|  |
| --- |
|  |
| Introduction to PHP and MySQL |
|  |

|  |
| --- |
| Andrew Taylor  @roohtaylor |

Contents

[Course Outline 2](#_Toc371490939)

[Client-Server Model 3](#_Toc371490940)

[PHP Files and Basic Syntax 3](#_Toc371490941)

[Function Syntax 4](#_Toc371490942)

[Operators 4](#_Toc371490943)

[Arithmetic Operators 4](#_Toc371490944)

[Assignment Operators 5](#_Toc371490945)

[Comparison Operators 5](#_Toc371490946)

[String Operators 5](#_Toc371490947)

[Logical Operators 5](#_Toc371490948)

[Label Rules 6](#_Toc371490949)

[Variables 6](#_Toc371490950)

[Data Types 6](#_Toc371490951)

[Integers 6](#_Toc371490952)

[Strings 6](#_Toc371490953)

[Booleans 6](#_Toc371490954)

[Floating Point Numbers 6](#_Toc371490955)

[Type Juggling 7](#_Toc371490956)

[Arrays 7](#_Toc371490957)

[Conditional Statements 8](#_Toc371490958)

[Loops 8](#_Toc371490959)

[While 8](#_Toc371490960)

[Do-while 9](#_Toc371490961)

[For 9](#_Toc371490962)

[Foreach 10](#_Toc371490963)

[License 11](#_Toc371490964)

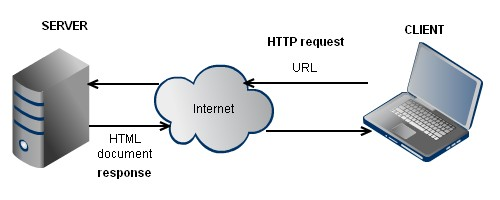
|  |  |
| --- | --- |
| Domain: | FTP Port: 21 |
| Username: | cPanel Login: cpanel. |
| Password: |  |

# Course Outline

* Intro
* What is PHP?
* PHP Files and “Hello World!”
* Delimiters
* Comments
* Variables and Operators
* Data Types
* Type Juggling
* Functions, Parameters, and Return Values
* PHP.net
* Conditional Statements
* User Defined Functions
* Variable Scope
* Arrays
* Loops

# 

# Client-Server Model



\*\* PHP execution occurs on the server before the response document is sent back to the client.

# PHP Files and Basic Syntax

PHP files are parsed and executed in real-time on the server, so they’re formatted just like text files when they are written and stored. PHP files can have several different extensions to support different conditions, but the best extension to use is .php.

<?php

?>

Don’t forget to use delimiters around your PHP code. Only code between the delimiters is executed. Anything outside the delimiters is treated as plaintext and passed through untouched.

<?php

echo ‘Hello World!’;

# This is a single line shell style comment

// This is also a single line C++ comment

/\* This

Is

A

Multi-line

Comment

\*/

?>

Remember to end each instruction with a semi-colon. Your script will not execute and throw errors if you forget one.

Don’t forget to add comments in your code to help yourself and others. Comments are ways of explaining what each section of your code is doing. There are three styles of comments in PHP and you can use them all interchangeably.

# Function Syntax

<?php

functionname(parameter1, parameter2, etc);

?>

Remember to use or capture the return value of a function if required. Parameters are separated by a single comma if there is more than one.

# Operators

## Arithmetic Operators

|  |  |  |
| --- | --- | --- |
| Example | Type | Result |
| -5 | Negation | Opposite of 5 [(int)-5] |
| 5 + 5 | Addition | Sum of 5 and 5 [(int)10] |
| 5 - 3 | Subtraction | Difference of 5 and 3 [(int)2)] |
| 3 \* 4 | Multiplication | Product of 3 and 4 [(int)12] |
| 3 / 5 | Division | Quotient of 3 and 5 [(float)0.6] |
| 5 % 3 | Modulus | Remainder of 5 divided by 3 [(int)2] |

## Assignment Operators

|  |  |
| --- | --- |
| Assignment | Same as |
| $a = $b | N/A |
| $a += $b | $a = $a + $b |
| $a -= $b | $a = $a - $b |
| $a \*= $b | $a = $a \* $b |
| $a /= $b | $a = $a / $b |
| $a %= $b | $a = $a % $b |
| $a .= $b | $a = $a . $b |

## Comparison Operators

|  |  |  |
| --- | --- | --- |
| Example | Name | Result |
| $a == $b | Equal | TRUE if $a is equal to $b after type juggling. |
| $a === $b | Identical | TRUE if $a is equal to $b, and they are of the same type. |
| $a != $b | Not equal | TRUE if $a is not equal to $b after type juggling. |
| $a <> $b | Not equal | TRUE if $a is not equal to $b after type juggling. |
| $a !== $b | Not identical | TRUE if $a is not equal to $b, or they are not of the same type. |
| $a < $b | Less than | TRUE if $a is strictly less than $b. |
| $a > $b | Greater than | TRUE if $a is strictly greater than $b. |
| $a <= $b | Less than or equal to | TRUE if $a is less than or equal to $b. |
| $a >= $b | Greater than or equal to | TRUE if $a is greater than or equal to $b. |

## String Operators

|  |  |  |
| --- | --- | --- |
| Example | Name | Result |
| $a . $b | Concatenation | Connect the two strings into one |

## Logical Operators

|  |  |  |
| --- | --- | --- |
| Example | Name | Result |
| $a and $b | And | TRUE if both $a and $b are TRUE. |
| $a or $b | Or | TRUE if either $a or $b is TRUE. |
| $a xor $b | Xor | TRUE if either $a or $b is TRUE, but not both. |
| ! $a | Not | TRUE if $a is not TRUE. |
| $a && $b | And | TRUE if both $a and $b are TRUE. |
| $a || $b | Or | TRUE if either $a or $b is TRUE. |

The reason for the two different variations of "and" and "or" operators is that they operate at different precedences. (See Operator Precedence on PHP.net.)

# Label Rules

A valid label in PHP starts with a letter or underscore, followed by any number of letters, numbers, or underscores. This applies to variables, functions, objects, and constants. Generally speaking, a label is PHP should be descriptive. For example, a function name should be descriptive of what the function does; likewise, a variable name should be descriptive of what that variable contains.

# Variables

PHP variables are used to store information during the execution of a script. Variable syntax is as follows.

<?php

$int = 10;

$str = ‘This is a String’;

$varname = ‘value’;

?>

# Data Types

## Integers

An integer is a whole number. -3, -2, -1, 1, 2, 3, 4, 5 … are examples of integers. An integer is typed literally and does not need quotes around it.

## Strings

A String is text. A string is defined with quotations. Strings that are defined with double quotes instead of single quotes will automatically expand variables and special characters like newlines (\n).

## Booleans

Booleans are the simplest data type. Booleans are true or false, and defined using those constants.

## Floating Point Numbers

Also known as “floats”, “doubles”, or “real numbers”. Floats are decimal or exponential numbers. We aren’t going to be working with them today, but if you would like to know more about them you can read about them on the PHP.net website.

<?php

$foo = true; //Type Boolean bool(true)

$foo = 10; //Type Integer int(10)

$foo = “10”; //Type String String(2)

$foo = 3.123; //Type Float float(3.123)

?>

## Type Juggling

PHP variables are scalar. This means that they do not need to have a data type explicitly defined, and that the data type of a variable will change depending on the context in which it is used. Let’s play around with it a little.

<?php

$a = ‘0’; // Is String(1) “0”

var\_dump($a);

$a = $a + 2; // Is now int(2)

var\_dump($a);

$a = 5 + “10 Little Piggies”; // Is int(15)

var\_dump($a);

?>

# Arrays

Arrays are a special data type that contain an index of data. An easy way to think about it is that arrays are a list of variables contained within a single variable. In PHP arrays can be indexed numerically or with a String. Arrays that have a String index are called associative arrays, but are also referred to as “hash tables” by some. A single array can have both numerical and String keys at the same time.

<?php

$a = array(‘value1’, ‘value2’, ‘value3’); //Index starts at (int)0

echo $a[0];

$a[0] = ‘new value1’; //Only sets index (int)0

$a[‘0’] = ‘Different than (int)0’; //String key of ‘0’ is different than (int)0

$b = array(‘key1’ => ‘value1’,

‘key2’ => ‘value2’,

‘key3’ => ‘value3’

);

$b[] = ‘blank’; //Append to the array. Takes next available (int)key

print\_r($a);

print\_r($b);

?>

Arrays are indexed with integers starting with 0 by default. If you set only String keys and then append to the array it will take the next available integer value, which would be 0. If you set integer key 5 and then append it will take the next available integer key, which it will assume is 6.

Access the data within an array by referencing it’s key within square brackets after the array name.

# Conditional Statements

<?php

$foo = true;

if (!$foo){

echo ‘$foo was false!’;

} elseif ($foo){

echo ‘$foo was true!’;

} else {

echo ‘$foo was neither true nor false!’;

}

?>

Conditional statements are used to execute specific code if certain conditions are met.

# Loops

Often times there is a requirement to repeat a certain section of code a number of times before carrying on. Rather than typing out that same section of code over and over again, we can use loops to do it for us.

## While

The while loop is the most basic of loops. The while loops checks for a condition, and as long as the condition proves true it will continue to execute the code within its braces. The condition check happens at the start of each loop iteration, which means the while loop will only execute if the condition proves true at the beginning. It also means that if the condition changes from true to false in the middle of an iteration, the code will continue to execute until the end of that iteration.

<?php

$i = 0;

while ($i <= 10){

echo $i . ‘<br>’;

$i++;

}

?>

## 

## Do-while

Do-while loops are basically the same thing as while loops. The exception being that a do-while checks the condition at the end of a loop iteration. This means that the code within the braces is executed once before the condition is checked.

<?php

$i = 0;

do {

echo $i . ‘<br>’;

$i++;

} while ($i < 0);

?>

This code will only execute once. Once the condition is checked it will prove false and not execute again. Do-while loops also support the conditional break statement. The break statement will end execution of the loop code and proceed with the rest of the script.

## For

For loops are one of the most complex loop types. For loops evaluate/execute three expressions as they run, and can be used any number of ways.

The first expression is evaluated/executed once at the beginning of the for loop. The second expression is evaluated at the beginning of each loop iteration. If it evaluates to true, the loop continues; otherwise the loop ends. The third expression is evaluated/executed at the end of each loop iteration. Each of the expression groups can have multiple expressions separated by a comma, or they can be left empty. If the second expression is left empty it defaults to true and the loop will run indefinitely.

For loops also support the break statement, to end execution at any point during the iteration.

<?php

/\* example 1 \*/

for ($i = 1; $i <= 10; $i++){

echo $i;

}

/\* example 2 \*/

for ($i = 1; ; $i++){

if ($i > 10) {

break;

}

echo $i;

}

/\* example 3 \*/

$i = 1;

for (; ; ) {

if ($i > 10){

break;

}

echo $i;

$i++;

}

/\* example 4 \*/

for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);

?>

Each of these examples will output the numbers 1 through 10.

## Foreach

A foreach loop is specifically purposed to loop through array and object values. An error will be generated if you try to run a foreach loop that references a non-array or object. It steps through each value of an array, and executes the code in its braces. The two syntaxes provide you with access to both the array key and the value, depending on which you use.

<?php

$i = array(‘value1’, ‘value 2’, ‘value3’);

foreach ($i as $value){

echo $value . ‘<br>’;

}

?>

A foreach loop uses an array’s internal pointer to iterate through each value, so changing the pointer during the loop could have unexpected results. In this example, only the value of each array index is available. The following shows the alternate syntax allowing access to the key as well.

<?php

$i = array(‘value1’, ‘value 2’, ‘value3’);

print\_r($i);

foreach ($i as $key => $value){

$i[$key] = ‘newvalue’ . $key;

}

print\_r($i);

?>

# License

This document and all supporting documents are licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported license.