Software Engineering

LECTURE:
PHP Scripting Language

Ivan Marsic
Rutgers University

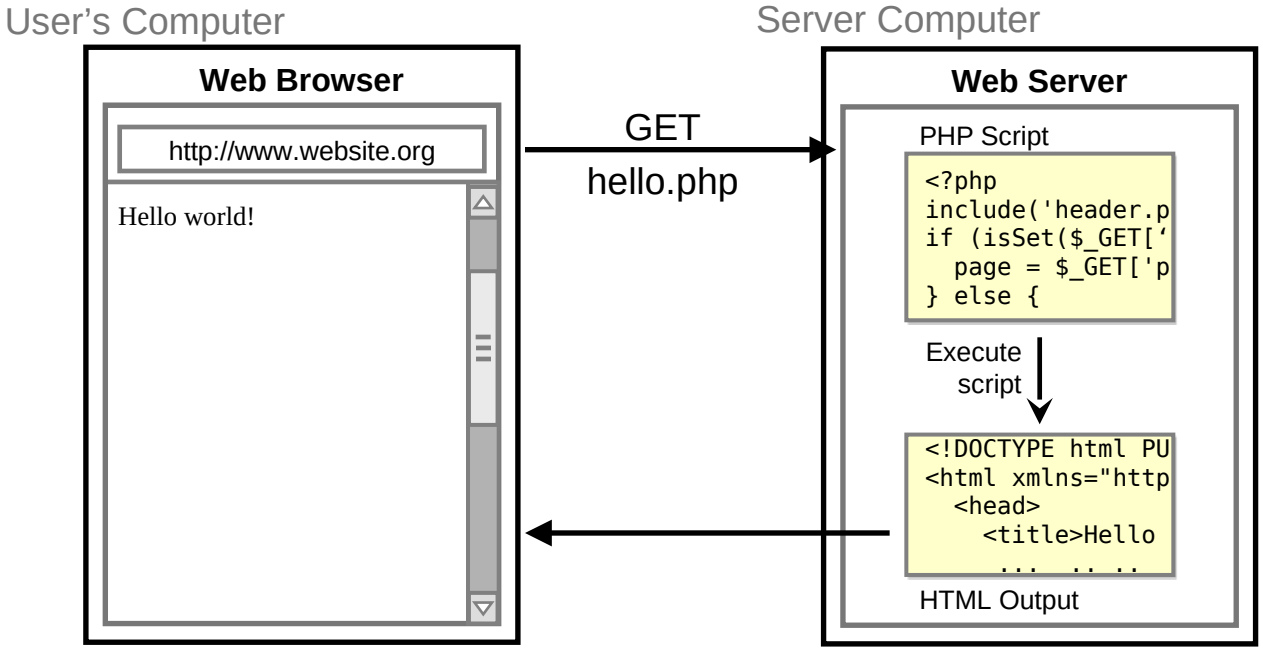
Topics

- Syntax, Variables, Types
- Operators, Expressions, Math Functions
- String Operations and Functions
- PHP Processor Output
- Control Statements
- PHP in HTML
- Arrays
- User-Defined Functions
- Pattern Matching
- Form Handling
- MySQL Database Access from PHP
- Cookies and Session Tracking

What is PHP?

- Originally an acronym for Personal Home Page that later became
  PHP: Hypertext Preprocessor
- PHP is a server-side scripting language whose scripts are embedded in HTML documents
  - Alternatives:
    JSP, Ruby on Rails, Python, ASP.NET, etc.
  - The web server contains software that allows it to run those programs and send back their output (HTML documents) as responses to client requests
- Server-side scripting languages are used for
  - HTML form handling
  - User authentication
  - File processing
  - Database access
  - Other services, e.g., email, …

Lifecycle of a PHP Web Request



- 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
  - PHP script outputs an HTML document that is sent back
- PHP file itself is never sent to the client/browser; only its output is

Operating Modes

- The PHP processor has two modes:
  - copy (HTML), and
  - interpret (PHP)
- PHP syntax is similar to that of JavaScript
- PHP is dynamically typed
- PHP is purely interpreted

General Syntactic Characteristics

- PHP code can be specified in an HTML document internally or externally:
- Internally:

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

- Externally:

```
include ("myScript.inc")
```

  - The included file can have both PHP and HTML code
  - If the file has PHP, the PHP must appear between these endpoints `<?php ... ?>`, even if the `include` is already within `<?php ... ?>`
- Comments — three different kinds (as in Java and C)

```
// ...    single-line comment (like in Perl)
# ...     single-line comment (like in Java or C++)
/* ... */ multi-line comment (like in C or Java)
```

- Compound statements are formed with braces `{ ... }`
- Compound statements cannot be blocks

Variables

- Variables
  - Every variable name begins with a `$`, on both declaration and usage
    - Names are *case sensitive*; use an underscore ('_') or *camelCase* to separate multiple words
  - There are no type declarations
    - The type of a variable is dynamically declared by value assignment (type is not written, but implicit)
  - If a variable is created without a value (unassigned or "unbound"), it is automatically assigned a value of `NULL`
    - The unset function sets a variable to `NULL`
    - The `IsSet()` function is used to determine whether a variable is `NULL`
- `error_reporting(15);`
  - prevents PHP from using unbound variables
- PHP has many predefined variables, including the environment variables of the host operating system
  - You can get a list of the predefined variables in a script by calling `phpinfo()`

Primitive Types

- There are eight basic/primitive types:
  - Four scalar types:
    `boolean`, `integer` (or, `int`), `float` (or, `double`), and `string`
  - Two compound types:
    `array` and `object`
  - Two special types:
    `resource` and `NULL`
- Integer & float are like those of other languages
- Boolean — values are `true` and `false` (case insensitive)
  - `0` and `""` and `"0"` are `false`; others are `true`
- PHP converts between types automatically in many cases:
  - `string` → `int` auto-conversion on `+`
  - `int` → `float` auto-conversion on `/`

Primitive Types — Strings

- Strings:
  - Characters are single bytes
  - String literals use single or double quotes
  - Strings can span multiple lines — the newline is part of the string
- Single-quoted string literals:
  - Embedded variables are NOT *interpolated* (i.e., expanded)
  - Embedded “escape” sequences (using backslash) are NOT recognized
- Double-quoted string literals:
  - Embedded variables ARE interpolated
  - If there is a variable name in a double-quoted string but you do not want it interpolated, it must be preceded by backslash (\) — an “escape” character
  - Embedded escape sequences ARE recognized
- For both single- and double-quoted literal strings, embedded delimiters must be backslashed

Arithmetic Operators, Expressions and Math Functions

- Arithmetic Operators and Expressions
  - Usual operators in a programming language

    `+ - * / % ++ -- = += -= *= /=`
    `%= == != > < >= <= && || !`
  - Two more operators: `===` `!==`
    - `==` just checks value: (`"5.0" == 5` is TRUE)
    - `===` also checks type: (`"5.0" === 5` is FALSE)
  - If the result of integer division is not an integer, a float is returned
  - Any integer operation that results in overflow produces a float
  - The modulus operator (`%`) coerces its operands to integer, if necessary
  - When a float is rounded to an integer, the rounding is always towards zero
- Math Functions

abs	ceil	cos	floor	log	log10	max
min	pow	rand	round	sin	sqrt	tan

- Math constants

M_PI	M_E	M_LN2

String Operations and Functions

- The only string operator is *period* (.), for catenation (or, concatenation) — not +, like in Java!
  - Example:

    ```
    5 + "2 things" === 7
    5 . "2 things" === "52 things"
    ```
- Zero-based indexing using bracket notation
  - Example: `$str{3}` is the *fourth* character
- `strlen()`, `strcmp()`, `strpos()`, `substr()`, as in C
- `chop()` — remove whitespace from the right end
- `trim()` — remove whitespace from both ends
- `ltrim()` — remove whitespace from the left end
- `strtolower()`, `strtoupper()`

Scalar Type Conversions

- Implicit (coercions)
  - String to numeric
  - If the string contains an `e` or an `E`, it is converted to `float`; otherwise to `integer`
  - If the string does not begin with a sign or a digit, zero is used
- Explicit conversions — casts
  - e.g., `(int)$total` or `intval($total)` or `settype($total, "integer")`
- The type of a variable can be determined with:
  - `gettype()` function; returns a variable's type as a string
  - `is_`*`type`*`()` functions, such as `is_string($var)`
- `gettype($total)` — it may return "unknown"
- `is_integer($total)` — a predicate function

PHP Processor Output

- Output from a PHP script is HTML that is sent to the browser
- HTML is sent to the browser through standard output
- There are two ways to produce output: `print` and `printf`
- `print` takes a string, but will coerce other values to strings

```
print "This is too <br /> much fun <br />";
print 72;
```

- `printf` is exactly as in C

```
printf(literal_string, param1, param2, …)
```

- PHP code is placed in the body of an HTML document

```
<html>
   <head><title> Trivial php example </title>
   </head>
   <body>
            <?php
              print "Welcome to my Web site!";
            ?>
   </body>
</html>
```

- ➔ SHOW `today.php` and display

Control Statements

- Control Expressions
  - Relational operators — same as JavaScript, (including `===` and `!==`)
  - Boolean operators — same as C (two sets, `&&` and `and`, etc.)
- Selection Statements
  - `if`, `if-else`, `elseif` a
  - `switch` — as in C
    - The switch expression type must be `integer`, `float`, or `string`
  - `while` — just like C
  - `do-while` — just like C
  - `for` — just like C
  - `foreach` — described later
  - `break` — in any `for`, `foreach`, `while`, `do-while`, or `switch`
  - `continue` — in any loop
- Alternative compound delimiters — more readability

```
if (...):
...
endif;
```

- ➔ SHOW `powers.php`

PHP in HTML

- HTML can be mixed with PHP script

```
<?php
   $a = 7;
   $b = 7;
   if ($a == $b) {
      $a = 3 * $a;
?>
<br /> At this point, $a and $b
 are equal<br />
So, we change $a to three times $a
<?php
   }
?>
```

PHP Arrays (1)

- PHP arrays are a mixture of regular arrays in other programming language and hash-maps (similar to Java hash-maps and Python Dictonaries+Lists)
  - A PHP array is a generalization of the arrays of other languages — "associative arrays"
  - A PHP array is really a mapping of keys to values, where the keys can be numbers (to get a regular array) or strings (to get a hash)
- Array creation
  - Use the `array()` construct, which takes one or more key => value pairs as parameters and returns an array of them
  - The keys are non-negative integer literals or string literals
  - The values can be anything, e.g.,

```
$list = array(0 => "apples", 1 => "oranges", 2 => "grapes")
```

    - This is a "regular" array of strings (indexed by numbers)

PHP Arrays (2)

- If a key is omitted and there have been integer keys, the default key will be the largest current key + 1
- If a key is omitted and there have been no integer keys, 0 is the default key
- If a key is repeated, the new associated value will overwrite the old one

- Arrays can have mixed kinds of elements
  - e.g.,

```
$list = array("make" => "Cessna",
"model" => "C210",
"year" => 1960,
3 => "sold");

$list = array(1, 3, 5, 7, 9);

$list = array(5, 3 => 7, 5 => 10, "month" => "May");

$colors = array('red', 'blue', 'green', 'yellow');
```

Accessing Array Elements

- Use brackets to access array elements

```
$list[4] = 7;
$list["day"] = "Tuesday";
$list[] = 17;
```

- If an element with the specified key does not exist, it is created
- If the array does not exist, the array is created
- The keys or values can be extracted from an array

```
$highs = array("Mon" => 74, "Tue" => 70,
"Wed" => 67, "Thu" => 62,
"Fri" => 65);
$days = array_keys($highs);
$temps = array_values($highs);
```

- Testing whether an element exists

```
if (array_key_exists("Wed", $highs)) …
```

- An array can be deleted with `unset()`

```
unset($list);
unset($list[4]);  # Deletes index 4 element
```

Functions on Arrays

- `is_array($list)` returns `true` if `$list` is an array
- `in_array(17, $list)` returns true if 17 is an element of `$list`
- `sizeof(an_array)` returns the number of elements
- `explode(" ", $str)` creates an array with the values of the words from `$str`, split on a space
- `implode(" ", $list)` creates a string of the elements from `$list`, separated by a space
- ➔ SHOW Figure 9.3

- Sequential access to array elements
  - `current()` and `next()`

```
$colors = array("Blue", "red", "green", "yellow");
$color = current($colors);
print("$color <br />");
while ($color = next($colors))
   print ("$color <br />");
```

Sorting Arrays

- `sort()`
  - To sort the values of an array, leaving the keys in their present order — intended for traditional arrays
    - e.g., `sort($list);`
  - The sort function does not return anything
  - Works for both strings and numbers, even mixed strings and numbers

```
$list = ('h', 100, 'c', 20, 'a');
sort($list);
// Produces ('a', 'c', 'h‘, 20, 100)
```

  - In PHP 4, the sort function can take a second parameter, which specifies a particular kind of sort

```
sort($list, SORT_NUMERIC);
```

- `asort()`
  - To sort the values of an array, but keeping the key/value relationships — intended for hashes
- Also see `rsort()`, `ksort()`, and `krsort()`

User-Defined Functions

- Syntactic form:

```
function function_name(formal_parameters) {
  ...
}
```

- General Characteristics:
  - Functions need not be defined before they are called
  - If you try to redefine a function, it is an error (no overloading!)
  - Functions can have a variable number of parameters
  - Default parameter values are supported
  - Function definitions can be nested
  - Function names are NOT case sensitive
  - The return function is used to return a value;
  - If there is no `return`, there is no returned value

User-Defined Functions: Parameters

- If the caller sends too many actual parameters, the subprogram ignores the extra ones
- If the caller does not send enough parameters, the unmatched formal parameters are unbound
- The default parameter passing method is *pass-by-value* (one-way communication)
- To specify *pass-by-reference*, prepend an ampersand to the formal parameter

```
function set_max(&$max, $first, $second) {
if ($first >= $second)
$max = $first;
else
$max = $second;
}
```

- If the function does not specify its parameter to be pass-by-reference, you can prepend an ampersand to the actual parameter and still get pass-by-reference semantics

User-Defined Functions

Return Values, Scope & Lifetime of Variables

- Return Values
  - Any type may be returned, including objects and arrays, using the return
  - If a function returns a reference, the name of the function must have a prepended ampersand

```
function &newArray($x) { … }
```

- The Scope of Variables
  - An undeclared variable in a function has the scope of the function
  - To access a nonlocal variable, it must be declared to be global, as in

```
global $sum;
```

- The Lifetime of Variables
  - Normally, the lifetime of a variable in a function is from its first appearance to the end of the function's execution

```
static $sum = 0;  # $sum is static
```

Pattern Matching

- PHP has two kinds:
  POSIX and Perl-compatible
  - `preg_match(regex, str)`
    - Returns a Boolean value
  - `preg_split(regex, str)`
    - Returns an array of the substrings
- ➔ SHOW word_table.php

Form Handling

- Forms could be handled by the same document that creates the form, but that may be confusing
- PHP particulars:
  - It does not matter whether `GET` or `POST` method is used to transmit the form data
  - PHP builds an array of the form values (`$_GET` for the GET method and `$_POST` for the POST method — subscripts are the widget names)
- ➔ SHOW popcorn3.html & popcorn3.php

PHP/MySQL Interface

`mysql_connect(host, username, password, new_link)`

- establishes a connection with MySQL server on `host`
- includes authentication information
- reuses existing link by default (unless `new_link` is `true`)
- all arguments have defaults (coming from `php.ini`)

`mysql_select_db(database, link)`

- selects `database` (makes it "active")
- similar to MySQL `USE` command
- `link` is optional, needed for multiple connections

`mysql_query(query, link)`

- makes an SQL query, `query`, on open connection
- returns a "handle" for result table
- consider `mysq_unbuffered_query` for large results
- use `mysql_num_rows()` for count of result rows (`SELECT`)
- use `mysql_affected_rows()` for rows affected by operation (`DELETE`, `INSERT`, `REPLACE`, `UPDATE`)

`mysql_fetch_array(result_handle, result_type)`

- returns next row of result table (`result_handle`) as an array
- possible `result_type` values: `MYSQL_ASSOC`, `MYSQL_NUM`, and `MYSQL_BOTH`
  - Using `MYSQL_ASSOC`, you only get associative indices
  - Using `MYSQL_NUM`, you only get number indices

Typical MySQL Session

```
<?php
// Connecting, selecting database
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
   or die('Could not connect: ' . mysql_error());
echo 'Connected successfully';
mysql_select_db('my_database') or die('Could not select database');
// Performing SQL query
$query = 'SELECT * FROM my_table';
$result = mysql_query($query) or die('Query failed: ' . mysql_error());
// Printing results in HTML
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
   echo "\t<tr>\n";
   foreach ($line as $col_value) {
      echo "\t\t<td>$col_value</td>\n";
   }
   echo "\t</tr>\n";
}
echo "</table>\n";
```

```
// Free resultset
mysql_free_result($result);
// Closing connection
mysql_close($link);
?>
```

Also see: MySQLi — MySQL Improved Extension
http://php.net/manual/en/set.mysqlinfo.php

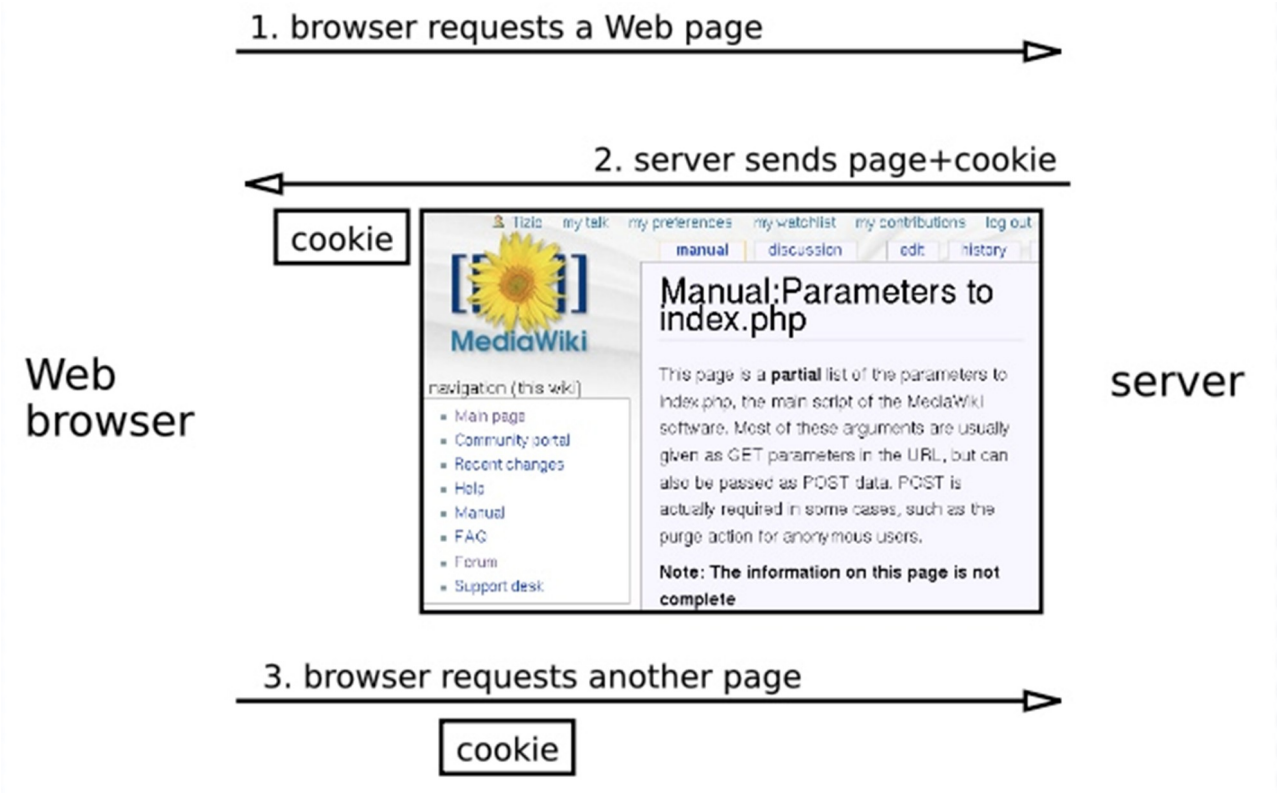
Cookies (1)

- Recall that the HTTP protocol is stateless; however, there are several reasons why it is useful for a server to relate a request to earlier requests
  - Targeted advertising
  - Shopping baskets
- A cookie is a name/value pair that is passed between a browser and a server in the HTTP header
- In PHP, cookies are created with setcookie

```
setcookie(cookie_name, cookie_value, lifetime)
```

  - e.g., `setcookie("voted", "true", time() + 86400);`
- Cookies are implicitly deleted when their lifetimes are over
- Cookies must be created before any other HTML is created by the script
- Cookies are obtained in a script the same way form values are obtained, using the `$_COOKIES` array

Cookies (2)



- http://en.wikipedia.org/wiki/HTTP_cookie
- Cookies are marked as to the web addresses they come from — the browser only sends back cookies that were originally set by the same web server

Session Tracking

- A session is the time span during which a browser interacts with a particular server
- For session tracking, PHP creates and maintains a session tracking id
- Create the id with a call to `session_start()` with no parameters
- Subsequent calls to `session_start()` retrieves any session variables that were previously registered in the session, using the array `$_SESSION`
- To create a session variable, use `session_register()`
  - The only parameter is a string literal of the name of the session variable (without the dollar sign)
- Example: To count number of pages visited, put the following code in all documents

```
session_start();
if (!IsSet($page_number))
$page_number = 1;
print("You have now visited $page_number");
print(" pages <br />");
$page_number++;
$session_register("page_number");
```