PHP

CSE 190 M (Web Programming), Spring 2008
University of Washington

What is PHP?

- **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 (seen later)

Why server-side programming?

JavaScript already allows us to create dynamic, programmable web pages. Why use a server-side language instead of JavaScript?

- **security**:
  - server-side code has access to server's important and/or private data
  - client can't see your source code
- **compatibility**: avoids browser JavaScript compatibility issues
- **efficiency**: faster for users
  - don't have to run a script to view each page
  - don't have to send entire data set from server to user's browser
- **power**: fewer restrictions (can write to files, open connections to other servers, connect to databases, ...)

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

PHP vs. JavaScript

- similarities:
  - **interpreted**, not compiled
  - relaxed about syntax and rules (loose types; variables don't need to be declared)
  - case-sensitive
  - has built-in regular expressions
- differences:
  - more procedural (`verb(noun)` rather than `noun.verb()`)
  - geared more toward text and file processing
  - can (and should) be mixed with XHTML, rather than in separate files

Hello, World!

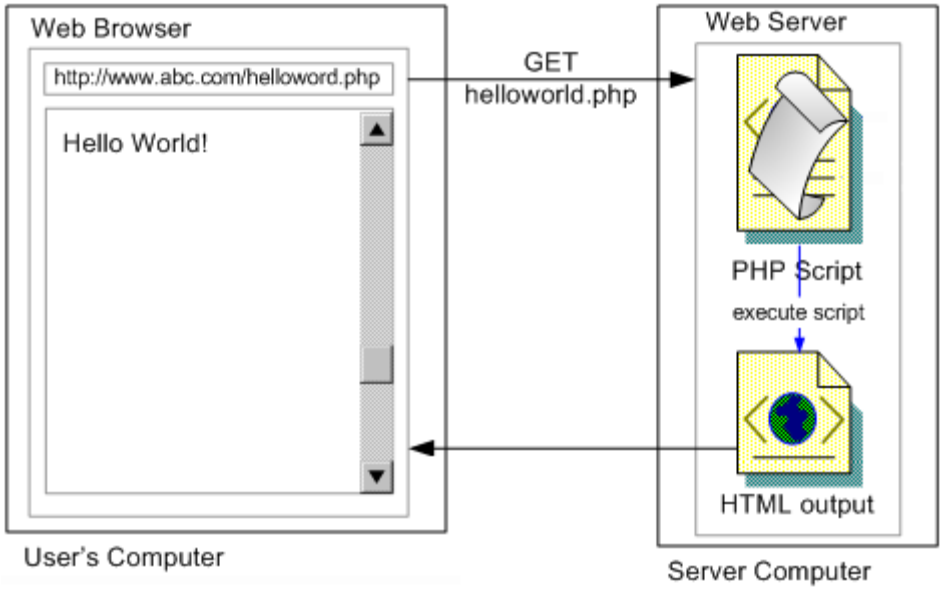
The following contents could go into a file `hello.php`:

```php
<?php
header("Content-type: text/plain");

print "Hello, world!\n";
print "\n";
print "This is my first PHP program.\n";
?>
```

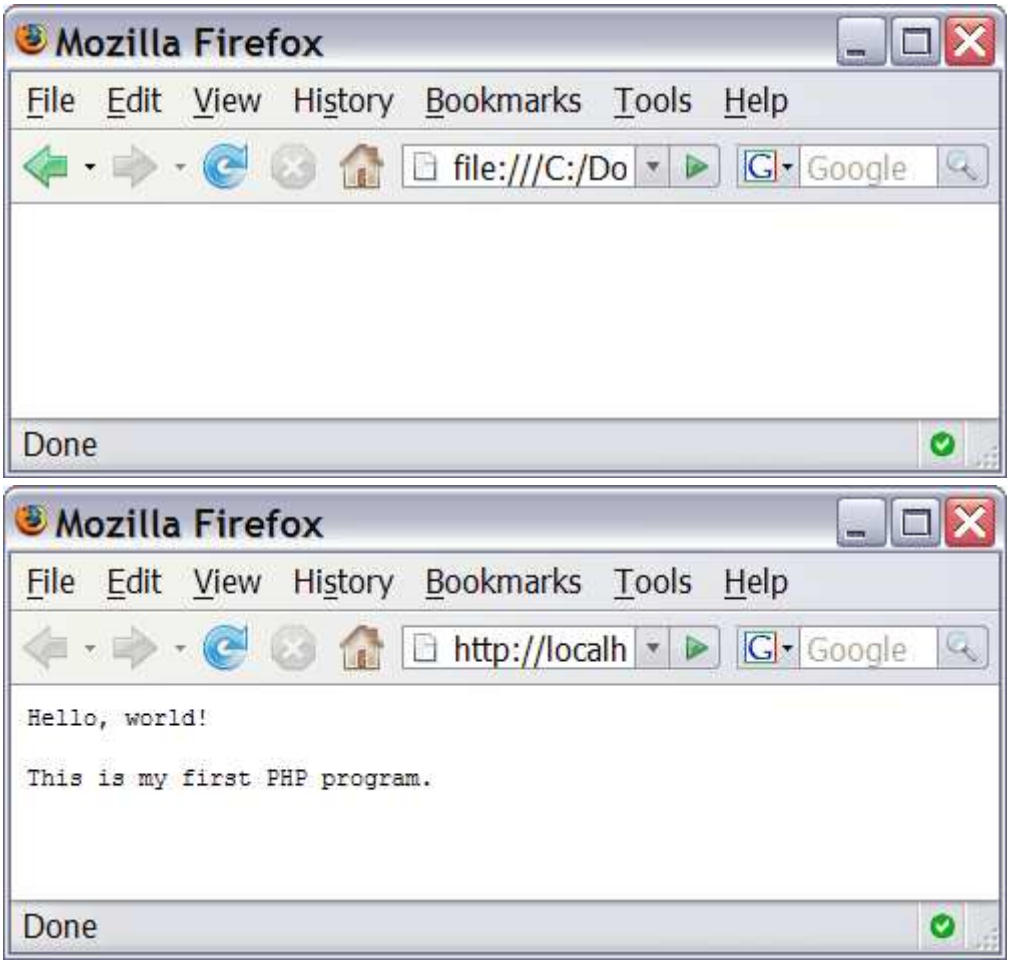
- a block or file of PHP code begins with `<?php` and ends with `?>`
- PHP statements, function declarations, etc. appear between these endpoints

Web servers and 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 the response sent back

Viewing PHP output



- 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

Basic PHP syntax

Creating simple PHP script programs

Console output: `print`

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

```
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!';
```
PHP

- some PHP programmers use the equivalent `echo` instead of `print`
- you can optionally surround the string with parentheses:

```
print("Hello, world!");
```
PHP

Headers

```
header("Content-type: text/plain");
```
PHP

- by default, a PHP script's output is assumed to be HTML
- for now, we want to output plain text instead
- use the `header` function to specify non-HTML output
  - must appear before any other output generated by the script
- (we'll see more about headers later)

Variables

```
$name = expression;
```

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

- 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)
- like JavaScript, a loosely typed language

Types

- 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";`

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

`int` and `float` types

```php
$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
```

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

Math operations

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

- functions:
  - `abs`, `ceil`, `cos`, `floor`, `log`, `log10`, `max`, `min`, `pow`, `rand`, `round`, `sin`, `sqrt`, `tan`, ...
- constants:
  - `M_PI`, `M_E`, `M_LN2`
- as you can see, the syntax for parameters and returns is the same as Java/JS

Comments

```php
# single-line comment

// single-line comment

/*
multi-line comment
*/
```

- like Java and JavaScript, but # is also allowed
  - a lot of PHP code uses # comments instead of `//`
  - we recommend # and will use it in our examples

`String` type

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

- 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 `''`

`String` functions

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

Name	Java/JS Equivalent
`explode`, `implode`	`split`, `join`
`strlen`	`length`
`strcmp`	`compareTo`
`strpos`	`indexOf`
`substr`	`substring`
`strtolower`, `strtoupper`	`toLowerCase`, `toUpperCase`
`trim`	`substring`

Interpreted strings

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

- strings inside " " are **interpreted**
  - variables that appear inside them will have their values inserted into the string
  - simpler syntax than concatenation (PHP was made for text processing)
- strings inside ' ' are *not* interpreted:

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

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

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

`for` loop (same as Java/JS)

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

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

bool (Boolean) type

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

$student_count = 96;
$nonzero = (bool) $student_count;     # TRUE
```

- 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

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

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

`while` loop (same as Java/JS)

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

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

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

NULL

- a variable is `NULL` if
  - it has not been set to any value (undefined variables)
  - it has been assigned the constant `NULL`
  - it has been `unset`
- can test if a variable is `NULL` using the `isset` function
- `NULL` prints as an empty string (no output)

Functions

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

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

- parameter types and return types are not written

Calling functions

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

```
$x = -2;
$a = 3;
$root = quadratic(1, $x, $a - 2);
```
PHP

- if the wrong number of parameters are passed, it's an error

Default parameter values

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

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

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
- default-valued parameters must come last

Variable scope: global and local vars

```
$school = "UW";                   # global
...

function downgrade() {
  global $school;
  $suffix = "Tacoma";             # local

  $school = "$school $suffix";
  print "$school\n";
}
```
PHP

- variables declared in a function are **local** to that function
- variables not declared in a function are **global**
- if a function wants to use a global variable, it must say so with a `global` statement at its start