Introduction to PHP

Jacek Dajda [dajda@agh.edu.pl]
Room: 3.11

Agenda

- Language overview
- Basic info, variables, types
- Control structures
- Arrays
- Functions
- Embedding into HTML

Overview

Server-side programming

- Unlike JavaScript which is executed by the web browser, all PHP code is executed on the web server.
- The goal of the server-side language is to produce HTML which can be interpreted by the client browser
- Server-side languages are embedded into the web servers. For example, PHP is available as a module for the most popular web server Apache

Web servers and static web pages

Usually when you type a URL in your browser:

`http://server/path/file`

1. your computer looks up the server's IP address using DNS
2. your browser connects to that IP address and requests the given file
3. the web server software (e.g. Apache) grabs that file from the server's local file system, and sends back its contents to you

Dynamic web pages

- Static web pages provide only data written in HTML file
- We need more dynamic behavior:
  - provide different content depending on context
  - interface with other services: database, e-mail, etc
  - authenticate users
  - receive data from users
  - process form information
- The HTML and CSS is not enough

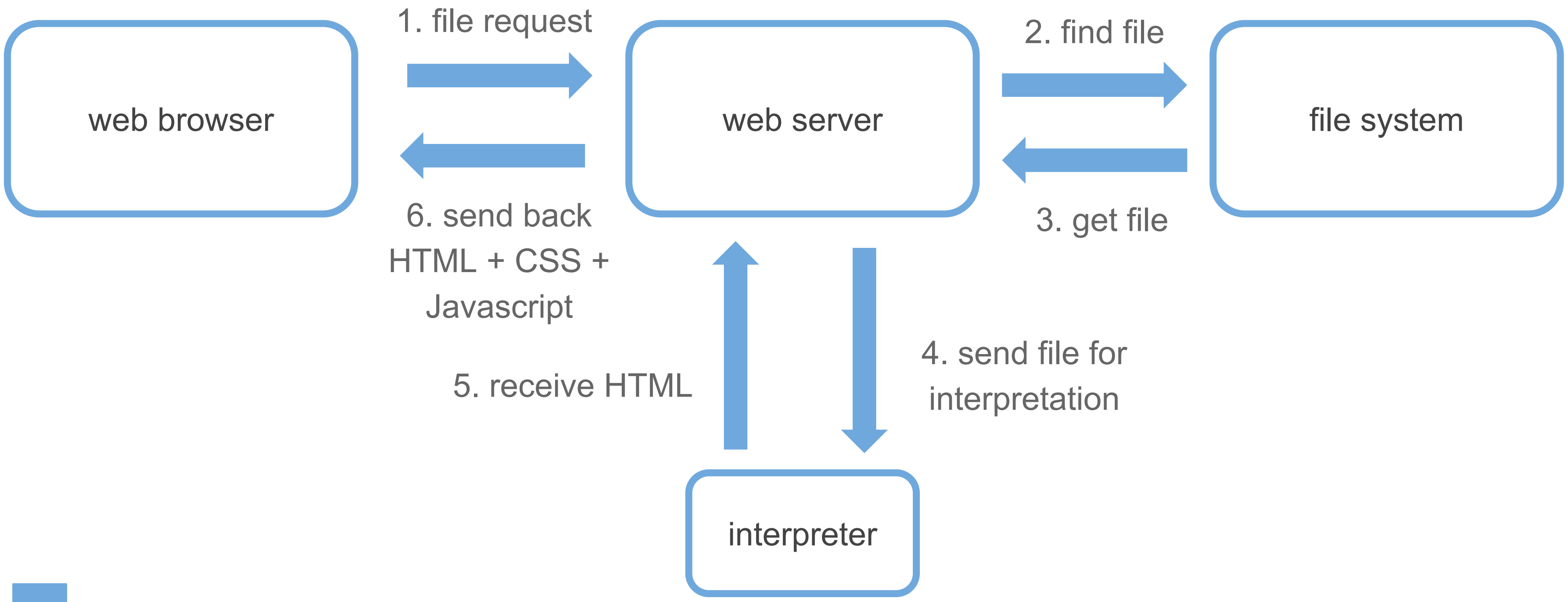
Dynamic web pages

Usually when you type a URL in your browser:

`http://server/path/scriptfile`

1. your computer looks up the server's IP address using DNS
2. your browser connects to that IP address and requests the given file
3. the web server software (e.g. Apache) grabs that file from the server's local file system **and interprets the file on the server in proper interpreter**
4. **the produced output** is sent back to you

Web server programming



PHP language in short

- Name: “PHP: Hypertext Preprocessor” (originally named Personal Home Page Tools)
- Invented by Rasmus Lerdorf in 1994 to help track visitors to Lerdorf’s personal web site
- Licensed under the GPL and is free
- Popular server-side language (most popular?) for web servers.
- Available on a variety of web servers (Apache, IIS, NGINX, etc.) and operating systems
- Script file extension: **.php**

Basic info, variables, types

PHP - basic info

- PHP is purely interpreted language
- PHP is dynamically typed language
- PHP is a loosely-typed (weak typed) language
  - Do not need to declare the type of a variable
  - Type can change throughout the program
- Every statement ends in a ;

Hello world!

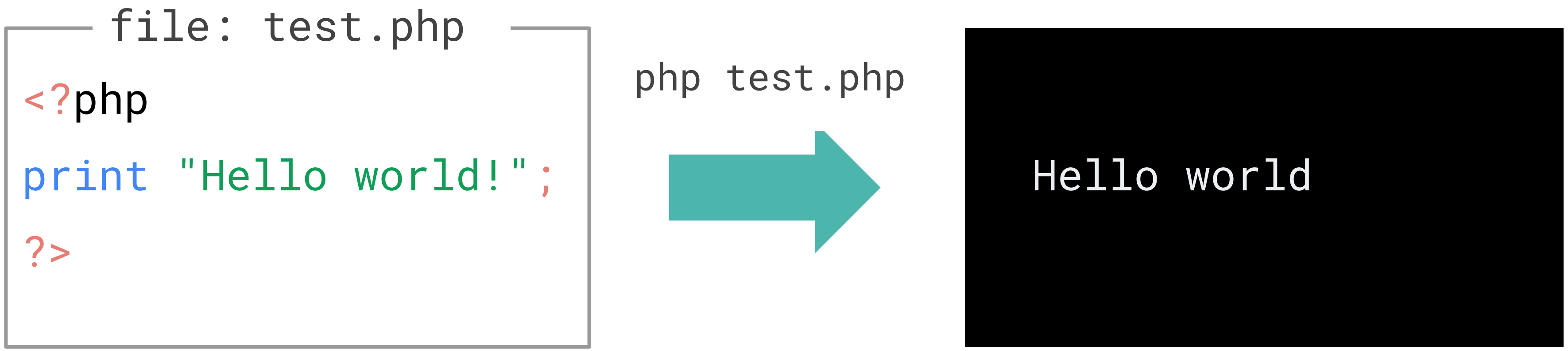
- A block or file of PHP code begins with <?php and ends with ?>
- PHP statements, function declarations, etc. appear between these endpoints

```
<?php
print "Hello world!";
?>
```

- You can also use `print(“Hello world”)`, `echo(“Hello world”)`, `echo “Hello world”`

Running PHP program

- Usually it is run by web server
- You can run it using command line interface (CLI) using php command



file: test.php

```
<?php
print "Hello world!";
?>
```

```
php test.php
```

```
Hello world
```

Variables

- Always starts with $ followed by either letter or underscore.
- Can be composed of numbers, underscores, and letters

```
$some_variable = “test”;

$this_is_2nd_variable = 1;

$_some_strange_variable = “other test”;
```

Case sensitive

- PHP is case sensitive for variable names
- Reserved words and functions are not case sensitive
- While, wHiLe etc. are all same

```
$my_var = 10;
$my_Var = 12;
print $my_var;
print $my_Var;
```

$\rightarrow$

```
10 12
```

Weak typed language

- No type declaration is possible

```
$my_var = “10”;
echo $my_var + 2;
print $my_var;
```

```
12
```

Types

- Four scalar types: integer, double, string, boolean
- Two compound types: array and object
- Two special types - resource and NULL
- Unassigned variables are also called unbounded variables and have type NULL

Operators

- Assignment
  - `= += -= /= *= %= ++ --` - like most programming languages
  - `.=` - string concatenation operator (the only one operating one string)
- Arithmetic
  - `+ - * / %` - like most programming languages
- Comparison
  - `== != < > <= >=` - like most programming languages. Also `<>` is the same as `!=`
  - `===` - true if arguments are equal and the same data type
  - `!==` - true if arguments are not equal or they are not of the same data type
- Logical
  - `&& ||` ! -like most programming languages
  - `xor` - true if either (but not both) of its arguments are true

String concatenation

```
print "Hello " . " world";
```

```
Hello world
```

```
$message = "Hello";
$message .= " world";
print $message;
```

```
Hello world
```

== versus ===

- == tests for “equality” in value but not necessarily type
- === tests for “identity” in value AND type
- == ignores the distinction between:
  - Integers, floating point numbers, and strings containing the same numerical value `2 == “2”`
  - Nonzero numbers and boolean TRUE `2 == TRUE`
  - Zero and boolean FALSE `0 == FALSE`
  - Empty string, the string ‘0’ and boolean FALSE `“” == 0`
  - Any other non-empty string and boolean TRUE `“abc” == TRUE`

Strings

- String literals are defined with either single quotes ' ' or double quotes " "

```
$my_text = "Hello!";
$my_text = 'Hello!';
```

- Double quotes are interpreted:
  - Variable values are substituted for their names
  - Escaped characters are replaced by their values

Strings: single quotes vs. double quotes

```
$who = "world";
print "Hello $who!";     →   Hello world!
print 'Hello $who!';     →   Hello $who!
```

Useful String functions

- `int strlen($str)` - Returns string length.
- `int strcmp($str1, $str2)` - Returns < 0 if str1 is less than str2; > 0 if str1 is greater than str2, and 0 if they are equal. (strcasecmp for case-insensitive comparison.) The < > == operators can also be used if both arguments are strings. strcmp is useful if an argument may not be a string and has to be converted into one.
- `string strstr($text, $search)` - returns first occurrence of $search in $text, FALSE if not found. (stristr for case-insensitive search.)
- `string str_replace($find, $replace, $text)` - Replaces all occurrences of $find with $replace in $text.
- More information available at http://www.php.net/manual/en/ref.strings.php

String functions - example

```
$email  = 'name@example.com';
$domain = strstr($email, '@');
echo $domain; // prints @example.com
echo strlen($email); // prints 16
```

Comments

```
// Single-line comment

# This is also single line comment (Perl-like)

/*  This is

    a multi-line

    comment */
```

Control Structures

Conditionals

```
if (condition)
        statement;
```

```
if (condition)
        statement;
else
        statement;
```

```
if (condition)
        statement;
else if (condition) // or elseif
        statement;
else
        statement;
```

‘If’ conditional - example

```
$value = 2;
if ($value<0)
    print "Negative";
else if ($value>0)
    print "Positive";
else
    print "0 value";
```

What is printed??

Switch

- Similar to other languages
- Works for integers, floats, strings

```
$condition_value = "a";
switch($condition_value) {
    case "a": print "case 1"; break;
    case "b": print "case 2"; break;
    case "c": print "case 3"; break;
    default: print "default";
}
```

Loops

```
$n = 0;
while ($n<10) {
    print("$n ");
    $n++;
}
```

What is printed??

‘Do’ and ‘for’ loops

```
do {
    print("$n ");
    $n++;
} while ($n < 10);


for ($n=1; $n<10; $n++) print("$n ");
```

Foreach loop

```
foreach($myarray as $item) print("$item ");
```

```
$my_array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
foreach ($my_array as $item) print("$item ");
```

Arrays

Arrays

- Arrays can be used in two ways:
    - as lists

```
$array[0]
```

    - as maps

```
$array["key"]
```

- http://www.php.net/manual/en/ref.array.php

Values in arrays

- Creating empty array:

```
$array = array();
$array = [];
```

- Creating initialized array:

```
$names = array(‘John’, ‘Patrick’);
$countries = array("FR" => "France", "PL" =>
"Poland");
```

Values in arrays

- Arrays can have any size and contain any type of value

```
$names = array(‘John’, 2);
```

- No danger of going beyond array bounds

```
$names = array("John", 2);
$names[3] = "Frank";
$names[“Frank”] = "Sinatra";
```

Operating on arrays as lists

```
$names = array("Patrick", "John", "Marie");
$names[] = "Claire"; // add to the list
unset($names[1]); // remove ‘John’ from list (empty index 1)
$names = array_values($names); // rebuild index
print_r($names); // Array([0] => Patrick [1] => Marie [2] =>
Claire )
```

Operating on arrays as maps

```
$countries = array("FR" => "France", "PL" => "Poland");
$countries["UK"] = "United Kingdom"; // add to the list
unset($countries["UK"]); //remove element with "UK" key
print_r($countries); // Array ( [FR] => France [PL] =>
Poland )
```

Arrays are associative

- It is always key and value element
- In case of list indexes are just unique keys
- Getting array size:

```
$names = array("Patrick", "John", "Marie");
print count($names); // 3

$countries = array("FR" => "France", "PL" =>
"Poland");
print count($countries); // 2
```

Iterating through array

```
foreach ($array_name as $value) {
}
```

```
$names = array("Patrick", "John", "Marie");
foreach ($names as $name) {
 print "$name, ";
}
// Patrick, John, Marie,
```

Iterating through array

```
foreach ($array_name as $key => $value) {
}

$countries = array("FR" => "France", "PL" =>
"Poland");
foreach ($countries as $key => $value) {
    print "$key => $value, ";
}
// FR => France, PL => Poland
```

Functions

Functions

- Functions may be declared anywhere in the source code (i.e., they do not need to be defined before they are called).
- Function names are case-insensitive

```
function func_name($param_1, $param_2, ..., $param_n) {
    // code
    return $retval; // optional: can return a scalar or
an array
}

$result = func_name($arg1, $arg2, ..., $argn);
```

Parameter passing

- Arguments may be passed by value (default) or by reference (using &)

```
// Pass by value
function sum($a, $b) {
    return $a + $b;
}

print(sum(5, 3)); // 8
```

```
// Pass by reference
function sum(&$result, $a, $b)
{
    $result = $a + $b;
}

$result;
sum($result, 5, 3);
print($result); // 8
```

Returning arrays

```
function generate_names() {
    return array("John", "Marie", "Claire");
}
list($name1, $name2, $name3) = generate_names();
print($name1); // John
print($name2); // Marie
print($name3); // Claire
```

Variables scope

- All variables have local scope (i.e., they are accessible only within the function or block in which they are initialized)
- Global variables may only be accessed within a function by using the global keyword

```
$message = "Hello world";
function show_global()  {
    echo $message;
}
show_global(); // NOTICE Undefined variable: x on line number 6
```

Variables scope - global keyword

```php
$message = "Hello world";
function show_global()  {
    global $message;
    echo $message;
}
show_global(); // "Hello world"
```

Classes

Classes

- Version 5 of PHP introduced full object model
- It supports such elements concepts as constructors, destructors, abstract classes and methods, interfaces, dynamic creation of members and methods, etc.
- Full documentation: http://php.net/manual/en/language.oop5.php

Property and method declaration

```php
class SimpleClass
{
    public $var = 'a default value';

    public function displayVar() {
        echo $this->var;
    }
}

$myObject = new SimpleClass();
echo $myObject->var; // a default value
echo $myObject->displayVar(); // displaying: a default value
```

Object references

```
class A {
    public $foo = 1;
}

$a = new A;
$b = $a;     // $a and $b are copies of the same identifier
$b->foo = 2;
echo $a->foo."\n"; // 2
```

Constructors

```
class Car
{
    public $model;

    public function __construct() {
        $this->model = "Peugot 102";
    }
}

$my_car = new Car();
echo $my_car->model; // Peugot 102
```

Acessing object properties

- You can access member variables in an object using another variable as name:

```
class SimpleClass
{
    public $var = 'a default value';
}
$someText = "var";
$myObject = new SimpleClass();
echo $myObject->$someText; // a default value
```

Same for methods

- You can access member variables in an object using another variable as name:

```
class SimpleClass
{
    public function displayVar() {
        echo "Hello world!";
    }
}
$someText = "displayVar()";
$myObject = new SimpleClass();
echo $myObject->$someText; // Hello world!
```

Embedding into HTML

‘Hello world’ index.php file

```
<html>
<head>
<title>Hello World</title>
</head>
<body>
<?php
$name = "World";
echo "<h1>Hello, $name!</h1>";
?>
</body>
</html>
```

Output

```
<html>
<head>
<title>Hello World</title>
</head>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

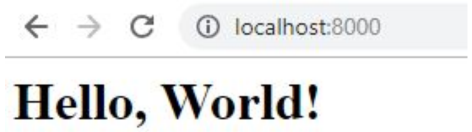
Running PHP program in web server

- You can install Apache with PHP and put your index.php file in home directory
- Since PHP Ver. 7.0 you can use built-in web server

file: test.php

```
<html>
<head>
<title>Hello World</title>
</head>
<body>
<?php
$name = "World";
echo "<h1>Hello, $name!</h1>";
?>
</body>
</html>
```

```
php -S localhost:8000 test.php
```



Hello, World!

Include

- One of the most useful tools is to insert another php script from a file into the current php script.
- The command `include("filename");`
- This will import contents of a text file called filename and insert it at the include spot.
- The included text may be composed of XHTML, PHP or both. Any PHP in the included text must be inside the <?php tags

Include

```
<html>
<head>
<title>Hello World</title>
</head>
<body>
<?php
include(“header.php”);
$name = "World";
echo "<h1>Hello, $name!</h1>";
include(“footer.php”);
?>
</body>
</html>
```

Further reading

Selected online resources

- www.phptester.net - playground, test your PHP code online, no need to install anything
- www.php.net – PHP distribution, tutorials, newsgroups, and more
- www.phpfreaks.com - PHP and MySQL tutorials, scripts, forums, and more
- www.phpbuilder.com – Collection of PHP resources