . $_GET['age'];
?>
```

PHP $_SESSION

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session.
The computer knows who you are. It knows when you start the application and when you end.

But on the internet, the web server does not know who you are or what you do, because **the HTTP address doesn't maintain state**.

PHP $_SESSION

`$_SESSION` variable solve this problem by storing user information to be used across multiple pages. By default, session variables last until the user closes the browser.

So; Session variables **hold information about one single user**, and are **available to all pages in one application**.

PHP $_SESSION

Start a PHP Session
A session is started with the `session_start()` function.

⚠ The `session_start()` function **must be the very first thing in your document**. Before any HTML tags.

Now, let's create a new page called "`demo_session1.php`". In this page, we start a new PHP session and set some session variables:

PHP $_SESSION

```
<?php
// Start the session
session_start();
?>
<html><body>

<?php
// Set session variables
$_SESSION["name"] = "Alexandre";
$_SESSION["fav_music"] = "Electro-Funk";
echo "Session␣variables␣are␣set.";
?>
</body></html>
```

PHP $_SESSION

Get PHP Session Variable Values

Now, let's create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

```
<?php
session_start();
?>
<html><body>
<?php
// Echo session variables that were set previously
echo "Name:␣" . $_SESSION["name"] . "<br>";
echo "Favorite␣music␣" . $_SESSION["fav_music"];
?>
</body></html>
```

PHP $_SESSION

Get PHP Session Variable Values

⚠ Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`).

⚠ Also notice that all session variable values are stored in the global `$_SESSION` variable

PHP $_SESSION

Modify a PHP Session Variable

To change a session variable, just overwrite it:

```
<?php
session_start();
?>
<html><body>

<?php
// to change a session variable, just overwrite it
$_SESSION["fav_music"] = "Jazz";
print_r($_SESSION);
?>

</body></html>
```

PHP $_SESSION

Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

```
<?php
session_start();
?>
<html><body>
<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>
```

PHP Include Files

The `include` (or `require`) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

Syntax

```
include 'filename';
```

or

```
require 'filename';
```

PHP Include Files

Use case

Assume we have a standard menu file called "menu.php".
All pages in the Web site should use this menu file. Here is how it can be done:

```
<html><body>

<?php include 'menu.php';?>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>

</body></html>
```

PHP Include Files

Assume we have a file called "vars.php", with some variables defined:

```
<?php
$color='red';
$car='BMW';
?>
```

Then, if we include the "vars.php" file, the variables can be used in the calling file:

```
<html><body>
<?php include 'vars.php';
echo "I have a $color $car.";
?>
</body></html>
```

PHP Include Files

`include` vs. `require`

The include and require statements are identical, except upon failure:

- `require` will produce a fatal error and stop the script
- `include` will only produce a warning and the script will continue

```
//Thorow an error and stop script
require 'not_existing_file.php'

//Produce a warning and the script continue
include 'not_existing_file.php'
```

PHP Include Files

`include` vs. `require`

- Use `require` when the file is required by the application.
- Use `include` when the file is not required and application should continue when file is not found.

header() Function

header() is used to send a raw HTTP header.

⚠ header() must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP. **Syntax**

Example:

```
<?php
header('Location: http://www.google.com/');
exit;
?>
```

File Upload

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the `file_uploads` directive, and set it to On: `file_uploads = On`

File Upload

Create an HTML form that allow users to upload a file:

- Make sure that the form uses `method="post"`
- The form also needs the following attribute: `enctype="multipart/form-data"`.

File Upload

```
<html>
<head></head>
<body>

<form enctype="multipart/form-data"
action="upload.php" method="post">
        Select a document :
        <input name="userfile" type="file" />
        <input type="submit" value="Upload␣file" />
</form>

</body>

</html>
```

File Upload

```php
<?php
$uploadFile = './uploads/'
.$_FILES['userfile']['name'];

move_uploaded_file($_FILES['userfile']['tmp_name'],
$uploadFile);
?>
```