Web Programming with The Laravel Framework

#3 - PHP Fast Track

Steven Lolong

steven.lolong@uni-tuebingen.de

Programming Languages Research Group
Tuebingen University

26 April 2022

Eberhard Karls
Universität
Tübingen

PHP

Start php file with **<?php** and close with **?>**

```php
<?php
/**
 * @author: Steven L.
 * @date: 26 April 2022
 */

//  Show "Hello World!" on web page
$notify = "Hello World!";
echo $notify;

?>
```

Listing: PHP structure – 3-helloWorld.php

Eberhard Karls Universität Tübingen

Variable

- **$** is the key for variable's name.

```
6  $vStr = "This is string";
7  $vInt = 323;
8  $vBoolean = true; //false
9  $vDouble = 9.9;
10 $vNull = null;
11 $vArray = Array(1,3,5,7);
```

Listing: Variable – 3-var-and-type.php

Type Juggling

- PHP is **dynamic type checking**, it means checking the type when running.
- There is no need to annotate type explicitly.

```
12 echo '$vStr = ' . $vStr . " : " . gettype($vStr) . "<br/>";
13 echo '$vInt = ' . $vInt . " : " . gettype($vInt) . "<br/>"
       ;
14 echo '$vBoolean = ' . $vBoolean . " : " . gettype($vBoolean
       ) . "<br/>";
15 echo '$vDouble = ' . $vDouble . " : " . gettype($vDouble)
       . "<br/>";
16 echo '$vNull = ' . $vNull . " : " . gettype($vNull) . "<br
       />";
17 echo '$vArray = ' . "<br/>";
18 print_r($vArray);
19 echo "<br/> : " . gettype($vArray) . "<br/>";
```

Listing: Variable – 3-var-and-type.php

Type Casting

- (*new_type*) *variable* or *value*
- *settype*(*variable_name*, *new_type*)

```
21 echo 'cast $vInt to string, new type: ' . gettype((string)
     $vInt) . "<br/>";
22 echo 'cast $vBoolean to integer, new type: ' . gettype((int
     )$vBoolean) . "<br/>";
23 echo "cast $vStr to integer, old value: $vStr <br/>";
24 settype($vStr, "integer");
25 echo "New type: " . gettype($vStr) . ", value $vStr <br/>";
```

Listing: Type casting – 3-var-and-type.php

Operator

- Arithmetic: $+, -, *, /, \%, **$
- Assignment: $=, +=, -=, *=, /=, \%=$
- Comparison: $==, ===, !=$ (<>), $!==, <, >, <=, >=, <=>$
- Increment / decrement: $++\$x, \$x++, --\$x, \$x--$
- Logical: *and*, *or*, *xor*, &&, ||, !
- String: ., .=
- Array: $+, ==, ===, !=$, <>, $!==$
- Conditional: ? :, ??

Operator equivalent

- **$a += $b** is equal to **$a = $a + $b**. This is also applied to $-=, *=, /=, \%=$.
- **++$a** is equal to **$a = $a + 1**. This is also applied to $--$.

Control Statement

- **if** statement, executes some code if on 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 from many blocks of code to be executed.

if and if-else

```
if Condition then
    "code to be executed if Condition is true";
end
```

Algorithm 1: if statatement

```
if Condition1 then
    "code to be executed if Condition1 is true";
else
    "code to be executed if Condition1 is false";
end
```

Algorithm 2: if-else statatement

if Statement

```php
$weather = "rain";
$temperatur = 10;
// if statement
if($weather == "rain"){
  echo "use umbrella <br/>";
}
```

Listing: if – 3-control-statement.php

if-else Statement

```
10 if($weather == "rain"){
11   echo "bring umbrella <br/>";
12 }else{
13   echo "leave umbrella at home <br/>";
14 }
```

Listing: if-else – 3-control-statement.php

if-elseif-else Statement

```
17 if($temperatur < 2){
18   echo "wear 4 layers <br/>";
19 }elseif(($temperatur >= 2) && ($temperatur < 10)){
20   echo "wear 3 layers <br/>";
21 }elseif(($temperatur >= 10) && ($temperatur < 18)){
22   echo "wear 2 layers <br/>";
23 }elseif(($temperatur >= 18) && ($temperatur < 25)){
24   echo "wear 1 layers <br/>";
25 }else{
26   echo "wear t-shirt <br/>";
27 }
```

Listing: if-elseif-else – 3-control-statement.php

Switch-Case Statement

```
switch Variable do
    case expression-1 do
        "case-1";
        break;
    end
    case expression-2 do
        "case-2";
        break;
    end
    otherwise do
        "Other";
    end
end
```

Algorithm 3: switch-case-default statatement

switch-case statement

```
30 switch($temperatur){
31   case ($temperatur >= 2) && ($temperatur < 10):
32     echo "wear 3 layers <br/>";
33     break;
34   case ($temperatur >= 10) && ($temperatur < 18):
35       echo "wear 2 layers <br/>";
36     break;
37   case ($temperatur >= 18) && ($temperatur < 25):
38       echo "wear 1 layers <br/>";
39     break;
40   default:
41     echo "wear t-shirt <br/>";
42 }
```

Listing: switch-case – 3-control-statement.php

Looping

- **while** pre-test looping, loops through a block of code as long as the specified condition is true.
- **do-while** post-test looping, 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.

While Statement

Pre-test
while Condition do
  Statement;
end

Algorithm 4: while statatement

```
2 $a = 1;
3 // while
4 while($a < 5){
5   echo "a = $a <br/>";
6   $a++;
7 }
```

Listing: while – 3-looping.php

Do-While Statement

Post-test

do

    Statement;

while Condition;

Algorithm 5: while statatement

```
10 do{
11   echo "a = $a <br/>";
12   $a++;
13 }while($a < 5);
```

Listing: do-while – 3-looping.php

```
16 $b = 2;
17 do{
18   echo "b = $b<br/>";
19 }while($b < 2);
```

Listing: post test – 3-looping.php

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

For Statement

Condition: start value ; condition ; post statement

for Condition do
  Statement;
end

Algorithm 6: for statatement

```php
for($i = 1; $i < 10; $i++){
  echo "i = $i <br/>";
}
```

Listing: for – 3-looping.php

Nested Looping

```
26 // nested looping
27 for($i = 1; $i < 5; $i++){
28   for($j = $i; $j < 5; $j++){
29     echo "*";
30   }
31   echo "<br/>";
32 }
```

Listing: nested looping – 3-looping.php

What is the output?

Iteration

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

foreach *array* **as** *value* **do**
  Statement;
end

Algorithm 7: foreach statatement

```
35 $colors = array("Red", "Green", "Blue");
36 foreach($colors as $value){
37   echo "$value <br/>";
38 }
```

Listing: iteration – 3-looping.php

Iteration

```
41 $colorsHex = array("red" => "#ff0000", "green" => "#00ff00",
       "blue" => "#0000ff");
42 foreach($colorsHex as $indx => $val){
43   echo "Index = $indx, Value = $val <br/>";
44 }
```

Listing: extracting the index – 3-looping.php

What is the output?

Function

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function (apply over value).

Function

syntax

function **funcName** (*parameters*){
statement-1;
⋮
statement-n;
return *value*;
}

Function

```php
// function abstraction
function sayHello(){
  echo "Hello <br/>";
}
// function application (call)
sayHello();
```

Listing: function – 3-functions.php

Function

```
9  function funEmpty(){
10   echo "Hello <br/>";
11 }
```

Listing: No parameter and no return value – 3-functions.php

```
13 function funRet(){
14   return "Hello <br/>";
15 }
```

Listing: Parameter but no return value – 3-functions.php

```
17 function funParam($param1, $param2){
18   echo "param1 = $param1, param2 = $param2 <br/>";
19 }
```

Listing: No parameter but return value – 3-functions.php

```
21 function funAdd($a, $b){
22   return ($a + $b);
23 }
```

Listing: Parameter and return value – 3-functions.php

UNIVERSITÄT TÜBINGEN

Array

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

Normal Variabl vs Array Variable

```php
$bike1 = "Hercules";
$bike2 = "Cube";
$bike3 = "Rose";
// Or
$bikes = array("Hercules", "Cuba", "Rose");
echo count($bikes) . "<br/>";
```

Listing: Array – 3-array.php

Indexed Arrays

```
10 $bikers = array();
11 $bikers[0] = "Biker_A";
12 $bikers[1] = "Biker_B";
13 $bikers[2] = "Biker_C";
```

Listing: Indexed array – 3-array.php

Associative Arrays

```
16 $bikersAndBikes = array("Biker_A" => "Hercules", "Biker_B"
       => "Cube", "Biker_C" => "Rose");
```

Listing: Associative array – 3-array.php

echo $bikersAndBikes["Biker_B"];

Multidimensional Arrays

```php
$cars = array(
  array("Mercedes", 22, 18),
  array("BMW", 15, 13),
  array("Audi", 4, 12),
);
```

Listing: Multidimensional array – 3-array.php

How to print all data in Multidimensional Array?

Object Oriented Programming in PHP

The fundamental idea behind an object-oriented language is to enclose a bundle of variables and functions into a single unit and keep both variables and functions safe from outside interference and misuse.
An object contains:

- Methods (functions)
- Properties (variables)

Definition

- Class: This is a programmer-defined data type, including local functions and local data. Programmers can think of a class as a template for making many instances of an object's same kind (or class).
- Object: An individual instance of the data structure defined by a class. Programmers define a class once and then make many objects that belong to it. Objects are also known as instances.
- Member Variable: These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions or instances. These variables are called attributes of the object once an object is created.
- Member function: These are the function defined inside a class and are used to access object data.
- Inheritance: When a class is defined by inheriting the existing function of a parent class, it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

Definition

- Parent class: A class that is inherited from another class. It is also called a base class or superclass.
- Child Class: A class that inherits from another class. It is also called a subclass or derived class.
- Polymorphism: This object-oriented concept where the same function can be used for different purposes. For example, the function name will remain the same, but it takes a different number of arguments and can do different tasks.
- Overloading: A polymorphism in which some or all operators have different implementations depending on their arguments. Similarly, functions can also be overloaded with different implementations.
- Data Abstraction: Any representation of data in which the implementation details are hidden (abstracted).

Definition

- Encapsulation: Refers to a concept where we encapsulate all the data and member functions to form an object.
- Constructor: Refers to a particular type of function called whenever there is an object formation from a class (instance).
- Destructor: Refers to a particular type of function called whenever an object is deleted or goes out of scope.

Class

- Basic class definitions begin with the keyword class, followed by a class name, followed by a pair of curly braces, which enclose the definitions of the properties and methods belonging to the class.
- The class name can be any valid label, provided it is not a PHP reserved word. A valid class name starts with a letter or an underscore, followed by any number of letters, numbers, or underscores.
- keywords:
  class, *const*, *static*, *public*, *private*, *protected*, $*this*, ::, $-> $, *.self*

Class

- **class**, keyword to define class.
- **conts**, keyword to create a constant variable.
- **static**, keyword to create a static variable.
- Visibility (accessibility):
  - *private*. Visible (accessible) only inside the parent; no access is granted outside the class.
  - *protected*. It is visible (accessible) in the parent and child class. No access is granted from outside the class except a class that's a child of the class with the protected property or method.
  - *public*. Visible (accessible) to all, can be accessed from outside the class.
- **$this**, keyword to access current class.
- **::** and **->**, to access class's member.

Class

```
3  class Car{
4    const constant_1 = "hello PHP";
5    private $wheel = 4;
6    protected $topSpeed = 200;
7    public function __construct(){
8        echo "Constructor of Car<br/>";
9    }
10   public function __destruct(){
11       echo "Destructor of Car<br/>";
12   }
13   public function setTopSpeed($newTopSpeed){
14       $topSpeed = $newTopSpeed;
15   }
16   public function showTopSpeed(){
17       echo $this->topSpeed . "<br/>";
18   }
19   public function printCons(){
20       echo self::constant_1 . "<br/>";
21   }
22 }
```

Listing: Class – 3-oop.php

Instance

```php
$cars = new Car();
$cars->printCons();
$cars->showTopSpeed();
// error, protected property
// echo $cars->topSpeed;
```

Listing: Instance – 3-oop.php

Inheritance

```
30 class Mercedes extends Car{
31   public function __construct(){
32       $this->topSpeed = 300;
33       echo "Mercedes's constructor <br/>";
34   }
35   public function __destruct(){
36       echo "Destructor of Mercedes <br/>";
37   }
38 }
39
40 $typeCMercy = new Mercedes();
41 // Set topSpeed in Mercedes and call method of Car
42 $typeCMercy->showTopSpeed();
```

Listing: Inheritance and its instance – 3-oop.php

Trait

- Traits are used to declare methods that can be used in multiple classes.
- PHP implements a way to reuse code called Traits.

Is it necessary for programmers?

Trait

```
2  trait Car{
3    public function start(){
4      echo "Start <br/>";
5    }
6    public function stop(){
7      echo "Stop <br/>";
8    }
9  }
10
11 class Mercedes{
12   use Car;
13 }
14
15 $newM = new Mercedes();
16 $newM->start();
17 $newM->stop();
```

Listing: Trait – 3-oop.php

Cookie

- A cookie is often used to identify a user.
- A cookie is a small file that the server embeds on the user's computer.
- Each time the same computer requests a page with a browser, it will send the cookie.
- Programmers can create and retrieve cookie values.
- Syntax:
  setcookie(*name*, *value*, *expire*, *path*, *domain*, *secure*, *httponly*)
- The only name parameter is mandatory, and others are optional.

Cookie

```
6 setcookie("wp1", "It's web programming", time() + (3600 *
      30), "/");
7 setcookie("wp2", "It's web programming-Secure", time() +
      (3600 * 30), "/","",true);
```

Listing: Set cookie – 3-cookies-setter.php

```
2 if(isset($_COOKIE["wp1"])){
3   echo "Cookie1 (not secure): " . $_COOKIE["wp1"] . "<br/>";
4 }
5 if(isset($_COOKIE["wp2"])){
6   echo "Cookie2 (secure): " . $_COOKIE["wp2"] . "<br/>";
7 }
```

Listing: Get cookie – 3-cookies-getter.php

Session

- The session allows users to store information on the server for later use (i.e. username and shopping cart items).
- The session information is temporary and immediately deleted after the user leaves the website that uses sessions.
- Sessions work by creating a unique identification(UID) number for each visitor and storing variables based on this ID. It helps to prevent two users' data from getting confused when visiting the same webpage.
- To start a session, users must call *session_start*() before creating a session variable.
- To destroy all sessions, call *session_destroy*().

Let's do it!

Let us build a small web application with authentication!

See on the whiteboard how this application works

Eberhard Karls
Universität
Tübingen

Login page

```
2 session_start();
3 $privatePage = "./3-session-private-page.php";
4 if(isset($_SESSION['login']) && ($_SESSION['login'] == true)
      ){
5   header('Location: ' . $privatePage);
6 }
```

Listing: Session in login page – 3-session-login.php

Login page

```
17  <h2>The Witcher</h2>
18  <h4 style="color: red;"><?php echo (isset($_SESSION['
      notifyLogin']) ? $_SESSION['notifyLogin'] : ""); ?> </h4
      >
19  <form action="3-session-process-login.php" method="post"
      target="_self">
20    <fieldset>
21      <legend>Login</legend>
22      <label for="uNameId">Username:</label>
23      <input type="text" id="uNameId" name="uName"><br><br>
24      <label for="passwId">Password:</label>
25      <input type="password" id="passwId" name="uPassw"><br>
26      <input type="submit" value="Login">
27    </fieldset>
28  </form>
```

Listing: Form in login page – 3-session-login.php

Processing Login Information

```
2  session_start();
3  // act as persistence
4  $uName = "gerald";
5  // this is hash of "007" using md5;
6  $uPass = "9e94b15ed312fa42232fd87a55db0d39";
7
8  $privatePage = "./3-session-private-page.php";
9  $loginPage = "./3-session-login.php";
10
11 // create notification variable
12 if(!isset($_SESSION['notifyLogin'])) {
13   $_SESSION['notifyLogin'] = "";
14 }
```

Listing: Act as persistence – 3-session-process-login.php

Processing Login Information

```
16 if(isset($_POST["uName"]) && isset($_POST["uPassw"])){
17   if(($_POST["uName"] == $uName) && (md5($_POST["uPassw"])
       == $uPass)){
18     $_SESSION['login'] = true;
19     header("Location: " . $privatePage);
20   }else{
21     $_SESSION['notifyLogin'] = "incorrect username or
       password";
22     header("Location: " . $loginPage);
23   }
24 }else{
25   header("Location: " . $loginPage);
26 }
```

Listing: Controller and routing – 3-session-process-login.php

Private Web Page

```php
<?php
session_start();
if(!isset($_SESSION['login'])){
  header("Location: 3-session-logout.php");
}
?>
<html>
  <head>
    <title>Welcome!</title>
  </head>
  <body>
    <h1>Welcome, Gerald</h1>
    <a href="3-session-logout.php">logout</a>
  </body>
</html>
```

Listing: Private page – 3-session-private-page.php

Process Logout

```php
<?php
session_start();
session_destroy();
$loginPage = "./3-session-login.php";
header('Location: ' . $loginPage);
?>
```

Listing: Destroy session and routing – 3-session-logout.php

Eberhard Karls Universität Tübingen