

Navigation

Home

Updates

Courses

> Client-side programming

> Server-side programming

Lessons (All)

Labs (All)

[Lessons \(All\)](#) >

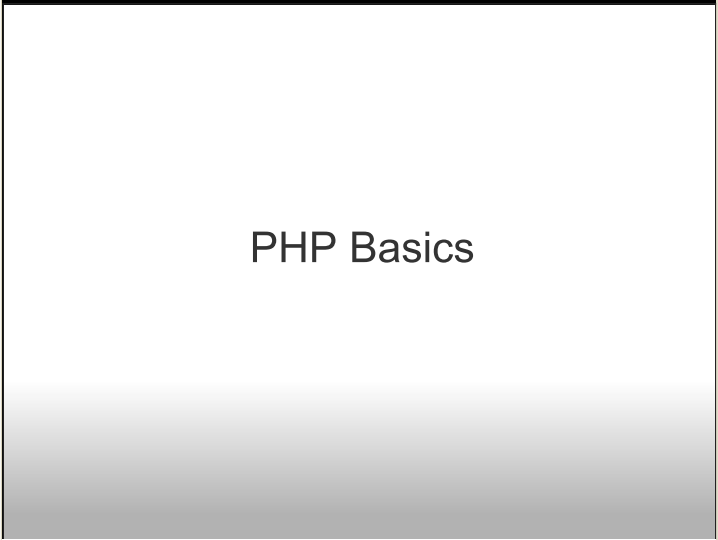
PHP Basics

The notes below provide a complete description of the PHP programming language and is meant to be used as a reference during the following exercises. If you're looking for a more cursory overview, please see the included slide set instead.

- 1. Prerequisites
- 2. Running PHP
- 3. Philosophy
- 4. A Basic PHP program
- 5. Variables And Functions
- 6. Classes And Objects
- 7. Strings And String Handling
- 8. Arrays And Dictionaries
- 9. Conditional Execution
- 10. require And require_once
- 11. User Input & Form Handling
- 12. File Handling
- 13. Database Access
- 14. Accessing Web Services

Slide set

PHP Basics



Prerequisites

This lesson assumes that you have prior experience programming, preferably using C/C++ or Java.

To complete this lesson, you'll also need:

- A webserver. The [Apache HTTP Server](#) is recommended.
- [PHP](#) version 5 or greater
 - Your webserver must be configured to execute PHP scripts using the PHP interpreter.
 - The following modules must be enabled: PDO (with SQLite support)
 - It is recommended that you compile PHP with CLI and readline support.

If you use a Windows OS, you may satisfy the above requirements by downloading [The Uniform Server](#).

Similarly, if you use Mac OS X, you can download [MAMP](#).

If you have a web hosting provider that supports running PHP scripts, you can use that for the purposes of this lesson. However, setting up a local development server is a good idea due to lower latency and less risk of a poorly written application impacting server performance.

Running PHP

HTML-embedded PHP

PHP is primarily used as language for web development. As such, most interaction with PHP will occur indirectly, via a web browser. However, unlike JavaScript, web browsers do not understand how to execute PHP code. Instead, any PHP code used in a web page must be executed on the server and must output standard HTML, which the web server will return to the client web browser.

While this requires some extra work on the part of server administrators and developers, it provides developers the freedom to access protected resources, as well as implement access control mechanisms. The most common use of this functionality is to access a shared database, providing web applications with global state across geographically distributed client sessions.

This pattern for web development is often referred to as the **three-tier model**, where the three tiers are the web server, interpreter (PHP), and database. s By default, any filename ending in `.php` on the server will be automatically passed through the PHP interpreter when requested. The interpreter is then responsible for telling the web server what should be returned to the client web browser.

You can test this by creating a file named `test.php` inside of your web server's document root. The contents of this file should contain the following two lines of text:

```
<?php
    phpinfo();
```

If you were to load this file in your web browser, you'll see nothing special. However, if you access this file through your web server (for example, by going to <http://localhost/test.php>), you would see output similar to the following:

In this case, `phpinfo()` is a special command that tells PHP to output diagnostic information. We'll get to the significance of the `<?php` command later in the lesson.

Interactive PHP

If you are familiar with other interpreted languages, such as Ruby or Python, you may be used to interacting with the interpreter directly via a command line tool.

If you have PHP compiled with CLI and readline support, you can launch an interactive version of PHP by running `php -a` from a command line. After doing so, the following prompt should be displayed:

```
$ php -a
  Interactive shell

php >
```

Note: If you receive a "command not found" error, then the PHP CLI client is not installed or not in your shell's include path. If PHP launches but does not display a prompt, then

you do not have readline support installed.

Any commands entered will be executed immediately and the output displayed on the command line. To exit, type `exit` and press enter.

While you may find this useful for testing and experimentation purposes, access to the command line client is not required to complete this lesson.

Philosophy

PHP is a unique language in that there's no enforced file structure and you can freely mix HTML and PHP code (as we'll explore momentarily). While this makes it very easy to get started developing web applications, it also makes it easy to write spaghetti code that's difficult to maintain later.

In this spirit, there are a few design goals that have been woven into this lesson:

1. Think Ahead (Reusability): Don't unnecessarily couple code. Maintain a separation between the code that renders your view (HTML) and the rest of your application logic.
2. Stay Lean (Simplicity): Keep individual functions short, and defer responsibility to other functions whenever possible.
3. Don't Reinvent The Wheel (Abstraction): Unlike low-level languages (such as C/C++), PHP includes functions for most common tasks as part of the SPL (Standard PHP Library). Using these functions not only saves you time, it makes your code easier for others to understand.

We hope that you'll continue to apply these principles to your code outside of this tutorial.

A Basic PHP Program

The first thing to know about writing a PHP program is that the PHP interpreter will not do anything unless you ask it to.

To demonstrate, create a file named `hello.php` inside of your web server's document root. Inside, paste the following:

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

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>My First PHP Script</title>
</head>
<body>
    <h1>This Is My First PHP Script!</h1>
</body>
</html>
```

Load this file from your webserver. If you aren't using a webserver on another computer, this would be done by navigating to <http://localhost/hello.php>.

If successful, you should see a page that says "This Is My First PHP Script!". This is, indeed a valid PHP script. However, we haven't done anything special. This is because PHP hasn't been *told* to do anything yet, so it just output the contents of `hello.php` verbatim.

Let's change that. Add the following block of code before `</body>`:

```
<?php
    echo("<p>Hello world!</p>");
?>
```

Save the file, then reload the page in your web browser. Notice that the text "Hello world!" has been output. If you view the page's source, however, only "`<p>Hello world!</p>`" appears.

PHP reads files from top to bottom. When it encounters the string `<?php`, it treats everything that follows it as PHP code, and stops once `?>` is encountered. The `echo()` function tells PHP to emit the specified string in the HTML response. Finally, notice that every line of PHP code ends with a semicolon.

Interpreted vs Compiled

If you're used to working in a compiled language, like Java or C/C++, you'll notice that we didn't need to compile the PHP code above before executing it. This is because PHP is an interpreted language: it's translated into machine code as it's executed.

From a development standpoint, this means that it's easier to write code in an iterative fashion. There's no time penalty to testing your code as you write it. However, this also means that if there's any errors in your code, they may not be detected until the offending line of code is executed.

For example, consider the following block of code:

```
<?php
    $x = true;
    if ($x == true) {
        echo("I think $x is true.");
    } else {
        echo("I think $x is false.");
    }
}
```

Even though the second invocation of `echo()` is misspelt, this code will always run because the line containing the error will never be executed (`$x` is always `true`).

Variables, Functions, and Comments

The previous examples introduced two common programming constructions: functions and variables.

Like in Java and C/C++, functions in PHP consist of a letter or underscore, followed by any number of letters, numbers, or underscores. The arguments to the function are placed inside a set of parenthesis.

For example, all of the following are valid function names:

```
echo("Hello");
doSomething();
_foo2($x, "Bar");
```

You can define your own functions by using the `function` keyword:

```
function doSomething() {
    echo("This function does something useless.");
}
```

Variables in PHP almost always begin with a `$` symbol, and follow the same naming conventions as functions. Variables can also be passed between functions as arguments:

```
function deliverMessage($text) {
    echo("I was asked to deliver the following message:<br />");
    echo($text);
}
```

The exception to this rule are constants, which do not require a `'$'`. Constants are like any other variable, except that they are immutable — their value cannot be changed after creation. To create a constant, you need to use the `define()` function.

```
define("MY_CONSTANT", 42);
echo(MY_CONSTANT);
```

The above block of code would output the number 42.

You can specify multiple arguments. You can assign arguments default values, too.

```
function deliverMessage($text, $recipient, $sender = "Anonymous") {  
    echo("I was asked to deliver the following message:<br />");  
    echo("To:" . $recipient . "<br />");  
    echo("From:" . $sender . "<br />");  
    echo($text);  
}
```

If we invoke this function with `deliverMessage("Hello", "Jane")`, the following will be output:

```
I was asked to deliver the following message:<br />  
To: Jane<br />  
From: Anonymous<br />  
Hello
```

Finally, you can insert single-line comments in PHP by using the `//` or `#` operators. Multi-line comments can be inserted by using `'/'` *at the beginning of the comment* and `'/'` at the end of the comment.

```
// This is comment. It won't be executed.  
# This is also comment. It won't be executed.  
/* This is a multi-line comment.  
   Nothing will be executed here.  
   This line ends the multi-line comment. */
```

Scope

Like in C/C++ and Java, variables are only accessible within the function they were defined in, unless these variables are declared as global. This is done in PHP using the `global` keyword. Any variables defined at the top level of a file (in other words, they aren't inside of a function or class) are automatically global.

```
$foo = "foo"; # This is global
```

```
function doSomething() {  
    $bar = "bar"; # This can only be accessed inside of doSomething()  
    global $baz = "baz"; # This is global  
}
```

However, unlike in C/C++ and Java, marking a variable as global will, in itself, make it visible within another scope. For this to occur, the function must be redeclared as global inside of each scope it is used.

```
function functionA() {  
    global $foo;  
    $foo = "Variable is global";  
    echo("Function A:" . $foo);  
}  
  
function functionB() {  
    $foo = "Variable is local";  
    echo ("Function B: " . $foo);  
}  
  
function functionC() {  
    global $foo;  
    echo("Function C: " . $foo);  
}
```

```
}  
  
functionA();  
functionB();  
functionC();
```

Type

PHP is a loosely typed language. While variables do have an implicit type (such as integer, string, object) determined and stored internally by the interpreter, this can *usually* be safely ignored by developers. PHP will automatically cast (convert) between datatypes as needed.

For example, the following code will output "true":

```
$a = 42;    # Integer  
$b = "42"; # String  
if ($a == $b)  
    echo("true");  
else  
    echo("false");
```

For reference, the internal types supported by PHP are:

- **NULL**: A variable representing 'nothing'.
- **integer**: A scalar valued number.
- **float**: A real valued number.
- **boolean**: A true/false value.
- **string**: A series of one-byte characters. (PHP 6, which is currently in development, plans to add native Unicode [multi-byte character] support.)
- **array**: An ordered collection of elements.
- **object**: An instance of a class.
- **resource**: A special variable that holds a reference to an external resource, such as a file pointer or a database connection. (In most cases, the use of objects is recommended over resources.)

Memory Management

PHP, like Java, implements automatic management of program memory. When a variable is no longer needed, the interpreter's garbage collector will automatically deallocate the memory that was assigned to that variable.

For example, the equivalent to the following program would contain a memory leak under C++, but is perfectly acceptable under PHP

```
$a = new Foo(); # Foo is a custom class  
$a = new Bar(); # Foo was not explicitly deleted, but PHP will detect that Foo is no longer referenced and automatically delete it
```

Classes and Objects

If you've done any work in an object-oriented programming language before, you'll be familiar with classes. Classes are collections of variables (called parameters) and functions (called methods) that know how to operate on those variables. When you want to use a class, you *instantiate* it. Each instance of a class is known as an *object*. Data stored in one object cannot be accessed by another object, even if they're instances of the same class.

Classes can be created by using the **class** keyword, followed by a class name. You can then define parameters using the **var** keyword, and methods using the **function** keyword (as before).

```
class Animal {  
    var $species;  
  
    function speak() {
```

```

        if ($species == "Dog")
            echo("Arf!");
        else
            echo("?");
    }
}

```

To instantiate a class, the `new` keyword is used. To access the members (parameters and methods) of a class, use the `->` operator.

```

$myDog = new Animal();
$myDog->species = "Dog";
$myDog->speak();

```

A full treatment of object-oriented programming is outside the scope of this lesson. However, more documentation on using classes is [available as part of the PHP documentation](#).

Strings And String Handling

There are several notations for strings that can be used in PHP, depending on preference and the way in which the string is being used.

The method used up to this point was to declare a string using double-quotation marks:

```
$s = "This is a string";
```

However, you can also declare a string using single-quotation marks:

```
$s = 'This is a string';
```

In C/C++ and Java, where single quotation marks indicate a character literal. This is not the case in PHP. Instead, the difference is in how variables within the string are expanded.

Since variables in PHP are indicated using a '\$' character, it's possible to reference them directly within a string:

```

$name = 'Cody';
echo("My name is $name.");

```

If this isn't desired, then single-quotes should be used instead:

```

$name = 'Cody';
echo('My name is $name.');
```

As mentioned above, strings in PHP are a primitive variable type. This may come as a surprise to user of object-oriented languages such as Java or C++, but should be familiar to users of C. The implication of this is that strings are not objects and do not have instance methods, and as a result manipulation must be done by global functions. A complete list of functions are [available as part of the PHP manual](#).

Also, note that strings can be concatenated (combined) using the '.' operator. You've seen this in several previous examples already, but in case you missed it:

```

$name = 'Cody';
echo('My name is' . $name);

```

Arrays And Dictionaries

Arrays are used to store ordered collections of information, and are created using the `array()` function. In their most basic form, each element within the array is assigned an integer index, which is determines that element's position within the array.

```
$a = array('a', 'b', 'c', 'd', 'e');
```

Given an index, you can extract a given value from the array using the '[]' operator:

```
echo($a[3]);
```

You iterate through all of the elements in the array in a similar fashion:

```
for ($i = 0; $i < count($a); $i++) {  
    echo($a[$i]);  
}
```

Since this is so common, PHP also has a special `foreach()` construct which simplifies the above statement:

```
foreach ($a as $item) {  
    echo $item;  
}
```

However, you're not limited to using integers as the index. Arrays also operate as associative arrays, otherwise known as maps (in Java and C/C++), dictionaries, or hash tables. In other words, you can input arbitrary key/value pairs into the array:

```
$colors = array('apple' => 'red', 'banana' => 'yellow', 'lime' => 'green');  
echo('An apple is' . $colors['apple']);  
  
foreach ($colors as $key => $value) {  
    echo('An $key is $value.');}
```

Conditional Execution & Operators

We've already used a few flow control structures, namely the `if` statement, the `for` statement, and the `foreach` statement. In general, most flow control statements in PHP mirror those used in C/C++ and Java.

In the interest of brevity, the list of control structures has been omitted. If you are interested in learning about the available control structures, or want more information, you can [view the appropriate section in the PHP manual](#).

Similarly, most operators (such as the comparison operators) mirror those used in C/C++ and Java. Noteworthy differences include:

- **Comparison operators:** In addition to `==` and `!=`, there are also `===` and `!==` operators. These operators test for exact equality (or inequality). For example, 42 is `==` to '42', but 42 is not `===` to '42'. (See the section on types, above.)
- **Error suppression operators:** Some PHP functions will emit error text into rendered HTML in the event of failure. If you implement custom error handling code, you can suppress this text by prefixing the relevant function call with `@`.
- **String concatenation:** As noted above, the `.` operator is used to combine strings together.

Importing Code

Occasionally, you may find it useful to reference classes and functions contained in other files. In these cases, you can use the `require_once()` function to read the given file into your application.

For example, consider a file named `factorial.php` that contains the following:

```
<?php
```



```
function factorial($i) {
    if ($i != 1)
        $i *= factorial($i - 1);
    return $i;
}
```

Now, let's say that we want to use this file as part of code that has already been written (say in a file named `index.php`). To do so, we'd do the following:

```
<?php
    require_once('factorial.php');
    echo("The factorial of 5 is:" . factorial(5));
```

Even though `factorial()` isn't defined in `index.php`, since we included `factorial.php`, that function becomes available to us.

Note that calling `require_once` simply reads the specified function into the current file. If there are any commands that aren't inside of a function or class, they will be executed immediately.

Note: Notice that we didn't include a trailing `?>` at the end of `factorial.php`. If we did, PHP would insert a newline corresponding to any newlines at the end of the file, which are easy to overlook. PHP will automatically insert the closing tag for us.

User Input & Form Handling

Eventually, you'll want to receive user input from your PHP script. In the case of HTML, this is usually accomplished using forms.

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

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Enter Your Name</title>
</head>
<body>
    <form action="submit.php" method="get" accept-charset="utf-8">
        <p><label for="name">Enter your name: </label><input type="text" name="name" id="name">
        <p><input type="submit" value="Continue &arr;"></p>
    </form>
</body>
</html>
```

When this form gets submitted, the data will be sent as an HTTP GET to `submit.php`. The `action=` attribute specifies the destination, and the `method=` line specifies whether to use HTTP POST or GET. HTTP GET operations send data as part of the URL, which is convenient for short submissions, such as search fields. Per [RFC 2616](#), HTTP GET operations must *never* perform an action other than data retrieval.

The contents of this form are available via one of two special “superglobal” arrays: `$GET` or `$POST` (depending on the form's method). These arrays are called superglobal because they're always available, even if you haven't declared them using the `global` keyword.

The following PHP code would display the user's name as entered in the form:

```
echo("Your name is: " . $_GET['name']);
```

Note: There's a third superglobal array you may find useful: `$_SERVER`. This contains data supplied from the webserver about the request itself. For example, any HTTP headers provided by the client would be available in this array.

File handling

PHP allows developers to open and manipulate any file accessible by the web server. (You can't access files on the client web browser's host, since that would be a security problem.)

Files are accessed using the `fopen()` function, which returns a file pointer. When you are done with the file, you should release the file pointer by calling `fclose()`. You can read a line from the file into a string by calling `fgets()`, and test to see if you're at the end of the file by calling `feof()`.

```
$myFile = fopen("myFile.txt", "r");
while (!feof($myFile)) {
    print(fgets($myFile));
}
fclose($myFile);
```

The above example opens the file "myFile.txt" for reading, and then prints out each line of the file, until it eventually reaches the end of the file.

Database access

Instead of managing your own files, for many tasks it can be easy to use a relational database to store application data. While the specifics of using SQL to interact with a relational database system is outside the scope of this lesson, the following code can be used to access a SQLite database.

We've chosen SQLite for this example because it doesn't require a dedicated server process. However, similar instructions can be used to connect to MySQL and PostgreSQL databases, among others.

```
$this->db = new PDO("sqlite:myDatabase.sqlite3");
# myDatabase.sqlite3 must already have a schema, otherwise we'll get an error when we try to use it
$query = $this->db->prepare("INSERT INTO lolcats (date, url) VALUES (:date, :url)");
$date = date(DATE_RFC822);
$url = "http://www.mylolcat.com"
$query->bindParam(":date", $date);
$query->bindParam(":url", $url);
$query->execute();

$query = $this->db->prepare("SELECT * FROM lolcats");
$query->execute();
while ($row = $query->fetch()) {
    echo("Date: " . $row['date'] . " URL: " . $row['url']);
}

$this->db->close();
```

Accessing web services

While also partially outside the scope of this lesson, it's worth mentioning how to access external resources on the web.

The following code sample connects to icanhascheezburger.com and retrieves the URL for the latest image that was posted:

```
$url = "http://icanhascheezburger.com";
$html = file_get_contents($url);
$dom = new DomDocument();
@$dom->loadHTML($html);
$xpath = new DOMXPath($dom);
$img = $xpath->evaluate("//div[@class='entry']/img")->item(0);
$imgSrc = $img->getAttribute('src');
print("<img src=\"$imgSrc\" />");
```

Online Help

Documentation for all of PHP's methods are available at [PHP.net](http://php.net). In general, you can access the documentation for any method by going to <http://php.net/symbol>, where `symbol` is the name of the function you want to lookup.

Comments

Commenting disabled due to a network error. Please reload the page.

You do not have permission to add comments.

[Sign in](#) | [Recent Site Activity](#) | [Report Abuse](#) | [Print Page](#) | Powered By [Google Sites](#)