CSCB20 – Week 8

Introduction to Database and Web Application Programming

Anna Bretscher*

Winter 2017

*thanks to Alan Rosselet for providing the slides these are adapted from.

Web Programming

We have seen how HTML and CSS work together to create a Web page (HTML) with styling details applied (CSS)

When you type a URL for an HTML document into a browser window:

- browser sends a request to the server (cmslab/mathlab in this case),
- the server locates the requested file (test.html),
- sends that file back to the browser to be rendered

Rather than providing the name of a file in a URL, it is possible to give the name of a program:

- server executes the program
- sends its output back to the browser to be rendered, e.g.:

`https://mathlab.utsc.utoronto.ca/…/hello.php`

What is PHP?

PHP == 'PHP Hypertext Preprocessor'.

- Free and open-source, server-side scripting language designed specifically for the Web
- Used to generate dynamic web-pages
- Supported by most Web servers

PHP scripts

- are bracketed by reserved PHP tags
- supports embedding of PHP scripts within HTML pages
- easy to learn operational behavior and common patterns for working with Web pages

PHP Overview (cont'd)

- Interpreted language
- Scripts are parsed at run-time rather than compiled beforehand
- Executed on the server-side
- Source-code not visible to client
- 'View Source' in browsers does not display PHP code, only output produced by PHP code
- Various built-in functions allow for fast development
- Compatible with many popular databases

PHP Overview

- LAMP (Linux, Apache, MySQL, PHP) is a common Web application platform – all components are free, open-source
- Syntax Perl- and C-like syntax. Relatively easy to learn
- Large function library
- Embedded directly into HTML
- Interpreted, no need to compile
- Loosely typed, like Python and JavaScript

Why PHP?

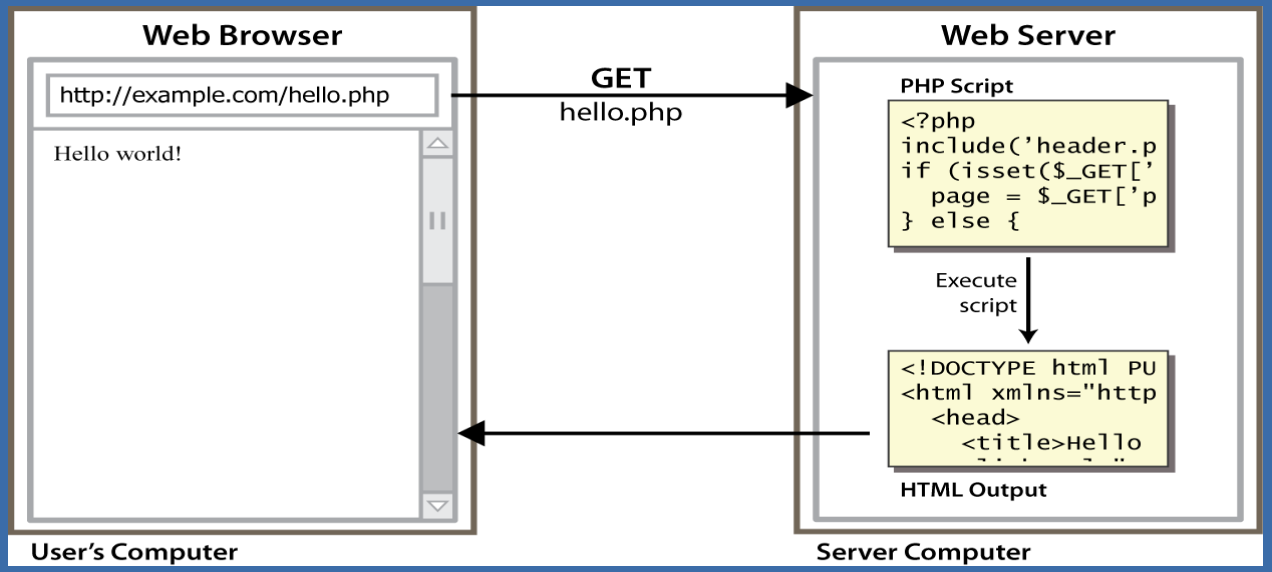
PHP is but one of many server-side languages for developing dynamic Web app's, other options include:

- Java Servlets with JSP, Ruby on Rails, ASP .Net

Why choose PHP?

- easy deployment – no complex infrastructure to set up
- compatible: supported by most popular Web servers
- simple: lots of built-in functionality; familiar syntax
- free and open source: anyone can run a PHP-enabled server free of charge
- available: installed on most commercial Web hosts, and on UTSC servers, including mathlab

PHP Web Page Request Lifecycle



browser requests .html file (static content);

- server sends file content

browser requests .php file (dynamic content);

- server reads file,
- executes any embedded script content,
- sends output of script back to browser.

What does PHP code look like?

- Supports procedural and object-oriented paradigms
- All PHP statements end with a semi-colon
- Each PHP script must be enclosed in the reserved PHP tag, denoted by

```
<?php
    ...
?>
```

- PHP code block may contain statements, function definitions, variable-value references

Hello World in PHP

```
<!-- hello.php -->
<html><body>
    <strong>Hello World!</strong><br />
    <?php print "<h2>Hello, World</h2>"; ?>
</body></html>
```

Output generated by PHP "`print`" and "`echo`" statements is inserted into the HTML returned to the browser.

Q. How do you view the output of a PHP script from a browser?

Place `hello.php` in your cscb20w17_space directory, then view in a browser with URL:

https://mathlab.utsc.utoronto.ca/courses/cscb20w17/UTORid/hello.php

Variables in PHP

- PHP variables begin with a "`$`" sign, both for declaration and value reference
- Case-sensitive (`$Foo` != `$foo` != `$f0o`)
- Global and locally-scoped variables
  - global variables can be used anywhere
  - local variables restricted to a function or class
- Certain variable names reserved by PHP
  - Form variables ($_POST, $_GET)
  - Server variables ($_SERVER)
  - Etc.

Variable Usage and Comments

```
<?php
$foo = 25;              // Numerical variable
$bar = "Hello";         // String variable

$foo = ($foo * 7);      // Multiplies foo by 7
$bar = ($bar * 7);      // Invalid expression
?>
```

single-line comments are written as one of:

```
// single-line comment
# single-line comment
```

multi-line comments bracketed by

```
/* multi-line comment ...
 */
```

Data Types

PHP is a loosely-typed language, like Python

PHP basic types are:

- int, float, boolean, string, array, object, NULL
- functions `is_type()` test whether a variable has a certain type, e.g. `is_string($myvar)`

Conversion between types is automatic in many cases, e.g.:

- string to int for "+"
- int to float for "/"

Types can be "cast" to another type using:

```
$int_val = (int) "33";
```

Strings

```
$myvar = "hello";
print $myvar[1];     # prints “e”
```

square bracket notation for 0-based indexing

concatenation using “.” operator (not “+”)

```
print $myvar . "world"; # prints hello world
```

strings quoted with double quotes are “interpreted”, meaning that embedded variables have their values inserted

strings quoted with single quotes are not interpreted

```
print "$myvar world"; # prints hello world
print '$myvar world'; # prints $myvar world
```

for loop

```
for (initialization; condition; update) {
    statements
}
```

uses same syntax as Java

```
for ($i = 10; $i >= 0; $i--) {
    print "$i cubed is " . $i * $i * $i . ".\n";
}
```

`for` loop

```
for (initialization; condition; update) {
    statements
}
```

uses same syntax as Java

```
$name = "preprocessor";
for ($i = 0; $i < strlen($name); $i++) {
    print "The next letter is".{$name[$i]}."\n";
}
```

`if/else` statement

```
if (condition) {
    statements;
} elseif (condition) {
    statements;
} else {
    statements;
}
```

```
<?php
if ($user=="John") {
   print "Hello John.";
}
else {
   print "You aren't John.";
}
?>
```

- `elseif` clause and `else` clause are both optional
- multiple `elseif` clauses may be used

`while` loop

same syntax as Java

```
while (condition) {
      statements;
}
```

or

```
do {
      statements;
} while (condition)
```

```
<?php
  $count=0;
  while($count<3) {
       print "hello PHP. ";
       $count += 1;
       // or
       // $count = $count + 1;
       // or
       // $count++;
    }
?>
```

```
hello PHP. hello PHP. hello PHP.
```

Arrays (ie, lists)

```
$myvar = array(); # create new array
$myvar = array(val0, val1, ..., valN);

$myvar[index];    # element at position index
$myvar[index] = val0;  # assign element at index
$myvar[] = valN;  # append valN

$a1 = array();  # empty, length-0 array
$a[2] = 12;  # store 12 in 3rd position of array
$a2 = array("a", "sequence", "of", "strings");
$a2[] = "the end";  # new last element of $a2
```

`foreach` loop

```
foreach ($array as $element) {
        ...
}
```

Similar to Python's: `for element in array:`

Simpler than regular "for" loop when using arrays

```
$a = array("a", "sequence", "of", "strings");
for ($i = 0; i < count($a); $i++) {
        print "the next word is {$a[$i]}\n";
}
foreach ($a as $element) {
       print "the next word is $element\n";
}
```

Embedded PHP

We could use PHP print and/or echo statements to generate HTML output, e.g.

```
<?php
    print "<html>\n<head>\n";
    print "<title>PHP Squares</title>\n";
    ...
    for ($i = 0; i <= 10; $i++) {
  print "<p>$i squared is $i*$i</p>\n";
    }
?>
```

What's wrong with this approach?

Suppose you want to change the page HTML …

Embedding PHP in HTML

Write HTML literally.

When scripting is needed to compute a value, embed PHP code.

General format of a PHP script written within HTML file:

```
HTML elements ...  <!-- output as HTML -->
  <?php
    PHP code ... # output embedded within HTML
  ?>
HTML elements ...
  <?php
    PHP code ...
  ?>
HTML elements ...
```

Embedding PHP in HTML

General format of a PHP script written within HTML file:

```
HTML elements ...  <!-- output as HTML -->
  <?php
   PHP code ... # output embedded within HTML
  ?>
HTML elements ...
```

The PHP code in an embedded block may consist of statements, declarations, or expression values.

Here's a sample expression block:

```
<?= $myvar ?>
```

which is equivalent to

```
<?php print $myvar; ?>
```

Embedding PHP in HTML

Here's our earlier "squares" calculator, with "poor style" print statements:

```
<?php
    print "<html>\n<head>\n";
    print "<title>PHP Squares</title>\n";
    ...
    for ($i = 0; i <= 10; $i++) {
      print "<p>$i squared is $i*$i</p>\n";
    }
?>
```

Embedding PHP in HTML

Here's our earlier "squares" calculator, now without print statements:

```
<html><head>
  <title>PHP Squares</title>
    ...
  <?php
  for ($i = 0; $i <= 10; $i++) { ?>
      <p><?= $i ?> squared is <?= $i*$i ?>
      </p>
  <?php } ?>
    ...
</html>
```

Functions

Functions must be defined before they can be called.

Function headers are of the format:

```
function functionName($arg_1, $arg_2, …, $arg_n)
```

Note that no return type is specified.

```
function quadratic($a, $b, $c) {
   return -$b + sqrt($b*$b - 4*$a*$c) / (2*$a);
}
$x = -2; $y = 3; $root = quadratic(1, $x, $y-2);
```

Unlike variables, function names are not case sensitive

```
foo(…) == Foo(…) == FoO(…)
```

Query Strings and Parameters

- We refer to Web pages using URL's (Uniform Resource Locators), of the form http://domain_name/path_value

- We can specify parameters to PHP scripts by appending a value to the end of the URL:

  http://www.google.com/search?q=android

  https://mathlab…./cscb20w17/utorid/fresh.php?film=sing

- Parameter name=value pairs follow the "?" at the end of the URL path_value, in 2nd example param name is film, value is sing

- Provides a mechanism by which a user can control/customize the behavior of a server-side PHP script

Query Strings and Parameters

- PHP can retrieve parameter values using the $_REQUEST array:

```
$_REQUEST["parameter_name"]
```

- Returns the parameter's value as a string

- Can check to see if a specific parameter is set using `isset()`:

```
$country_name = $_REQUEST["country"];
$population = (int) $_REQUEST["population"];
if (isset($_REQUEST["code"])) {
    $code = (int) $_REQUEST["code"];
} else {
    $code = -1;
}
```

Reading Files

Two ways to read the contents of a text file:

1. `file("my_file.txt");`

   returns array of lines in my_file.txt

2. `file_get_contents("my_file.txt");`

   returns a single string containing all lines in my_file.txt

```
<?php
# display lines from file as a bulleted list
$cities = file("cities.txt");
foreach ($cities as $city) {
?>
   <li> <?= $city ?> </li>
<?php
}
?>
```

Unpacking Arrays, Splitting Strings

Sometimes it is useful to be able to refer to the elements of an array by individual variable names, rather than using indices

```
$movie_info = array("Sing", "2016");
list($title,$year) = $movie_info;
print "$title opened in $year.";
# now can use $title rather than $movie_info[0]
```

A string consisting of delimited values can be split (same idea as in Python)

```
$title = "Databases and Web Programming";
$words = explode(" ", $title);
```

Reading Directories

- If your application needs to read from a set of files in a directory, how can your code automatically detect and read the specific files present?
- `glob` enables you to use pattern matching to select files

```
$notes = glob("note_*.txt");
foreach ($notes as $note) {
   print $note;
}
```

- `*` is just one of several "regular expression" pattern-matching forms (others include matching on character ranges, matching digits, optional characters)