COMP519 Web Programming

Lecture 19: PHP (Part 1)

Handouts

Ullrich Hustadt

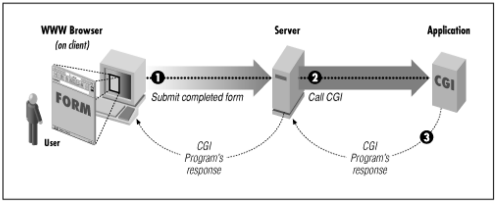
Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Common Gateway Interface — CGI

The Common Gateway Interface (CGI) is a standard method for web servers to use external applications, a CGI program, to dynamically generate web pages

1. A web client generates a client request, for example, from a HTML form, and sends it to a web server
2. The web server selects a CGI program to handle the request, converts the client request to a CGI request, executes the program
3. The CGI program then processes the CGI request and the server passes the program's response back to the client



Disadvantages of CGI

- A distinction is made between static web pages and dynamic web pages created by external CGI programs
- Using CGI programs it is difficult to add 'a little bit' of dynamic content to a web page
  - $\rightsquigarrow$ can be alleviated to some extent by 'packing' big chunks of HTML markup into a few strings
- Use of an external program requires
  - starting a separate process every time an external program is requested
  - exchanging data between web server and external program

  $\rightsquigarrow$ resource-intensive

If our main interest is the creation of dynamic web pages, then the programming language we use

- should integrate well with HTML
- should not require a web server to execute an external program

PHP

- PHP is (now) a recursive acronym for PHP: Hypertext Preprocessor
- Development started in 1994 by Rasmus Lerdorf
- Originally designed as a tool for tracking visitors at Lerdorf's website
- Developed into full-featured, scripting language for server-side web programming
- Shares a lot of the syntax and features with other languages
- Easy-to-use interface to databases
- Free, open-source
- Probably the most widely used server-side web programming language
- Negatives: Inconsistent, muddled API; no scalar objects; compatibility problems between PHP 5.x and PHP 7.x (PHP 6 was never released)

PHP Processing

- Server plug-ins exist for various web servers
  ⇝ avoids the need to execute an external program
- PHP code is embedded into HTML pages using tags
  ⇝ static web pages can easily be turned into dynamic ones

PHP satisfies the criteria we had for a good web scripting language

Processing proceeds as follows:

1. The web server receives a client request
2. The web server recognizes that the client request is for a HTML document containing PHP code
3. The server executes the PHP code, substitutes output into the HTML document, the resulting page is then send to the client

As in the case of CGI programs, the client never sees the PHP code, only the HTML document that is produced

PHP: Applications

- Applications written using PHP
  - activeCollab – Project Collaboration Software
    `http://www.activecollab.com/`
  - Drupal – Content Management System (CMS)
    `http://drupal.org/home`
  - Magento – eCommerce platform
    `http://www.magentocommerce.com/`
  - MediaWiki – Wiki software
    `http://www.mediawiki.org/wiki/MediaWiki`
  - Moodle – Virtual Learning Environment (VLE)
    `http://moodle.org/`
  - Sugar – Customer Relationship Management (CRM) platform
    `http://www.sugarcrm.com/crm/`
  - WordPress – Blogging tool and CMS
    `http://wordpress.org/`

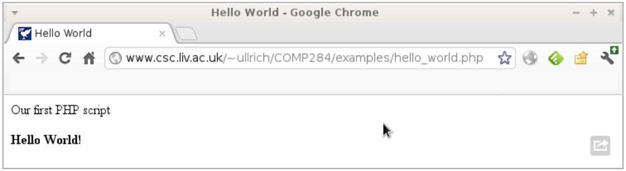
PHP: Websites

- Websites using PHP:
  - Delicious – social bookmarking
    `http://delicious.com/`
  - Digg – social news website
    `http://digg.com`
  - Facebook – social networking
    `http://www.facebook.com`
  - Flickr – photo sharing
    `http://www.flickr.com`
  - Frienster – social gaming
    `http://www.frienster.com`
  - SourceForge – web-based source code repository
    `http://sourceforge.net/`
  - Wikipedia – collaboratively built encyclopedia
    `http://www.wikipedia.org`

PHP: Hello World!

```
<html lang="en-GB">
<head><title>Hello World</title></head>
<body>
<h1>Our first PHP script</h1>
<?php
  print ("<p><b>Hello␣World!</b></p>\n");
?>
</body></html>
```

- PHP code is enclosed between `<?php` and `?>`
- File must be stored in a directory accessible by the web server, for example `$HOME/public_html`, and be readable by the web server
- File name must have the extension `.php`, e.g. `hello_world.php`


Hello World - Google Chrome
Hello World
www.csc.liv.ac.uk/~ullrich/COMP284/examples/hello_world.php

Our first PHP script

Hello World!

PHP: Hello World!

Since version 4.3.0, PHP also has a command line interface

```
#!/usr/bin/php
<?php
  /* Author: Ullrich Hustadt
     A "Hello World" PHP script. */
  print ("Hello␣World!\n");
  // A single-line comment
?>
```

```
Hello World!
```

- PHP code still needs to be enclosed between `<?php` and `?>`
- Code must be stored in an executable file
- File name does not need to have any particular format

$\leadsto$ PHP can be used to write CGI programs

$\leadsto$ PHP can be used as a scripting language outside a web programming context

PHP: Hello World!

```
<!DOCTYPE html>
<html lang="en-GB">
<head><title>Hello World</title></head>
<body><h1>Our first PHP script</h1>
<?php
  print ("<p><b>Hello␣World!</b></p>\n");
?>
</body></html>
```

- Can also ‘executed’ using

```
php filename
```

- File does not need to exectuable, only readable for the user

Output:

```
<!DOCTYPE html>
<html lang="en-GB">
<head><title>Hello World</title></head>
<body><h1>Our first PHP script</h1>
<p><b>Hello World!</b></p>
</body></html>
```

PHP Scripts

- PHP scripts are typically embedded into HTML documents and are enclosed between `<?php` and `?>` tags
- A PHP script consists of one or more statements and comments
  $\leadsto$ there is no need for a main function (or classes)
  - Statements end in a semi-colon
  - Whitespace before and in between statements is irrelevant
    (This does not mean its irrelevant to someone reading your code)
  - One-line comments start with // or # and run to the end of the line or ?>
  - Multi-line comments are enclosed in /* and */

Types

PHP has eight datatypes

- Four primitive types:

`bool`	– booleans
`int`	– integers
`float`	– floating-point numbers
`string`	– strings

- Two compound types:

`array`	– arrays
`object`	– objects

- Two special types:
  - `resource`
  - `NULL`

- Integers, floating-point numbers, and booleans do not differ significantly from the corresponding JavaScript types
- Strings differ from those in JavaScript

Integers and Floating-point numbers

- PHP distinguishes between

integer numbers	`0`	`2012`	`-40`	`1263978`
floating-point numbers	`1.25`	`256.0`	`-12e19`	`2.4e-10`

- PHP supports a wide range of pre-defined mathematical functions

`abs(number)`	absolute value
`ceil(number)`	round fractions up
`floor(number)`	round fractions down
`round(number [,prec,mode])`	round fractions
`log(number [,base])`	logarithm
`rand(min,max)`	generate an integer random number
`sqrt(number)`	square root

- PHP provides pre-defined number constants including

`M_PI`	3.14159265358979323846
`NAN`	'not a number'
`INF`	'infinity'

Integers and Floating-point numbers: NAN and INF

The constants `NAN` and `INF` are used as return values for some applications of mathematical functions that do not return a number

- `log(0)` returns `-INF` (negative 'infinity')
- `sqrt(-1)` returns `NAN` ('not a number')

In PHP 5

- `1/0` returns `FALSE` and produces a PHP warning
- `0/0` returns `FALSE` and produces a PHP warning

and execution of the script continues!

In PHP 7

- `1/0` returns `INF` and produces a PHP warning
- `0/0` returns `NAN` and produces a PHP warning

and execution of the script continues!

Booleans

- PHP has a boolean datatype
  with constants `TRUE` and `FALSE` (case insensitive)
- PHP offers the same short-circuit boolean operators as Java and JavaScript:

  `&&` (conjunction) `||` (disjunction) `!` (negation)

  - Alternatively, `and` and `or` can be used instead of `&&` and `||`, respectively
  - However, `not` is not a PHP operator
- The truth tables for these operators are the same as for JavaScript
- Remember that `&&` and `||` are not commutative, that is,
  `(A && B)` is not the same as `(B && A)`
  `(A || B)` is not the same as `(B || A)`

Type conversion to boolean

When converting to boolean, the following values are considered `FALSE`:

• the boolean	`FALSE`	
• the integer	`0`	(zero)
• the float	`0.0`	(zero)
• the string	`'0'`	(but not `0.0` nor `'00'`)

- the empty string `''`
- an array with zero elements
- an object with zero member variables (PHP 4 only)
- the special type `NULL` (including `unset` variables)
- SimpleXML objects created from empty tags

Every other value is considered `TRUE` (including any resource)

When converting a boolean to a string,

- `TRUE` becomes `"1"`
- `FALSE` becomes `""`

Strings

- PHP supports both single-quoted and double-quoted strings
- PHP also supports heredocs as a means to specify multi-line strings

```
<<<identifier
here document
identifier
```

  - *identifier* might optionally be surrounded by double-quotes
  - *identifier* might also be surrounded by single-quotes, making the string a nowdoc in PHP terminology

```
print "<html␣lang=\"en-GB\">
<head><title>Multi-line␣String</title></head>";

print<<<EOF
<body>Some text
<img alt="Picture␣of␣Crowne␣Plaza" src="pic.png">
</body>
</html>
EOF;
```

Strings

PHP distinguishes between

- single-quoted strings and
- double-quoted strings

single-quoted strings ('taken literally')			double-quoted strings ('interpreted'/'evaluated')		
`'hello'`	⇝	`hello`	`"hello"`	⇝	`hello`
`'don\'t'`	⇝	`don't`	`"don't"`	⇝	`don't`
`'"hello"'`	⇝	`"hello"`	`"\"hello\""`	⇝	`"hello"`
`'backslash\\'`	⇝	`backslash\`	`"backslash\\"`	⇝	`backslash\`
`'glass\\table'`	⇝	`glass\table`	`"glass\\table"`	⇝	`glass\table`
`'glass\table'`	⇝	`glass\table`	`"glass\table"`	⇝	`glass    able`

Strings

- Variable interpolation is applied to double-quoted strings

```
$title  = "String␣Operators";
print "<title>$title</title>";
```

```
<title>String Operators</title>
```

- The string concatenation operator is denoted by '.'
- The string multiplication / repetition operator in PHP is

```
string str_repeat(string_arg, number)
```

```
$string = "<p>I␣shall␣not␣repeat␣myself.<p>\n";
print "<body>" . str_repeat($string,3) . '</body>';
```

```
<body><p>I shall not repeat myself.<p>
<p>I shall not repeat myself.<p>
<p>I shall not repeat myself.<p>
</body>
```

Variables

- All PHP variable names start with $ followed by a PHP identifier
- A PHP identifier consists of letters, digits, and underscores, but cannot start with a digit
  PHP identifiers are case sensitive
- In PHP, a variable does not have to be declared before it can be used
- A variable also does not have to be initialised before it can be used, although initialisation is a good idea
- Uninitialized variables have a default value of their type depending on the context in which they are used

Type	Default	Type	Default
`bool`	`FALSE`	`string`	empty string
`int`/`float`	0	`array`	empty array

If there is no context, then the default value is `NULL`

Assignments

- Just like Java, JavaScript and Python, PHP uses the equality sign = for assignments

  ```
  $student_id = 200846369;
  ```

  As in JavaScript, this is an assignment expression
- The value of an assignment expression is the value assigned

  ```
  $b = ($a = 0) + 1;
  // $a has value 0
  // $b has value 1
  ```

Binary Assignments

PHP also supports the standard binary assignment operators:

Binary assignment	Equivalent assignment
`$a += $b`	`$a = $a + $b`
`$a -= $b`	`$a = $a - $b`
`$a *= $b`	`$a = $a * $b`
`$a /= $b`	`$a = $a / $b`
`$a %= $b`	`$a = $a % $b`
`$a **= $b`	`$a = $a ** $b`
`$a .= $b`	`$a = $a . $b`

```
// Convert Fahrenheit to Celsius:
// Subtract 32, then multiply by 5, then divide by 9
$temperature = 105;          // temperature in Fahrenheit
$temperature -= 32;
$temperature *= 5;
$temperature /= 9;           // converted to Celsius
```

Constants

- `bool define(`*string*`,` *expr* `[,` *case_insensitive*`])`
  - defines a constant that is globally accessible within a script
  - *string* should be a string consisting of a PHP identifier (preferably all upper-case)
    The PHP identifier is the name of the constant
  - *expr* is an expression that should evaluate to a value of a scalar type
    (In PHP 7, *expr* can also be an array)
  - *case_insensitive* is an optional boolean argument, indicating whether the name of the constant is case-insensitive (default is FALSE)
  - returns TRUE on success or FALSE on failure

```
define("PI",3.14159);
define("SPEED_OF_LIGHT",299792458,true);
// PHP 7
define("ANIMALS",["bird","cat","dog"]);
```

Constants

- To use a constant we simply use its name

```
define("PI",3.14159);
define("SPEED_OF_LIGHT",299792458,true);
// PHP 7
define("ANIMALS",["bird","cat","dog"]);

$circumfence = PI * $diameter;
$distance    = speed_of_light * $time;
$myPet       = ANIMALS[1];
```

- Caveat: PHP does not resolve constants within double-quoted strings (or here documents)

```
print "1␣-␣Value␣of␣PI:␣PI\n";
1 - Value of PI: PI
print "2␣-␣Value␣of␣PI:␣" . PI . "\n";
2 - Value of PI: 3.14159
```

Values, Variables and Types

PHP provides several functions that explore the type of an expression:

string gettype(*expr*)	returns the type of *expr* as string
bool is_*type*(*expr*)	checks whether *expr* is of type *type*
void var_dump(*expr*)	displays structured information about *expr* that includes its type and value

```
<?php print "Type␣of␣23:␣␣␣".gettype(23)."\n";
      print "Type␣of␣23.0:␣".gettype(23.0)."\n";
      print "Type␣of␣\"23\":␣".gettype("23")."\n";

      if (is_int(23)) { echo "23␣is␣an␣integer\n"; }
         else { echo "23␣is␣not␣an␣integer\n"; }
?>
```

```
Type of 23:   integer
Type of 23.0: double
Type of "23": string
23 is an integer
```

Type juggling and Type casting

- PHP automatically converts a value to the appropriate type as required by the operation applied to the value (type juggling)

```
2 . "␣worlds"          ⇝  "2␣worlds"
"2" * 3                ⇝  6
"1.23e2" + 0           ⇝  123
"hello" * 3            ⇝  0          (in PHP 7 also a warning)
"10hello5" + 5         ⇝  15         (in PHP 7 also a warning)
```

- We can apply an identity function of the target type to force a type conversion

`"12" * 1`	⇝	`12`	`!!1`	⇝	`TRUE`	`!!"1"`	⇝	`TRUE`
`"12" * 1.0`	⇝	`12.0`	`!!0`	⇝	`FALSE`	`!!"0"`	⇝	`FALSE`
`"12.1" * 1`	⇝	`12.1`	`!!1.0`	⇝	`TRUE`	`!!""`	⇝	`FALSE`
`12 . ""`	⇝	`"12"`	`!!0.0`	⇝	`FALSE`	`FALSE . ""`	⇝	`""`
`12.1 . ""`	⇝	`"12.1"`				`FALSE * 1`	⇝	`0`

Conversion of arrays to strings or numbers does not work

Type juggling and Type casting

- PHP also supports explicit type casting via `(`*type*`)`

`(int) "12"`	$\leadsto$	`12`	
`(int) "10hello5"`	$\leadsto$	`10`	
`(int) "1.23e2"`	$\leadsto$	`1`	in PHP 5
`(int) "1.23e2"`	$\leadsto$	`123`	in PHP 7
`(int) ("1.23e2" * 1)`	$\leadsto$	`123`	in both PHP 5 and 7
`(int) (float) "1.23e2"`	$\leadsto$	`123`	in both PHP 5 and 7
`(int) "1.23e2h5"`	$\leadsto$	`1`	in PHP 5
`(int) "1.23e2h5"`	$\leadsto$	`123`	in PHP 7
`(int) 10.5`	$\leadsto$	`10`	
`(float) "1.23e2"`	$\leadsto$	`123.0`	
`(float) "1.23e2h5"`	$\leadsto$	`123.0`	
`(bool) "0"`	$\leadsto$	`FALSE`	(was `true` in JavaScript)
`(bool) "foo"`	$\leadsto$	`TRUE`	
`(array) "foo"`	$\leadsto$	`array(0 => "foo")`	

Revision and Further Reading

- Read
  - Chapter 3: Introduction to PHP
  - Chapter 4: Expressions and Control Flow in PHP: Expressions

  of R. Nixon: Learning PHP, MySQL & JavaScript: with jQuery, CSS & HTML5. O'Reilly, 2018.
- Read
  - Language Reference: Types: Booleans
    `http://uk.php.net/manual/en/language.types.boolean.php`
  - Language Reference: Types: Integers
    `http://uk.php.net/manual/en/language.types.integer.php`
  - Language Reference: Types: Floating Point Numbers
    `http://uk.php.net/manual/en/language.types.float.php`
  - Language Reference: Types: Strings
    `http://uk.php.net/manual/en/language.types.string.php`

  of P. Cowburn (ed.): PHP Manual. The PHP Group, 25 Oct 2019. `http://uk.php.net/manual/en` [accessed 26 Oct 2019]