PHP: Getting Started

Introduction

This document describes:

1. the basic syntax for PHP code,
2. how to execute PHP scripts,
3. methods for sending output to the browser,
4. the handling of whitespace, and
5. how to comment your code.

Basic PHP syntax

PHP code typically resides in a plaintext file with a `.php` extension. The code itself is defined within PHP delimiters: `<?php` and `?>`. The PHP interpreter considers anything outside of these delimiters as plaintext to be sent as-is to the client.

> *NOTE*: In some code examples on the Internet, you may see the short tag form of the PHP delimiters: `<?` and `?>`. There are two reasons to use the standard delimiters instead:
>
> 1. The short tag feature is *optional*, and can be disabled in the PHP configuration. Thus, to make your scripts more portable, use `<?php` to begin your PHP code.
> 2. If an XML (or XHTML) parser is used in the context of your PHP code, the parser will likely be confused by the presence of `<?`. Recall that valid XML documents begin with `<?xml version="1.0" encoding="utf-8"?>`.

As with C every PHP statement ends with a semicolon, and most PHP constructs such as function and variable names are case-sensitive. If most of the page content is HTML, the embedded PHP code block is called a code island. Consider the example below.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
    <title>Some title here</title>
  </head>
  <body>
    <?php
      phpinfo();   // phpinfo() is a built-in PHP function
    ?>
  </body>
</html>
```

Script execution

There are two methods for executing PHP scripts: via the Web server, and the command-line interface (CLI). The first method will be used almost exclusively in this course, so you may ignore the CLI for now.

1. Upload your `.php` file to your Web account (i.e., within the `web` directory).
2. Make sure the permissions are set correctly; for example, `chmod 644 somefile.php`.
3. Navigate to the file with a Web browser. The URL will be http://web.eecs.utk.edu/~*username*/*somefile*.php.

The process happening behind the scenes is as follows:

1. You (the client) are requesting the URL http://web.eecs.utk.edu/~*username*/*somefile*.php.
2. The Web server is configured to associate any filename ending in `.php` with the PHP interpreter. The PHP interpreter opens the file and begins processing it.
3. The output of the PHP interpreter is sent to the client.

Script debugging will be discussed later in the course; however, there are two common situations that may occur:

1. If the permissions are not set correctly for a `.php` file, the user will see *no output* in the Web browser. With HTML, however, the user will see a "403 Forbidden" error page.

   NOTE: The "403" page is a pre-defined error page that typically comes standard with the Web server. This page is sent to the client if the Web server could not find the specified file.

2. If there is a parsing error in the PHP script (e.g., a missing semicolon, unbalanced parentheses), the user will see *no output* in the Web browser. One way to check your PHP scripts is to SSH into an EECS/Claxton machine and run your script via the PHP CLI (e.g., `php foo.php`).

Output

To display simple, unformatted strings, use the `print(`*`string`*`)` function:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
    <title>Some title here</title>
  </head>
  <body>
    <p>The timeless phrase is:
    <?php
      print("Hello, world!");
    ?>
    </p>
  </body>
</html>
```

There are other printing functions in PHP that you may encounter, such as `echo()` and `printf()`. The `echo()` function is slightly different in that you can provide comma-separated strings. The `print()` function requires that you provide *one* string argument; therefore, to work with multiple strings, you must use the string concatenation operator (`.`).

```
echo "Hello, world! ", "Goodbye, world!";
print "Hello, world! " . "Goodbye, world!";
```

You will find several constructs, such as `echo()`, in PHP that *act* like functions, even though they are technically not. In these cases, the interpreter does not require parentheses; however, using the parentheses is also acceptable. The following two statements are equivalent:

```
print "Hello, world! " . "Goodbye, world!";
print("Hello, world! " . "Goodbye, world!");
```

NOTE: Take care when using quotation marks for strings. If you do not use quotation marks correctly – for example, `print(Hello");` – the script may not execute properly. Even worse, you may not get any output in the browser whatsoever.

Writing HTML to the browser is essentially no different than writing simple text. To print the double-quotation mark, escape it within the string using `\"`.

```
print "<h2>Useful Links</h2>";
print "<p><a href=\"http://www.google.com\">Google</a></p>";
```

Any text that would exist in the Web page (e.g., HTML elements, cascading style sheets, JavaScript) can be sent to the browser using printing functions.

Whitespace

As with HTML whitespace (e.g., blank lines, tabs, extra spaces) is generally ignored. Even though whitespace does not affect how the user sees the page, indentation helps makes the code more readable, and thus easier to debug and maintain.

> *NOTE*: Whitespace within a string literal is preserved, even if the browser ends up ignoring it. In the following example, even though the string contains whitespace that PHP preserves, the browser will display "Dr. Donald Knuth", having collapsed the whitespace.

```
<h2>About Me</h2>
<?php
  print("Dr.     Donald   Knuth");
?>
```

Comments

Comments are an excellent way of documenting your code – for yourself and others. An advantage of PHP comments over HTML comments is that PHP comments are never sent to the browser. Comments are denoted using the same syntax as C, C++, or Perl: `/* */`, and `//`, and `#` respectively.

```
<h2>About Me</h2>
<?php
  /* I'll put more useful
     PHP code here later. */
  print("Computer Science person of great fame.");  // Ego boost!
  # die "Why not use Perl" if $lang ne "perl";
?>
```

The C++ style comment is preferred; however multi-line comments (`/* */`) are useful for commenting out large blocks of code for debugging purposes.