

Web Technologies

Muhammad Kamran

Lecture 19



PHP Basics

Web Applications in Hatch



1. What are PHP, CGI and Web Server?
2. Web applications
3. Syntax
4. Variables, variable types
5. Basic functions
6. Some predefined variables
7. Strings escaping
8. PHP – advantages and disadvantages

What are PHP, CGI and Web Server?

- ◆ "PHP Hypertext Preprocessor"
 - ◆ Scripting language
 - ◆ Creation of dynamic content – i.e. HTML and JSON
 - ◆ Interaction with databases (CRUDs)
 - ◆ Server side, or via command line (CLI)
 - ◆ Can be embedded in HTML
 - ◆ First introduced in 1995 as module for Apache
 - ◆ Open source, written in C
 - ◆ Similar to Perl and C

- ◆ "Common Gateway Interface"
 - ◆ Unified specification for interaction between web server and a CGI program
 - ◆ The CGI program accepts data from the web server and usually returns generated HTML content
 - ◆ CGI programs are used to generate also XML files, images, video streams and any other content, understandable by the browser
 - ◆ The very code of the CGI program is not visible for the client, only it's output

What is web server?

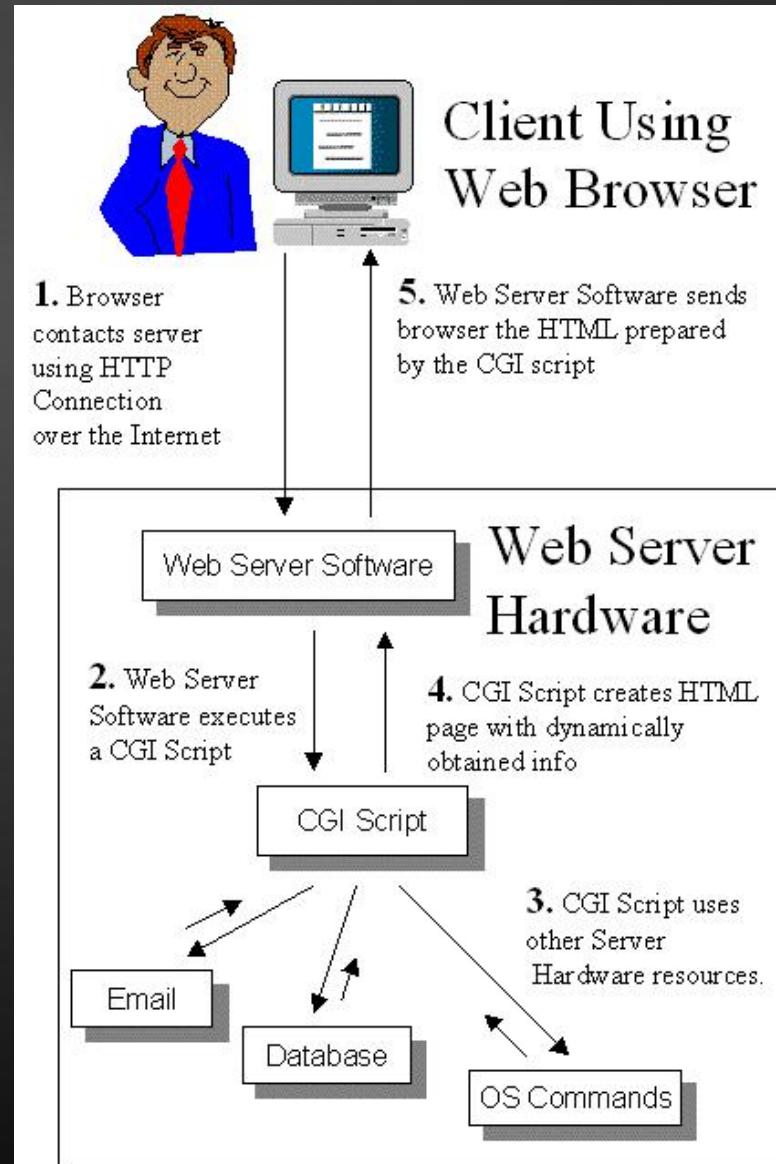
- ◆ Computer program that is responsible for handling HTTP requests and returning responses
 - ◆ Receives HTTP request
 - ◆ Finds the requested resource or executes CGI program
 - ◆ Returns the resource or program output to the browser
 - ◆ Most common web servers are Apache, IIS, NodeJS, nginx, ligHttpd and others
- ◆ "LAMP" – Linux, Apache, MySQL, PHP/Perl – the most common software on a web server

Web applications

- ◆ Application that can be accessed over the web
 - ◆ Relies on web servers
 - ◆ Usually written in server-side scripting languages like PHP, Perl, Java, ASP
 - ◆ Has dynamically generated content
 - ◆ Commonly structured as three-tier application - web server, CGI Program (s) and database
 - ◆ Not just web pages

- ◆ Gmail
- ◆ SkyDrive / Live.com
- ◆ Google Office / Windows Office
- ◆ Prezi
- ◆ Creately
- ◆ Slideshare
- ◆ Almost everything that can be accessed via web browsers ☺

Web application lifecycle



- ◆ The PHP code is usually in files with extension ".php"

- ◆ Can be configured

- ◆ The code can be run by a web browser directly from the file.

```
<html>
<head><title>Hello world page</title></head>
<body>
    <?php print ("Hello PHP!"); ?>
</body>
</html>
```

<?php denotes start of
PHP code

?> denotes end of PHP
code

- ◆ In the webroot directory
 - ◆ XAMMP – htdocs/
 - ◆ WAMP – www/
- ◆ Webroot directory can be configured
- ◆ Can be accessed via
http://localhost/path/to/scriptName.php

Hello PHP

Live Demo – Web and CLI

Syntax

- ◆ The PHP code starts with <?php and ends with ?>
 - ◆ Depending on server configuration may also start with <? (Short style) – but this is bad practice!
 - ◆ In terms of XML the <?php - ?> part is called "processing instruction"
- ◆ PHP follows the Perl syntax
 - ◆ Simplified
 - ◆ Procedural (Now has OOP too)
 - ◆ Similar to C and Java

- ◆ PHP Script contains one or more statements
 - ◆ Statement are handed to the PHP Preprocessor one by one
 - ◆ Each statement ends in semicolon ";"
- ◆ Our first script contains only one statement:

```
<?php  
print ("Hello PHP!"); // this is the statement  
?>
```

- ◆ call of the function **print**

- ◆ PHP script can contain unlimited number of statements

```
<?php  
print "<div>";  
print "Hello PHP!";  
print "</div>";  
?>
```

- ◆ Some function can be called without brackets
- ◆ You can add comments to the code
 - ◆ Starting with "//", "#" or block in "/*" and "*/"
 - ◆ Only "/*" – "*/" can be used over several lines
 - ◆ Comments are NOT executed

- ◆ Short opening tag <?=

```
<html>
<head><title>Hello world page</title></head>
<body>
<?="Hello PHP!" ?>
</body>
</html>
```

- ◆ Forces the result of the expression to be printed to the browser
- ◆ Similar to print
- ◆ Allowed to omit ending ";"

Shorttags

Live Demo

Variables

- ◆ All variables in PHP start with \$ (Perl style)

```
<?php // declare string variable $output  
$output = "<div>Hello PHP!</div>";  
print $output;  
?>
```

- ◆ PHP is "type-less" language
 - ◆ Variables are not linked with type – they can store value with different types
 - ◆ No int a = 5; Just \$a = 5;
- ◆ Each variable is declared when it's first assigned value
 - ◆ This leads to problems due to typing mistakes!
 - ◆ The type of the value determines the type of the variable

- ◆ Possible PHP Variable Types are:
 - Numeric (real or integer)
 - The decimal separator is dot ".", not comma ","
 - Boolean (true or false)
 - PHP defines the constants as true, TRUE, True and false, FALSE, False
 - Empty string, zero and some other values are implicitly converted to "false" in boolean expressions
 - May cause problems when boolean not used properly

◆ String values

- ◆ Strings may be in single or double quotes

```
<?
$output1 = "Hello PHP!";
$output2 = 'Hello again!';
?>
```

- ◆ Start and end quote type should match
- ◆ Difference between two types of quotes is the escape sequences

- ◆ **Arrays are aggregate values – combination of values, each assigned a key in the array**
 - ◆ **PHP supports associative arrays – keys may be numeric, strings or any other scalar data types**
 - ◆ **Keys must be unique across the array**
 - ◆ **Values in the array may be with different types**
 - ◆ **PHP Arrays are dynamic – they don't require explicit size when created**

- ◆ PHP Array is declared with keyword **array**

```
<?
// simple array
$arr = array ("a", "b", 7);
// this produces $arr[0], $arr[1] and $arr[2]
// with values respectively "a", "b" and 7
$arr2 = array ("one" => 1, "two" => 2);
// this produces $arr2["one"] and $arr2["two"]
// with values respectively 1 and 2
?>
```

- ◆ "**=>**" means "points to"
- ◆ If keys are not supplied they are assigned automatically, starting from 0

- ◆ We access value in the array with "[" and "]" containing the key
- ◆ Arrays are flexible and types of values and keys may be mixed

```
<?
$arr = array ("a", "b", 7, "one" => 1, "two" =>
2, "other" => array(1,2,3));
// keys types may be mixed:
// $arr[0] will be "a" and $arr["one"] will be 1
// $arr["other"] is also array
// $arr["other"][0]" is 1
print $arr["other"][2]; // will output 3
?>
```

- ◆ In PHP there is special value (null) that means that the variable has no value
 - ◆ It is used to express the absence of any data type
 - ◆ Different from "undefined" variable!
 - ◆ Different from empty string or zero

```
<?
$null_variable = null;
?>
```

Variables

Live Demo

- ◆ PHP supports "object" variable type
 - ◆ Will be explained further in the OOP lecture
- ◆ "Resource" variable type
 - ◆ The resource type means the variable is holding reference to resource or data, external to your script
 - ◆ Example – opened file, database connection, etc

PHP Basic Expressions

- ◆ PHP expressions are similar to C
 - ◆ "`=`" - assigning value to variable
 - ◆ `+, -, /, *, %` - arithmetic operations
 - ◆ `==, <=, >=, !=, <, >` - comparison
 - ◆ `+=, -=, /=, *=, %=, ++, --, etc` – prefix/postfix operators
 - ◆ `(and)` – for expressions combining
 - ◆ `&, |, >>, <<, ^, ~` - bitwise operators

- ◆ String operators
 - ◆ "." (period) – string concatenating
- ◆ ===, !== comparison
 - ◆ different from ==, !=
 - ◆ "10"==10 will produce true, while "10"===10 will produce false
 - ◆ Strict comparison – **\$a === \$b** :
 - ◆ TRUE if **\$a** is equal to **\$b**, and they are of the same type.
 - ◆ Note: Assignment of value to variable returns as result the value being assigned
 - ◆ We can have **\$a = \$b = \$c = 7;**

- ◆ In PHP constants are defined with the `define` function

```
<?
define ('CONSTANT_NAME', 123);
// from here on CONSTANT_NAME will have value 123
print CONSTANT_NAME; // will output 123
?>
```

- ◆ Cannot change value
- ◆ Doesn't start with \$
- ◆ Can hold any scalar value

PHP Constants

Live Demo

Basic Functions

Phpinfo

Live Demo

- ◆ We already know **print**

- ◆ Similar to print is **echo**

```
<?
echo "123"; // will output 123 to the browser
?>
```

- ◆ **print_r(array)** – prints array with keys and values detailed
 - ◆ **phpinfo()** – Produces complete page containing information for the server, PHP settings, installed modules, etc

Basic Functions

Live Demo

Predefined Variables

Predefined Variables

- ◆ PHP provides a lot predefined variables and constants
 - ◆ `__FILE__`, `__LINE__`, `__FUNCTION__`,
`__METHOD__`, `__CLASS__` - contain debug info
 - ◆ `PHP_VERSION`, `PHP_OS`, `PHP_EOL`,
`DIRECTORY_SEPARATOR`, `PHP_INT_SIZE` and others are provided for easy creating cross-platform applications

Predefined Variables

- ◆ `$_SERVER` – array, holding information from the web server – headers, paths and script locations
 - `DOCUMENT_ROOT` – the root directory of the site in the web server configuration
 - `SERVER_ADDRESS`, `SERVER_NAME`,
`SERVER_SOFTWARE`, `SERVER_PROTOCOL`
 - `REMOTE_ADDR`, `REMOTE_HOST`, `REMOTE_PORT`
 - `PHP_AUTH_USER`, `PHP_AUTH_PW`,
`PHP_AUTH_DIGEST`
 - And others

Predefined Variables

- ◆ `$_GET`, `$_POST`, `$COOKIE` arrays hold the parameters from the URL, from the post data and from the cookies accordingly
- ◆ `$_FILES` array holds information for successfully uploaded files over multipart post request
- ◆ `$_SESSION` array holds the variables, stored in the session

Variable variables

- ◆ PHP supports \$\$ syntax- variable variables

```
<?
$str1 = 'test';
$test = 'abc';
echo $$str1; // outputs abc
?>
```

- ◆ The variable `$str1` is evaluated as 'test' and so `$$str1` is evaluated as `$test`

Predefined Variables

Live Demo

Strings Escaping

- ◆ Special chars in strings are escaped with backslashes (C style)

```
$str1 = "this is \"PHP\"";
```

- ◆ The escape sequences for double quoted string:
 - ◆ \n – new line (10 in ASCII)
 - ◆ \r – carriage return (13 in ASCII)
 - ◆ \t – horizontal tab
 - ◆ \v – vertical tab
 - ◆ \\ - backslash
 - ◆ \\$ - dollar sign
 - ◆ \" – double quote

- ◆ Single-quoted strings escape the same way

```
$str1 = 'Arnold once said: "I\'ll be back"';
```

- ◆ Difference is that instead of \" you need \' to escape the closing quotes
 - ◆ No other escaping sequences will be expanded
- ◆ In both single and double quoted strings, backslash before any other character will be printed too!

- ◆ Double quoted strings offer something more:

```
$saying = "I'll be back!";
$str1 = "Arnold once said: $saying";
// this will output:
// Arnold once said: I'll be back!
```

- ◆ Variables in double-quoted strings are evaluated
- ◆ Note on arrays:

```
$sayings = array ('arni' => "I'll be back!");
$str1 = "Arnold once said: ${sayings['arni']}";
```

- ◆ Define strings with heredoc syntax ('<<<')

```
$str = <<<EOT  
This Is the string content  
EOT;
```

- ◆ After the <<< we put "ending delimiter" – string goes all the way to this delimiter
 - ◆ The delimiter must be followed by new line
 - ◆ The ending delimiter must be alone on the last line, starting from first column
- ◆ Same escaping behavior as double-quoted string
- ◆ In single and double quoted strings you can embed new lines too

- ◆ In order to allow people to easily write large amounts of text from within PHP, but without the need to constantly escape things, heredoc syntax was developed. Heredoc might be a little tricky to understand at first, but it's actually a big help. Put simply, it allows you to define your own string limiter so that you can make it something other than a double or single quote. So, for example, we could use the string "EOT" (end of text) for our delimiter, meaning that we can use double quotes and single quotes freely within the body of the text - the string only ends when we type EOT.

Advantages and Disadvantages

Advantages and disadvantages

◆ Advantages

- Easy to learn, open source, multiplatform and database support, extensions, community and commercial driven.
- Considered to be one of the fastest languages

◆ Disadvantages

- Too loose syntax – risk tolerant, poor error handling, poor OOP (before version 6 a lot things are missing!)

HTML Forms

- ◆ The user sends data to the server only one way
 - with HTML Forms
- They are sets of fields that determine the types of data to be sent
- The server receives the filled-in data and produces new page
- To handle the submitted data you need CGI script
- The forms data is similar to arguments to a normal application

How Does It Work

username:

password:

The user enters data and submits
The form has "action" URL
to send the data to



```
<?
echo "Welcome ".$_POST['username'].";
?>
```

The PHP script receives
the data as
\$_GET and \$_POST
arrays and runs



```
...
<body>
Welcome Dimitar!
...
```

Producing HTML that is
result of the user's
posted data

GET And POST

\$_POST and \$_GET

- ◆ PHP receives the data in the **\$_GET** and **\$_POST** arrays
 - URL parameters go into the **\$_GET** array
 - Data from forms with **method="post"** do into the **\$_POST** array
 - The request method is post
 - We can check what is the current request method in the **\$_SERVER** array
 - Both arrays are global and can be used as any other array

- ◆ **\$_POST is associative array**

- ◆ The name attribute of form input becomes key in the array
- ◆ If in the example form the user fills "John" and "mypass":

```
<form method="post" action="test.php">
    <input type="text" name="mname" />
    <input type="password" name="pass" />
</form>
```

- ◆ test.php will start with built-in array **\$_POST** :
 - ◆ **\$_POST ['mname']** will be "John"
 - ◆ **\$_POST ['pass']** will be "mypass"

POST

Live Demo

- ◆ **\$_GET** is also associative array

- ◆ If we open the URL:

```
http://phpcourse.com/test.php?page=1&user=john
```

- ◆ The **test2.php** script will start with built-in array
\$_GET
 - ◆ **\$_GET ['page']** will be **1**
 - ◆ **\$_GET ['user']** will be "john"

GET
Live Demo

GET Array

Live Demo

\$_POST Versus \$_GET

- ◆ The get requests passes the parameters through the URL
 - ◆ Allows user to send link or bookmark the page as it is
 - ◆ URL is limited to 255 symbols
- ◆ The post request passes the parameters through the request body
 - ◆ User cannot open the page without first filling the post data in the form
 - ◆ Allows sending files
 - ◆ Unlimited size of data

Determine The Request Type

- ◆ `$_SERVER['REQUEST_METHOD']` holds the name of the request type
 - Can be one of 'GET', 'POST', 'HEAD', 'PUT'
 - Can be used to detect if user has submitted data or just opens the page from URL
 - Case sensitive!

Questions?