PHP

CSE 190 M (Web Programming) Spring 2007
University of Washington

Reading: Sebesta Ch. 12

What is PHP?

- PHP stands for "PHP Hypertext Preprocessor"
- an HTML-embedded server-side scripting language
- used to make web pages dynamic
    - process form information
    - authenticate users
    - provide different content depending on context
    - interface with other services: database, e-mail, etc
- generates HTML and/or client-side scripts sent to client browsers
- similar syntax to Javascript

Why PHP?

- many other options: ASP.NET, ColdFusion, JSP...
- PHP is:
    - free and open source: anyone can run a PHP-enabled server
    - compatible: supported by most popular web servers
    - simple: lots of built-in functionality; familiar syntax
    - installed on UW's dante server

Why use PHP instead of Javascript?

- PHP has access to server's important and/or private data
- avoids many browser JS compatibility issues
- faster for users (doesn't have to run a script to view each page)
- client can't see your source code
- fewer security restrictions (can write to files, open web pages on other servers, connect to databases, ...)

Similarities between PHP and Javascript

- interpreted
- relaxed syntax and rules
    - "loose" data types
    - variables don't need to be declared (initialized to `NULL`)
- variable names case sensitive
- built-in regular expressions

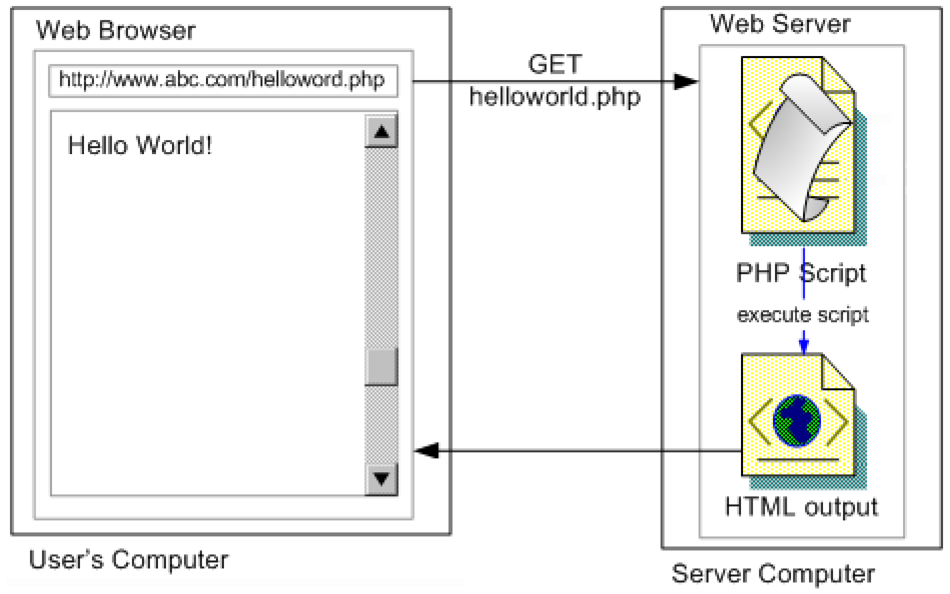
Differences between PHP and Javascript

- PHP is more procedural and geared more toward text processing
    - `verb(noun)` rather than `noun.verb()`
- variable names have a `$` prefix
- end-of-line semicolon is required
- `.` string concatenation operator
- `elseif` keyword

PHP files

- generally have `.php` or `.phtml` extensions
- generally contain both HTML and PHP
- when a client views the source, only HTML is visible
- all PHP script blocks start with `<?php` and end with `?>`

A typical web server request using PHP



- browser requests a `.html` file (static content): server just sends that file
- browser requests a `.php` file (dynamic content): server reads it, runs any script code inside it, then sends result across the network
  - script produces output that becomes part of the HTML page

Hello, World!

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Hello world</title>
<meta http-equiv="Content-Type"
 content="text/html; charset=iso-8859-1" />
</head>
<body>

<?php
print("Hello, World");
?>

</body>
</html>
```

System information: phpinfo()

```
<?php
phpinfo();
?>
```

- a typical way to test an installation
- good way to find out where configuration files are
- describes built-in variables
- lists which modules are enabled

Variables

```
$name = value;
```

```
$username = 'pinkHeartLuvr78';
$age = 16;
$thisClassRocks = TRUE;
$éléphant = "totally legit variable name";
```

- names are case sensitive
- always implicitly declared through assignment
- like Javascript, a "loosely typed" language
- `gettype`, `settype` functions access and modify a variable's type (generally set as needed)

Injecting text: print()

```
print("text");
```

```
print("Hello, World!");
print("Escape \"chars\" are the SAME as in Java!\n");

print("You can have
line breaks in the string
and they'll show up");

print('A string can use single-quotes.  It\'s cool!');
```

Interpreted strings

```
print("This will print the variable's value: $var");
print('This will print the variable\'s name: $var');
```

- strings inside `""` are interpreted
  - variables that appear inside them will have their values inserted into the string
  - a simpler syntax than string concatenation;
    PHP was designed so that it would be easy to format and output text
- strings inside `' '` are not interpreted

Comments

```
# single-line comment

# another single-line comment style

/*
multi-line comment
*/
```

- like Java and Javascript but # is also allowed
  - a lot of PHP code uses # comments instead of `//`

Operators

- `+ - * / % . ++ -- = += -= *=`
  `/= %= == != > < >= <= && || !`

- `==` just checks value (`"5.0" == 5` is `true`)
- `===` also checks type (`"5" === 5` is `false`)
- many operators auto-convert types: `5 < "7"` is `true`
- NOTE: concatenation operator is `.` (the dot character), not `+`

`for` loop

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

Write a loop that prints out squares from 0 - 81 like this: "0 squared is 0."

```
for($i = 0; $i < 10; $i++) {
    print("<p> $i squared is " . $i * $i . " </p>");
}
```

Including scripts: include()

```
include("filename");
```

```
include("header.php");
```

- inserts the entire contents of the given file into the PHP script's output page
- encourages modularity
- useful for defining reused functions like form-checking

String type

```
$favoriteFood = "ethiopian";
$favoriteFood[2];            # evaluates to "h"
```

- concatenation is done with `.`
- zero-based indexing using bracket notation
- when specified with `""`, variables and escaped chars are interpreted, but not using `''`
- functions (complete list)
  - `explode`, `implode`, `strlen`, `strcmp`, `strpos`, `substr`, `strtolower`, `strtoupper`, `trim`
  - for example, to get length of a string:
    `$length = strlen($favoriteFood);`

String functions

```
$name = "Kenneth Kuan";
$length = strlen($name);              # 12
$cmp = strcmp($name, "Jeff Prouty");  # > 0
$index = strpos($name, "e");          # 1
$first = substr($name, 8, 4);         # "Kuan"
$upper = strtoupper($name);           # "KENNETH KUAN"
```

Name	Javascript or Java Name
explode, implode	split, join
strlen	length
strcmp	compareTo
strpos	indexOf
substr	substring
strtolower, strtoupper	toLowerCase, toUpperCase
trim	substring

Numbers

```
$piApprox = 355/113;          # double: 3.141592920354
(int) $piApprox;              # int: 3
round($piApprox);             # double: 3
```

- `int` for integers and `double` for reals
- use `intval()` to convert a `String` into an `int`
- result of division between two `int` values can have type `double`

Mathematics

- functions:
  - abs, ceil, floor, max, min, rand, round, srand...
- constants:
  - M_PI, M_E, M_LN2

Boolean type

```
$feelsLikeSummer = FALSE;
$phpIsRad = True;
$studentCount = 96;
(bool) $studentCount;     # evaluates to TRUE
```

- both `TRUE` and `FALSE` keywords are case insensitive
- the following values are considered to be `FALSE` (all others are `TRUE`):
  - (int) `0`
  - (double) `0.0` (but NOT `0.00` or `0.000`!)
  - `""` (the empty string) and `"0"`
  - arrays with no elements
  - `NULL` (includes unset variables)
- can cast to boolean using `(bool)`

NULL

- a variable is `NULL` if
  - it has been assigned the constant `NULL`
  - it has not been set to any value
  - it has been `unset()`
- can test if a variable is NULL by using `isset()`

`if/else` statement

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

- NOTE: although `elseif` keyword is much more common, `else if` is also supported

`while` loop

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

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

- `break` and `continue` keywords also behave as in Java

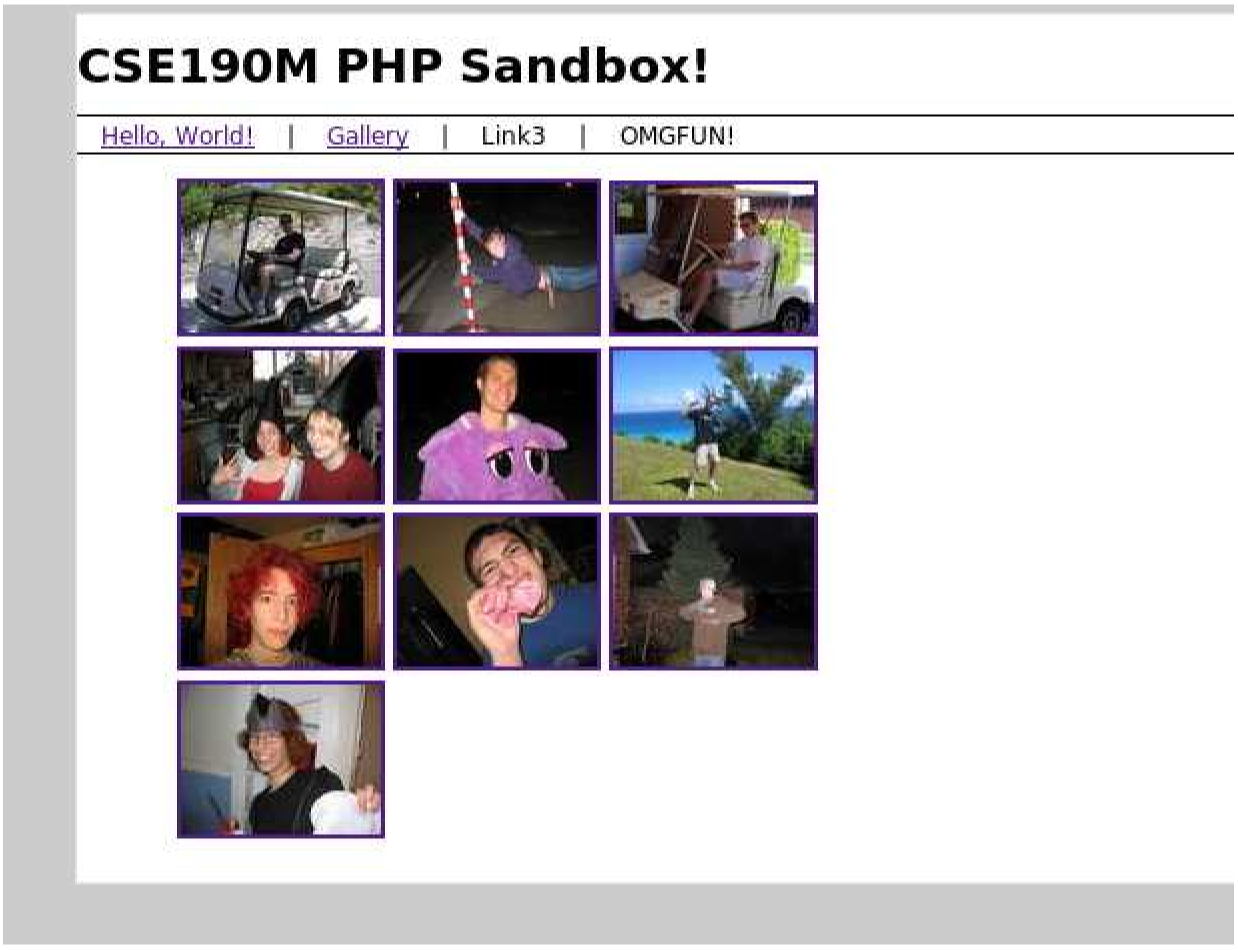
Reading directories

```
$DIR = "awesomeFiles";
$dh = opendir($DIR);
while ($file = readdir($dh)) {
    print("$file<br>\n");
}
closedir($dh);
```

- `opendir()` - begins reading a directory and returns a reference to it
- `readdir()` - reads one file name from the directory reference
- `closedir()` - stops reading the directory

Practice problem: image gallery

Given a directory called `thumbs` containing picture thumbnails and a directory called `images` which contains the full images, create a page that displays the thumbnails. These should link to their corresponding full-sized image. The filenames are the same in the two directories.



Functions

```
function name(parameterName, ..., parameterName) {
    statements;
}
```

```
function quadratic($a, $b, $c) {
    return -$b + sqrt($b*$b - 4*$a*$c) / (2*$a);
}
```

- parameter types and return types are not written
- any variables declared in the function are local (only exist in that function)

Calling functions

```
name(parameterValue, ..., parameterValue);
```

```
$root = quadratic(1, $x, $a + 3);
```

Arrays

```
$name = array(value0, value1, ..., valueN);   # create
$name[] = value;                             # create or append
$name[index] = value;                         # set element value
$name[index]                                 # get value
```

```
$arr[] = 23;       # creates array with 23 at index 0
$arr2 = array("some", "strings", "in", "an", "array");
$arr2[] = "Ooh!";  # add string to end (at index 5)
```

- created by assignment
- to append, use bracket notation without specifying an index
- type is not specified; can mix types

`foreach` loop

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

```
$stooges = array("Larry", "Moe", "Curly", "Shemp");
foreach ($stooges as $stooge) {
    print("<p>Moe slaps $stooge</p>");  # even himself!
}
```

- a convenient way to loop over each element of an array without indexes

Array functions

- `count` : number of elements in the array
- `print_r` : print array's contents
- using an array as a list:
  `array_pop`, `array_push`, `array_shift`, `array_unshift`
- reordering an array:
  `array_reverse`, `in_array`, `rsort`, `shuffle`, `sort`
- creating, filling, filtering an array:
  `array_fill`, `array_merge`, `array_slice`, `array_unique`, `range`

Array function example

```
$tas = array("MD", "BH", "KK", "HM", "JP");
for ($i = 0; $i < count($tas); $i++) {
    $tas[$i] = strtolower($tas[$i]);
}                                  # ("md", "bh", "kk", "hm", "jp")
$morgan = array_shift($tas);       # ("bh", "kk", "hm", "jp")
array_pop($tas);                   # ("bh", "kk", "hm")
array_push($tas, "ms");            # ("bh", "kk", "hm", "ms")
array_reverse($tas);               # ("ms", "hm", "kk", "bh")
sort($tas);                        # ("bh", "hm", "kk", "ms")
$best = array_slice($tas, 1, 2);   # ("hm", "kk")
```

Regular expressions in PHP (PDF)

- syntax: strings that begin and end with `/`, such as `"/[AEIOU]+/"`
- `preg_match(pattern, string)`
  returns `TRUE` if the given string contains the given pattern
  - for a case-insensitive match, place an `i` at end of regular expression (after closing `/`)
- `preg_replace(pattern, replacement, string)`
  returns new string with first occurrence of pattern replaced by replacement
  - to replace all occurrences, place a `g` at end of regular expression (after closing `/`)
- `preg_split(pattern, string)`
  returns array of strings from given string broken apart by given pattern
- complete list

Regular expression example 1

```
$str = "the quick    brown  fox";
$words = preg_split("/[ ]+/", $str);
                         # ("the", "quick", "brown", "fox")

for ($i = 0; $i < count($words); $i++) {
    $words[$i] = preg_replace("/[aeiou]/g", "*", $words[$i]);
}
                         # ("th*", "q**ck", "br*wn", "f*x")

$str = implode("_", $words);
                         # "th*_q**ck_br*wn_f*x"
```

Regular expression example 2

```
$str = "<10><20><30><40>";
if (preg_match("/(<\d\d>){4}/", $str)) {
    $str = preg_replace("/[<>]+/", ",", $str);
    $tokens = preg_split("/0/", $str);
    foreach ($tokens as $num) {
        print("Number: $num\n");
    }
}
```

- What is the value of `$str` after the `preg_replace` call?
- What elements are stored in `$tokens` ?

Reading files

```
$text = file_get_contents("filename");
$lines = preg_split("/\n/", $text);
foreach ($lines as $line) {
    do something with $line;
}
```

- `file_get_contents` returns entire contents of a file as a large string
- often these contents are split into an array of lines using `preg_split`
- `file_set_contents` writes a string into a file

Reading files example

```
# Returns how many lines in this file are empty or just spaces.
function count_blank_lines($file_name) {
    $text = file_get_contents($file_name);
    $lines = preg_split("/\n/", $text);
    $count = 0;
    foreach ($lines as $line) {
        if (strlen(trim($line)) == 0) {
            $count++;
        }
    }
    return $count;
}
...
print(count_blank_lines("15-php.html"));
```

Query parameters: `$_GET` and `$_POST`

```
$cc = $_GET["creditcard"];        # if it is a GET request
$username = $_POST["username"];   # if it is a POST request
```

- many PHP scripts are used to handle data from HTML forms
- `$_GET["parameter name"]` returns the value of the query parameter with that name, if the browser made a GET request
- `$_POST["parameter name"]` returns the value of the query parameter with that name, if the browser made a POST request
- `$_GET` and `$_POST` are called associative arrays or maps (seen later)

Checking for a parameter's existence

```
if (array_key_exists("creditcard", $_GET)) {
    $cc = $_GET["creditcard"];
    ...
} else {
    print("Error, you did not submit a credit card number.");
    ...
    return;
}
```

- the `array_key_exists` function returns `TRUE` if the `$_GET` or `$_POST` has a parameter with the given name
- can abort the rest of a PHP block using `return` or `exit` function

Headers

- by default, a PHP script's output is HTML and its result code is 200 (success)
- use the `header` function if you need to output other data or result codes
  - must appear before any other output generated by the script
- examples:
  - `header("Content-type: text/plain");`
  - `header("Content-type: application/xml");`
  - `header("HTTP/1.1 400 Invalid Request");`
  - `header("HTTP/1.1 404 File Not Found");`
  - `header("HTTP/1.1 500 Server Error");`

Practice problem: Baby Names server

Write a PHP script that mimics the Baby Names server app used in Homework 5. Have your script accept a query parameter named `type` that is either set to `list`, `meaning`, or `rank`.

- If `type` is `list`, display an HTML page with the entire contents of the file `list.txt`.
- If `type` is `meaning`, also accept a parameter named `name`. Search the file `meanings.txt` for the line associated with that name and display it.
- If `type` is `rank`, also accept a parameter named `name`. Search the file `rank.txt` for the line associated with that name and display its ranking data as text or as XML.