Web Programming Step by Step

Lecture 6
Introduction to PHP
Reading: 5.1 - 5.3

Except where otherwise noted, the contents of this presentation are Copyright 2009 Marty Stepp and Jessica Miller.

4.4: Sizing and Positioning

The `display` property (4.4.4)

```css
h2 { display: inline; background-color: yellow; }
```
CSS

This is a heading **This is another heading**

output

property	description
`display`	sets the type of CSS box model an element is displayed with

- values: `none`, `inline`, `block`, `run-in`, `compact`, ...
- use sparingly, because it can radically alter the page layout

Displaying block elements as inline

```html
<ul id="topmenu">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```
HTML

```css
#topmenu li {
  display: inline;
  border: 2px solid gray;
  margin-right: 1em;
}
```
CSS

Item 1 Item 2 Item 3

output

- lists and other block elements can be displayed inline
  - flow left-to-right on same line
  - width is determined by content (block elements are 100% of page width)

The `visibility` property

```
p.secret {
  visibility: hidden;
}
```
CSS

output

property	description
`visibility`	sets whether an element should be shown onscreen; can be `visible` (default) or `hidden`

- `hidden` elements will still take up space onscreen, but will not be shown
    - to make it not take up any space, set `display` to `none` instead
- can be used to show/hide dynamic HTML content on the page in response to events

5.1: Server-Side Basics

URLs and web servers

```
http://server/path/file
```

- usually when you type a URL in your browser:
  - your computer looks up the server's IP address using DNS
  - your browser connects to that IP address and requests the given file
  - the web server software (e.g. Apache) grabs that file from the server's local file system, and sends back its contents to you

- some URLs actually specify *programs* that the web server should run, and then send their output back to you as the result:

  ```
  https://webster.cs.washington.edu/quote2.php
  ```

  - the above URL tells the server `webster.cs.washington.edu` to run the program `quote2.php` and send back its output

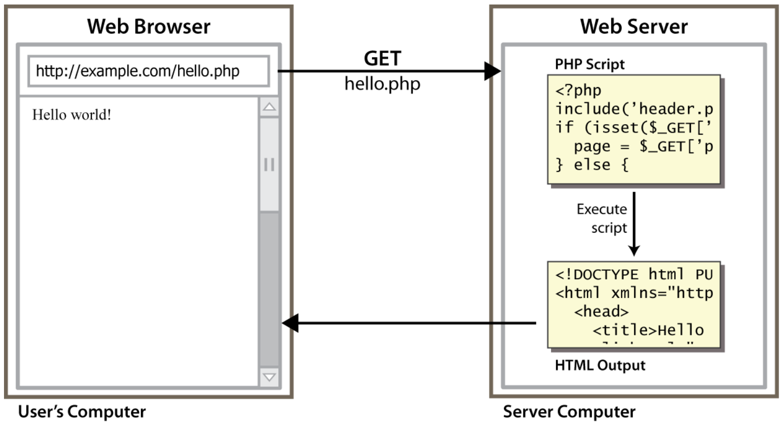
Server-Side web programming

- server-side pages are programs written using one of many web programming languages/frameworks
  - examples: PHP, Java/JSP, Ruby on Rails, ASP.NET, Python, Perl
- the web server contains software that allows it to run those programs and send back their output as responses to web requests
- each language/framework has its pros and cons
  - we use PHP for server-side programming in this textbook

What is PHP? (5.1.2)

- **PHP** stands for "PHP Hypertext Preprocessor"
- a server-side scripting language
- used to make web pages dynamic:
    - provide different content depending on context
    - interface with other services: database, e-mail, etc
    - authenticate users
    - process form information
- PHP code can be embedded in XHTML code

Lifecycle of a PHP web request (5.1.1)



```
<?php
include('header.p
if (isset($_GET['
  page = $_GET['p
} else {
```

```
<!DOCTYPE html PU
<html xmlns="http
  <head>
    <title>Hello
```

- 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 the response sent back

Why PHP?

There are many other options for server-side languages: Ruby on Rails, JSP, ASP.NET, etc. Why choose PHP?

- free and open source: anyone can run a PHP-enabled server free of charge
- compatible: supported by most popular web servers
- simple: lots of built-in functionality; familiar syntax
- available: installed on UW's servers (Dante, Webster) and most commercial web hosts

Hello, World!

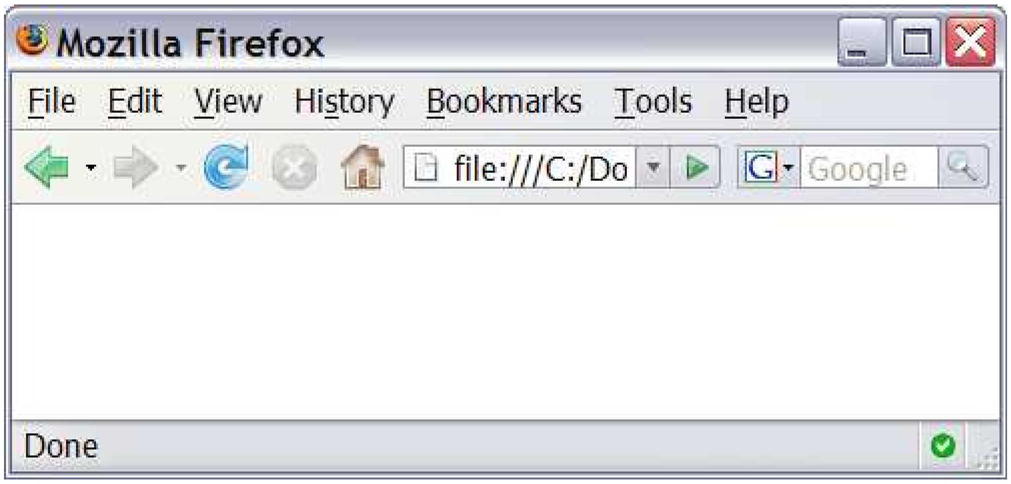
The following contents could go into a file `hello.php`:

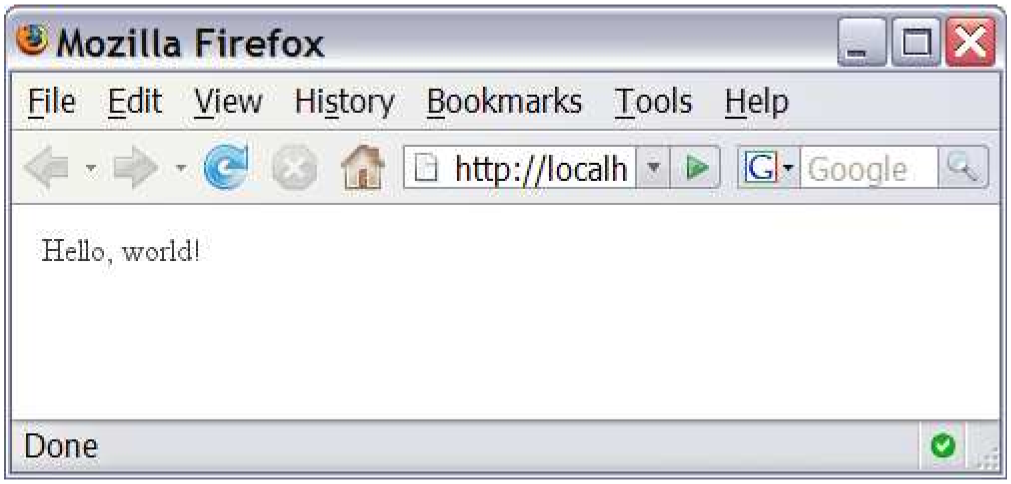
```php
<?php
print "Hello, world!";
?>
```
PHP

```
Hello, world!
```
output

- a block or file of PHP code begins with `<?php` and ends with `?>`
- PHP statements, function declarations, etc. appear between these endpoints

Viewing PHP output





```
Hello, world!
```

- you can't view your `.php` page on your local hard drive; you'll either see nothing or see the PHP source code
- if you upload the file to a PHP-enabled web server, requesting the `.php` file will run the program and send you back its output

5.2: PHP Basic Syntax

Console output: `print` (5.2.2)

```
print "text";
```
PHP

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

print "You can have
line breaks in a string.";

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

```
Hello, World! Escape "chars" are the SAME as in Java! You can have line breaks in a string. A string can use
"single-quotes". It's cool!
```
output

- some PHP programmers use the equivalent `echo` instead of `print`

Variables (5.2.5)

```
$name = expression;
```
PHP

```
$user_name = "PinkHeartLuvr78";
$age = 16;
$drinking_age = $age + 5;
$this_class_rocks = TRUE;
```
PHP

- names are case sensitive; separate multiple words with _
- names always begin with `$`, on both declaration and usage
- always implicitly declared by assignment (type is not written)
- a loosely typed language (like JavaScript or Python)

Types (5.2.3)

- basic types: `int`, `float`, `boolean`, `string`, `array`, `object`, `NULL`
  - test what type a variable is with `is_`*type* functions, e.g. `is_string`
  - `gettype` function returns a variable's type as a string (not often needed)
- PHP converts between types automatically in many cases:
  - `string` → `int` auto-conversion on `+`
  - `int` → `float` auto-conversion on `/`
- type-cast with `(`*type*`)`:
  - `$age = (int) "21";`

Arithmetic operators (5.2.4)

- `+ - * / % . ++ --`
  `= += -= *= /= %= .=`
- many operators auto-convert types: `5 + "7"` is `12`

Comments (5.2.7)

```
#  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 //
  - we recommend # and will use it in our examples

`String` type (5.2.6)

```
$favorite_food = "Ethiopian";
print $favorite_food[2];            # h
```
PHP

- zero-based indexing using bracket notation
- string concatenation operator is `.` (period), not `+`
  - `5 + "2 turtle doves" === 7`
  - `5 . "2 turtle doves" === "52 turtle doves"`
- can be specified with `""` or `''`

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

- if necessary to avoid ambiguity, can enclose variable in `{}`:

```
print "Today is your $ageth birthday.\n";      # $ageth not found
print "Today is your {$age}th birthday.\n";
```
PHP

`for` loop (same as Java) (5.2.9)

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

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

bool (Boolean) type (5.2.8)

```
$feels_like_summer = FALSE;
$php_is_rad = TRUE;

$student_count = 217;
$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
- can cast to boolean using `(bool)`
- `FALSE` prints as an empty string (no output); `TRUE` prints as a `1`
- `TRUE` and `FALSE` keywords are case insensitive

`if/else` statement

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

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

`while` loop (same as Java)

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

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

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

Math operations

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

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

math functions

M_PI	M_E	M_LN2

math constants

- the syntax for method calls, parameters, returns is the same as Java