

CSC309 Winter 2016

Lecture 4

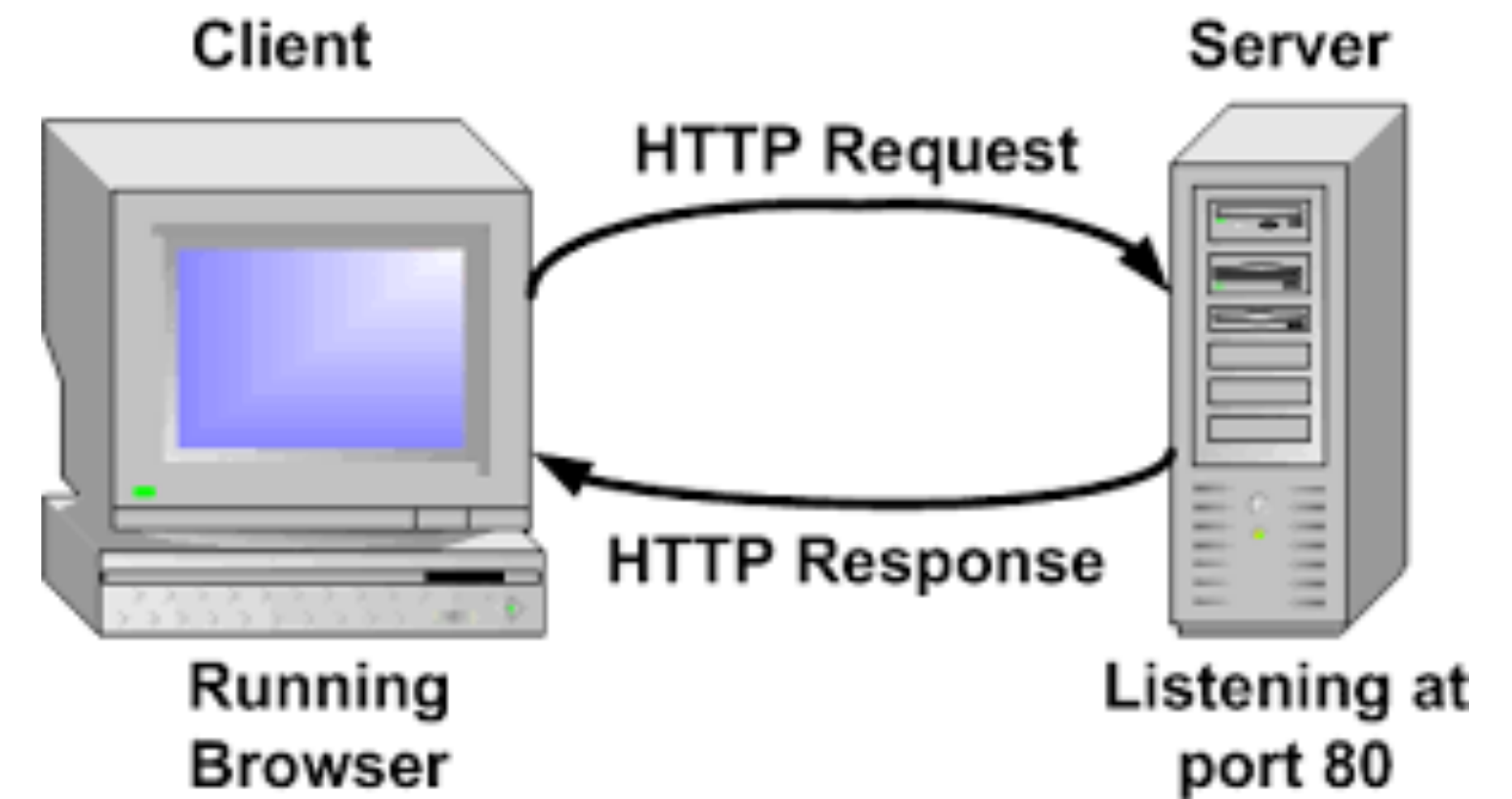
Larry Zhang

Today's topic

- Server-side programming
- PHP
- We will go through some basic concepts then we will do demos.

What is server-side programming

- Normally, the client requests an **HTML document** (e.g., hello.html) on the server; the server simply gets the document and sends it back to the client as response.
- But if the client requests a **script file** (e.g., hello.php), the server would run that script, do some computations, generate an HTTP response and send it back to the server.
- The script is run at the server, not at the client.



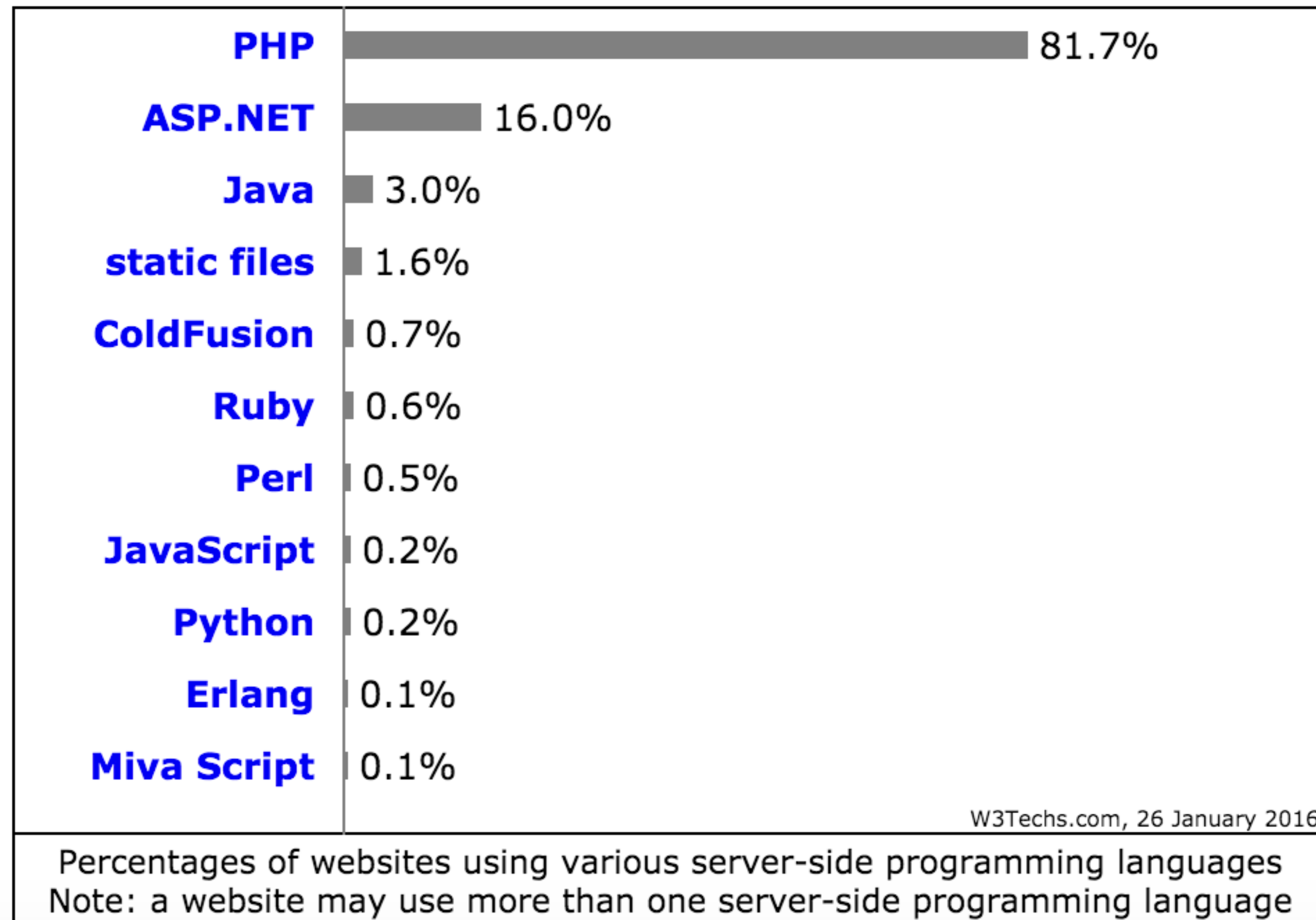
Why server-side programming?

- Create dynamic web page (the content of the web page is different for different computation result from the script)
- Process user input (user data submitted using HTML forms can be used by the script run by the server)
- Interact with database on the server.
- Provides security: The content of the script cannot be seen in the browser.

Server-side script can be any language

- PHP, Python, Ruby, Javascript, ASP.NET, Java, C/C++, Haskell, Go, Perl, ...
- We will learn **PHP** first, since it is the easiest to get started with
- It's free and open source
- Used by Facebook, Yahoo!, Wikipedia, Wordpress, Tumblr, ...

Market share

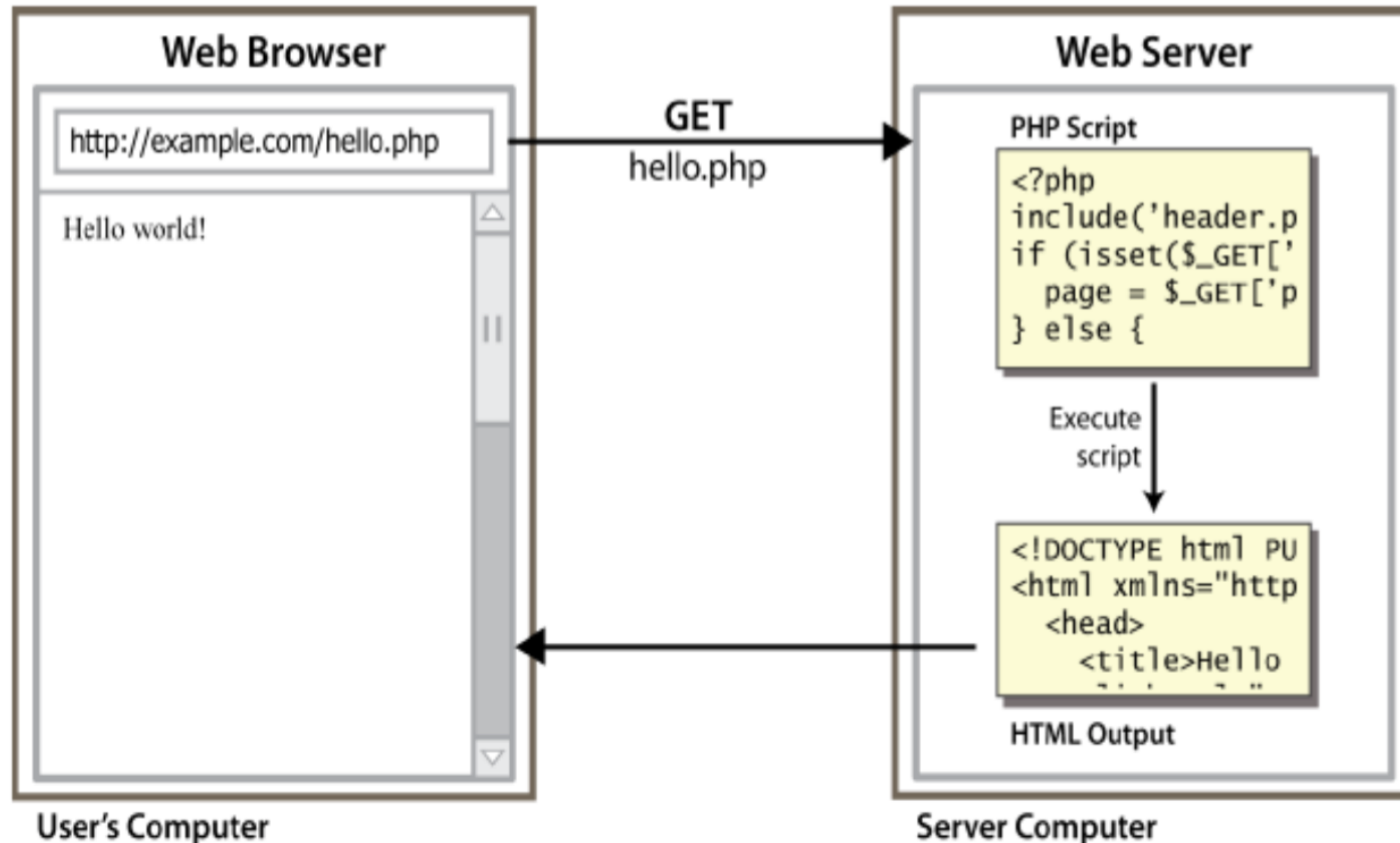


PHP



- stands for “PHP Hypertext Preprocessor”
- We are using PHP 5.3 on cs.utm, released on June, 2009
- PHP6 was started but was never released
- PHP7.0 released on December 3, 2015

Lifecycle a PHP request



Hello World!

```
<?php  
print "Hello, world!";  
?>
```

PHP

Hello world!

output

PHP Basic Syntax

PHP Syntax Template (.php file)

```
HTML content  
<?php  
PHP code  
?>  
HTML content  
<?php  
PHP code  
?>  
HTML content ...
```

PHP

- ❑ Place PHP code in .php file between **<?php** and **?>**
- ❑ Switch between HTML and PHP "modes"

Variables

```
$name = expression;
```

PHP

```
$user_name = "mundruid78";  
$age = 16;  
$drinking_age = $age + 5;  
$this_class_rocks = TRUE;
```

PHP

- ❑ Variable names begin with \$
 - ▣ both *declaration* and *use*
 - ▣ names are *case sensitive*
- ❑ loosely typed language (like JavaScript or Python)
 - ▣ Type implicitly declared by assignment

Variable: types

- types: *int, float, boolean, string, array, object, NULL*
 - ▣ test type of variable with `is_type` functions, e.g.
`is_string`
 - ▣ `gettype` function returns a variable's type as a string
- PHP *converts between types automatically*:
 - ▣ `string` → `int` auto-conversion on `+`
 - ▣ `int` → `float` auto-conversion on `/`
- type-cast with **(type)**:
 - ▣ `$age = (int) "21";`

bool

```
$feels_like_summer = FALSE;  
$php_is_great = TRUE;  
$student_count = 7;  
$nonzero = (bool) $student_count; # TRUE
```

PHP

- the following values are considered to be FALSE (all others are TRUE):
 - ▣ 0 and 0.0 (but NOT 0.00 or 0.000)
 - ▣ "", "0", and NULL (includes unset variables)
 - ▣ arrays with 0 elements
- FALSE prints as an empty string (no output);
- TRUE prints as a 1

Arithmetic operators

□ + - * / % . ++ --

□ = += -= *= /= %= .=

□ many operators auto-convert types: 5 + "7" is 12

Math operations

```
$a = 3;  
$b = 4;  
$c = sqrt(pow($a, 2) + pow($b, 2));
```

PHP

math functions

<u>abs</u>	<u>ceil</u>	<u>cos</u>	<u>floor</u>	<u>log</u>	<u>log10</u>	<u>max</u>
<u>min</u>	<u>pow</u>	<u>rand</u>	<u>round</u>	<u>sin</u>	<u>sqrt</u>	<u>tan</u>

math constants

M_PI	M_E	M_LN2
------	-----	-------

int and float

```
$a = 7 / 2;           # float: 3.5  
$b = (int) $a;        # int: 3  
$c = round($a);       # float: 4.0  
$d = "123";           # string: "123"  
$e = (int) $d;         # int: 123
```

PHP

- int for integers and float for reals
- division between two int values can produce a float

Strings

String

```
$favorite_food = "Ethiopian";  
print $favorite_food[2];  
$favorite_food = $favorite_food . " cuisine";  
print $favorite_food;
```

PHP

- ❑ zero-based indexing using bracket notation
- ❑ there is no char type; each letter is itself a String
- ❑ string concatenation operator is . (period), not +
 - `5 + "2 turtle doves" === 7`
 - `5 . "2 turtle doves" === "52 turtle doves"`

String functions

```
# index 0123456789012345
$name = "Stefanie Hatcher";
$length = strlen($name);
$cmp = strcmp($name, "Brian Le");
$index = strpos($name, "e");
$first = substr($name, 9, 5);
$name = strtoupper($name);
```

PHP

Interpreted strings

```
$age = 16;  
print "You are " . $age . " years old.\n";  
print "You are $age years old.\n"; # You are 16 years old.  
PHP
```

- strings inside " " are **interpreted**
 - ▣ variables that appear inside them will have their values inserted into the string
- strings inside ' ' are **not interpreted**:

```
print 'You are $age years old.\n'; # You are $age years  
old. \n  
PHP
```

Arrays

Arrays

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

PHP

```
$a = array();             # empty array (length 0)  
$a[0] = 23;               # stores 23 at index 0 (length 1)  
$a2 = array("some", "strings", "in", "an", "array");  
$a2[] = "Ooh!";          # add string to end (at index 5)
```

PHP

- Append: use bracket notation without specifying an index
- Element type is not specified; can mix types

Some array functions

function name(s)	description
<u>count</u>	number of elements in the array
<u>print_r</u>	print array's contents
<u>array_pop</u> , <u>array_push</u> , <u>array_shift</u> , <u>array_unshift</u>	using array as a stack/queue
<u>in_array</u> , <u>array_search</u> , <u>array_reverse</u> , <u>sort</u> , <u>rsort</u> , <u>shuffle</u>	searching and reordering
<u>array_fill</u> , <u>array_merge</u> , <u>array_intersect</u> , <u>array_diff</u> , <u>array_slice</u> , <u>range</u>	creating, filling, filtering
<u>array_sum</u> , <u>array_product</u> , <u>array_unique</u> , <u>array_filter</u> , <u>array_reduce</u>	processing elements

Functions

Functions

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

PHP


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

PHP

- ❑ PHP supports passing **arguments by value** (the default), passing by reference, default argument values, and variable-length argument lists
- ❑ Parameter types and return types are not written
- ❑ A function with no return statements implicitly returns NULL

Default parameter values

```
function print_separated($str, $separator = ", ") {  
    if (strlen($str) > 0) {  
        print $str[0];  
        for ($i = 1; $i < strlen($str); $i++) {  
            print $separator . $str[$i];  
        }  
    }  
}
```



PHP

```
print_separated("hello"); # h, e, l, l, o  
print_separated("hello", "-"); # h-e-l-l-o
```

PHP

- if no value is passed, the default will be used

Comments

```
# single-line comment  
// single-line comment  
/*  
multi-line comment  
*/
```

PHP

- like Java, but # is also allowed
 - ▣ a lot of PHP code uses # comments instead of //

for loop

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

PHP

while loop

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

PHP

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

PHP

if-elseif-else

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

PHP

foreach loop

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

PHP

```
$fellowship = array("Frodo", "Sam", "Gandalf",  
    "Strider", "Gimli", "Legolas", "Boromir");  
print "The fellowship of the ring members are: \n";  
for ($i = 0; $i < count($fellowship); $i++) {  
    print "{$fellowship[$i]}\n";  
}  
print "The fellowship of the ring members are: \n";  
  
foreach ($fellowship as $fellow) {  
    print "$fellow\n";  
}
```

PHP

Expressions in PHP

PHP expression block

```
<?= expression ?>
```

PHP

```
<h2> The answer is <?= 6 * 7 ?> </h2>
```

PHP

The answer is 42

output

- PHP expression block: a small piece of PHP that evaluates and embeds an expression's value into HTML

- ▣ `<?= expression ?>` is equivalent to:

```
<?php print expression; ?>
```

PHP

A more complex example

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>CSE 190 M: Embedded PHP</title></head>
<body>
<?php
for ($i = 99; $i >= 1; $i--) {
?>
<p> <?= $i ?> bottles of beer on the wall, <br />
<?= $i ?> bottles of beer. <br />
Take one down, pass it around, <br />
<?= $i - 1 ?> bottles of beer on the wall. </p>
<?php
}
?>
</body>
</html>
```

PHP

Another complex example

```
....  
<body>  
<?php  
for ($i = 1; $i <= 3; $i++) {  
    ?>  
    <h<?= $i ?>>This is a level <?= $i ?>  
heading.</h<?= $i ?>>  
    <?php  
}  
?>  
</body>
```

PHP

This is a level 1 heading.

This is a level 2 heading.

This is a level 3 heading.

output

Bad style

```
<?php
print "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML
1.1//EN\"\\n";
print "
\"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">\\n";
print "<html xmlns=\"http://www.w3.org/1999/xhtml\">\\n";
print " <head>\\n";
print " <title>Geneva's web page</title>\\n";
...
for ($i = 1; $i <= 10; $i++) {
    print "<p> I can count to $i! </p>\\n";
}
?>
```

HTML

- ❑ Try to minimize print/echo statements in embedded PHP code

PHP include file

PHP include file

```
<a href="/default.php">Home</a>
<a href="/tutorials.php">Tutorials</a>
<a href="/references.php">References</a>
<a href="/examples.php">Examples</a>
<a href="/contact.php">Contact Us</a>
```

PHP

```
<html>
<body>

<div class="leftmenu">
<?php include("menu.php"); ?>
</div>

<h1>Welcome to my home page.</h1>
<p>I have a great menu here.</p>

</body>
</html>
```

PHP

PHP include files

- Insert the content of one PHP file into another PHP file before the server executes it
 - ▣ `include()`
 - if not found, generates a warning, but the script will continue execution
 - ▣ `require()`
 - if not found, generates a fatal error, and the script will stop

demo

HTML Forms

HTML Forms

- In short, HTML forms enables users to submit data to the website.
- This was a big step in the history of Web.
- Server-side programs accepts parameters sent from the users to guide their execution.

Query string

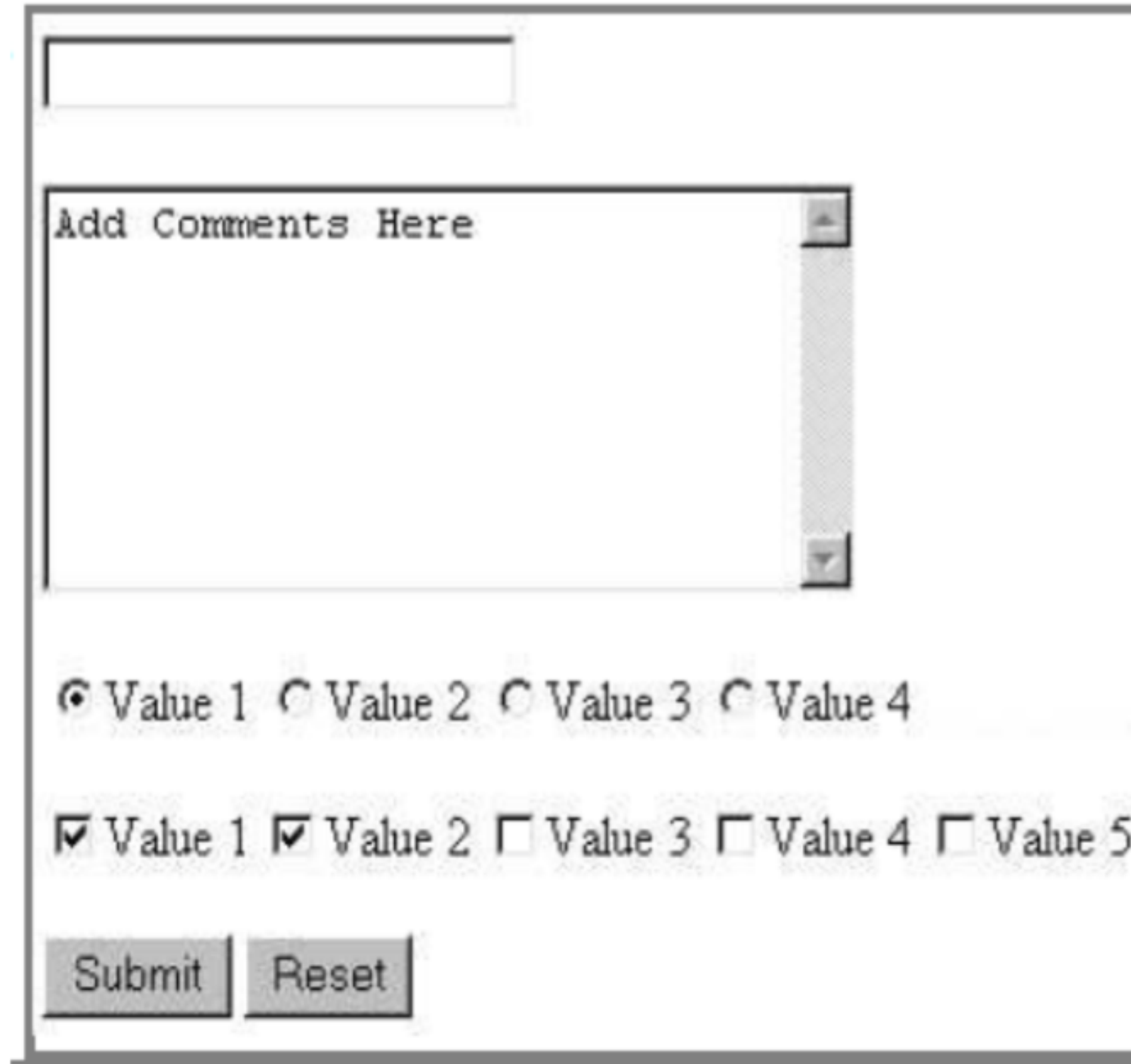
```
URL?name=value&name=value...
```

```
http://example.com/student_login.php?username=bob&sid=1234
```

- **query string:** a set of parameters passed from a browser to a web server
 - ▣ passed by placing *name/value pairs* at the end of a URL
- PHP code on the server can examine and utilize the value of parameters

HTML Forms

- **form:** a group of UI controls that accepts information from the user and sends the information to a web server
- the information is sent to the server as a query string



The image shows a screenshot of an HTML form within a window. At the top is a single-line text input field. Below it is a larger text area with the placeholder text "Add Comments Here" and a vertical scrollbar on the right. Under the text area are two rows of radio buttons: the first row contains "Value 1", "Value 2", "Value 3", and "Value 4", with "Value 1" selected; the second row contains "Value 1", "Value 2", "Value 3", "Value 4", and "Value 5", with "Value 1" and "Value 2" selected. At the bottom of the form are two buttons labeled "Submit" and "Reset".

HTML Form Example

```
<form action="http://www.google.com/search">  
  <div>  
    Let's search Google:  
    <input name="q" />  
    <input type="submit" />  
  </div>  
</form>
```

HTML

Let's search Google:

- required **action** attribute gives the URL of the page that will process this form's data
- when form has been filled out and **submitted**, its data will be sent to the action's URL

demo continued...

PHP Sessions

Why sessions

- Normal HTTP requests are **stateless**, i.e., request, response and done. Nothing is remembered between different requests.
 - If you refresh a page, all states of the page would be gone
- But sometimes we want to remember things across different page visits.
 - The user can start a session, store information of the session in a file on the server, and use it across different requests.

How session works

- client's browser makes an initial request to the server
- server notes client's IP address/browser, stores some local session data, and sends a session ID back to client (via cookies)
- client sends that same session ID (in a cookie) back to server on future requests
- server uses session ID to retrieve the data for the client's session later, like a ticket given at a coat-check room

Use sessions in PHP

- `session_start()`
- `session_destroy()`
- `value = $_SESSION[key]`
- `session_save_path(dirname)`
 - `dirname` must exist in your current folder, with permission 700
 - current folder since we have suPHP on CS.UTM server

demo continued...

Reference:

- <http://www.w3schools.com/php/default.asp>
- <http://php.net/manual/en/>

Next week

- More PHP and server-side programming