Introduction to Web Development

Chapter 3:

Book: Learning PHP, MySql and JavaScript

Dr. Mohamed ElGazzar

Introduction to PHP

- PHP is the language that you use to make the server generate dynamic output—output that is potentially different each time a browser requests a page.
- In production, your web pages will be a combination of PHP, HTML, and JavaScript, and some MySQL statements laid out using CSS, and possibly utilizing various HTML5 elements.
- Furthermore, each page can lead to other pages to provide users with ways to click through links and fill out forms.

Incorporating PHP within HTML

- By default, PHP documents end with the extension .php
- When a web server encounters this extension in a requested file, it automatically passes it to the PHP processor.
- Your PHP program is responsible for passing back a clean file suitable for display in a web browser.
- At its very simplest, a PHP document will output only HTML
- To prove this, you can take any normal HTML document such as an index.html file and save it as index.php, and it will display identically to the original.

Incorporating PHP within HTML

- To trigger the PHP commands, you need to learn a new tag. Here is the first part:

<?php

- The entire sections of PHP can be placed inside this tag, and they finish only when the closing part is encountered, which looks like this:

?>

- Example:

```
<?php
echo "Hello World";
?>
```

The structure of PHP

Using comments:

- There are two ways in which you can add comments to your PHP code. The first turns a single line into a comment by preceding it with a pair of forward slashes:

```
// This is a comment
```

- When you need multiple-line comments, there's a second type of comment, which looks like

```
<?php
/* this section consists of
Multiple
Line
Comments */
?>
```

Basic Syntax

Semicolon

- The PHP commands ended with a semicolon, like this:

```
$x += 10;
```

The $ symbol:

- you **must** place a $ in front of all variables. This is required to make the PHP parser faster, as it instantly knows whenever it comes across a variable.

```
<?php
$mycounter = 1;
$mystring = "hello";
?>
```

Basic Syntax

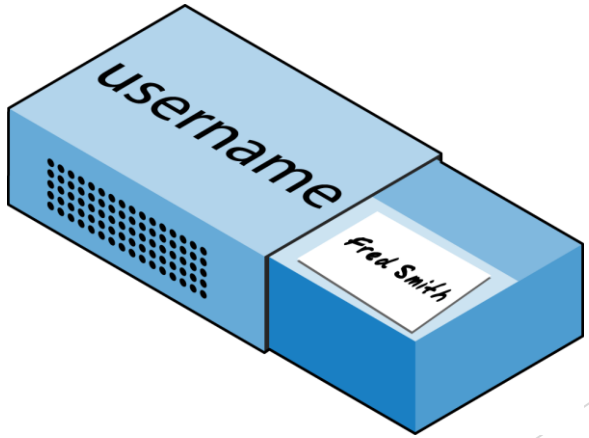
Variables

- There's a simple metaphor that will help you understand what PHP variables are all about. Just think of them as little (or big) matchboxes! That's right—matchboxes that you've painted over and written names on.

- **String variable:**

Imagine you have a matchbox on which you have written the word username. You then write Fred Smith on a piece of paper and place it into the box:

$username = "Fred Smith";



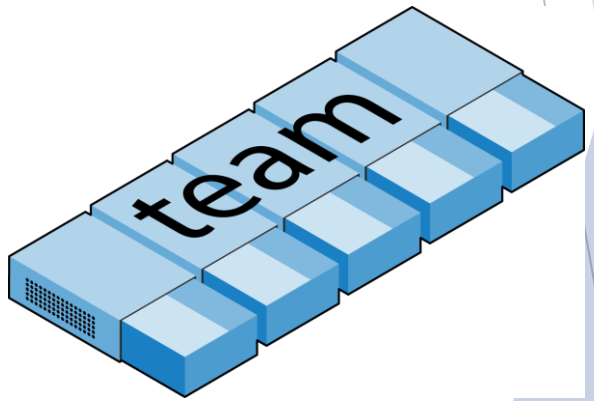
Basic Syntax

Variables

- **Numeric variables:**

Variables can also contain numeric values such as

```
$count = 17;
$count2 = 17.5;
```



- **Arrays:**

You can think of them as several matchboxes glued together.

For example, let's say we want to store the player names for a five-person soccer team in an array called $team. To do this, we could glue five matchboxes side by side and write down the names of all the players on separate pieces of paper, placing one in each matchbox.

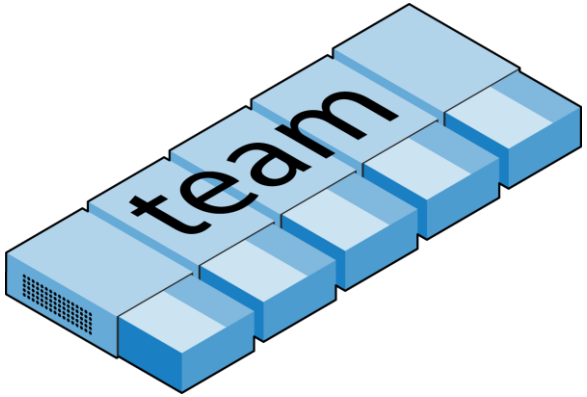
Basic Syntax

Variables

- Arrays:

```
$team = array('Bill', 'Mary', 'Mike', 'Chris', 'Anne');


echo $team[3]; // Displays the name Chris
```



- Two dimensional arrays:

let's say we want to keep track of a game of tic-tac-toe, which requires a data structure of nine cells arranged in a 3×3 square. To represent this with matchboxes, imagine nine of them glued to each other in a matrix of three rows by three columns

```
<?php
$oxo = array(array('x', ' ', 'o'),
array('o', 'o', 'x'),
array('x', 'o', ' '));
echo $oxo[1][2];
?>
```

Basic Syntax

Variables Naming Rules:

When creating PHP variables, you must follow these four rules:

• Variable names must start with a letter of the alphabet or the _ (underscore) character.

• Variable names can contain only the characters a-z, A-Z, 0-9, and _ (underscore).

• Variable names may not contain spaces. If a variable must comprise more than one word, it should be separated with the _ (underscore) character (e.g., $user_name).

• Variable names are case-sensitive. The variable $High_Score is not the same as the variable $high_score.

Basic Syntax

Operators:

Operators are the mathematical, string, comparison, and logical commands such as plus, minus, multiply, and divide. PHP looks a lot like plain arithmetic; for instance, the following statement outputs 8:

```
6 + 2
```

Basic Syntax

Arithmetic Operators:

Arithmetic operators do what you would expect. They are used to perform mathematics. You can use them for the main four operations (plus, minus, times, and divide) as well as to find a modulus (the remainder after a division) and to increment or decrement a value.

Operator	Description	Example
+	Addition	`$j + 1`
-	Subtraction	`$j - 6`
*	Multiplication	`$j * 11`
/	Division	`$j / 4`
%	Modulus (division remainder)	`$j % 9`
++	Increment	`++$j`
--	Decrement	`--$j`

Basic Syntax

Assignment Operators:

- These operators assign values to variables. They start with the very simple = and move on to +=, -=, and so on
- The operator += adds the value on the right side to the variable on the left, instead of totally replacing the value on the left.

- Thus, if $count starts with the value 5, the statement:

$count += 1;

- sets $count to 6, just like the more familiar assignment statement:

$count = $count + 1

Operator	Example	Equivalent to
=	$j = 15	$j = 15
+=	$j += 5	$j = $j + 5
-=	$j -= 3	$j = $j - 3
*=	$j *= 8	$j = $j * 8
/=	$j /= 16	$j = $j / 16
.=	$j .= $k	$j = $j . $k
%=	$j %= 4	$j = $j % 4

Basic Syntax

Comparison Operators:

- Comparison operators are generally used inside a construct such as an if statement in which you need to compare two items.
- For example, you may wish to know whether a variable you have been incrementing has reached a specific value, or whether another variable is less than a set value, and so on.

Operator	Description	Example
==	Is *equal* to	`$j == 4`
!=	Is *not equal* to	`$j != 21`
>	Is *greater than*	`$j > 3`
<	Is *less than*	`$j < 100`
>=	Is *greater than or equal* to	`$j >= 15`
<=	Is *less than or equal* to	`$j <= 8`
<>	Is *not equal to* to	`$j <> 23`
===	Is *identical to* to	`$j === "987"`
!==	Is *not identical to* to	`$j !== "1.2e3"`

Basic Syntax

Logical Variables:

- you generally use a logical operator to combine the results of two of the comparison operators shown in the previous section.
- A logical operator can also be input to another logical operator.
- As a rule, if something has a TRUE or FALSE value, it can be input to a logical operator.
- A logical operator takes two true or false inputs and produces a true or false result.

Operator	Description	Example
&&	*And*	`$j == 3 && $k == 2`
and	Low-precedence *and*	`$j == 3 and $k == 2`
||	*Or*	`$j < 5 || $j > 10`
or	Low-precedence *or*	`$j < 5 or $j > 10`
!	*Not*	`! ($j == $k)`
xor	*Exclusive or*	`$j xor $k`

Basic Syntax

Variable Assignment

- The syntax to assign a value to a variable is always

  variable = value.

- Or, to reassign the value to another variable, it is

  other_variable = variable.

Variable incrementing and decrementing:

Adding or subtracting 1 is such a common operation that PHP provides special oper- ators for it. You can use one of the following in place of the += and -= operators:

```
++$x;
--$y;
if (++$x == 10) echo $x;
```

This tells PHP to first increment the value of $x and then to test whether it has the value 10 and, if it does, to output its value.

Basic Syntax

Variable incrementing and decrementing:

But you can also require PHP to increment (or, as in the following example, decrement) a variable after it has tested the value, like this:

```
if ($y-- == 0) echo $y;
```

String Types:

PHP supports two types of strings that are denoted by the type of quotation mark that you use. If you wish to assign a literal string, preserving the exact contents, you should use single quotation marks (apostrophes), like this:

```
$info = 'Preface variables with a $ like this: $variable’;
```

In this case, every character within the single-quoted string is assigned to $info.

If you had used double quotes, PHP would have attempted to evaluate $variable as a variable. On the other hand, when you want to include the value of a variable inside a string, you do so by using double-quoted strings:

```
echo "This week $count people have viewed your profile";
```

Basic Syntax

Escaping characters:

```
$text = 'My spelling's atroshus'; // Erroneous syntax
```

To correct this, you can add a backslash directly before the offending quotation mark to tell PHP to treat the character literally and not to interpret it:

```
$text = 'My spelling\'s still atroshus’;
```

Multiple line commands:

There are times when you need to output quite a lot of text from PHP, and using several echo (or print) statements would be time-consuming and messy.

To overcome this, PHP offers two conveniences.

The first is just to put multiple lines between quotes, as in the example

```
<?php
  $author = "Steve Ballmer";

  echo "Developers, developers, developers, developers, developers,
  developers, developers, developers, developers!

  - $author.";
?>
```

Basic Syntax

Variable Typing:

Note: PHP is a very loosely typed language. This means that variables do not have to be declared before they are used, and that PHP always converts variables to the type required by their context when they are accessed.

```
<?php
  $number = 12345 * 67890;
  echo substr($number, 3, 1);
?>
```

returning a result of 838102050,

substr can access it and return the character, which in this case is 1.

Basic Syntax

Variable Typing:

Automatic converting string to a number:

The same goes for turning a string into a number, and so on. In the example, the variable $pi is set to a string value, which is then automatically turned into a floatingpoint number in the third line by the equation for calculating a circle's area, which outputs the value 78.5398175.

```
<?php
  $pi     = "3.1415927";
  $radius = 5;
  echo $pi * ($radius * $radius);
?>
```

Basic Syntax

Constants:

- Constants are similar to variables, holding information to be accessed later, except that they are what they sound like—constant.
- In other words, once you have defined one, its value is set for the remainder of the program and cannot be altered.
- One example of a use for a constant is to hold the location of your server root (the folder with the main files of your website). You would define such a constant like this:

```
define("ROOT_LOCATION", "/usr/local/www/");
```

Basic Syntax

Predefined Constants:

- PHP comes ready-made with dozens of predefined constants However, there are a few—known as the magic constants—that you will find useful.

Magic constant	Description
`__LINE__`	The current line number of the file.
`__FILE__`	The full path and filename of the file. If used inside an `include`, the name of the included file is returned. Some operating systems allow aliases for directories, called *symbolic links*; in `__FILE__` these are always changed to the actual directories.
`__DIR__`	The directory of the file. (Added in PHP 5.3.0.) If used inside an `include`, the directory of the included file is returned. This is equivalent to *`dirname`*`(__FILE__)`. This directory name does not have a trailing slash unless it is the root directory.
`__FUNCTION__`	The function name. (Added in PHP 4.3.0.) As of PHP 5, returns the function name as it was declared (case-sensitive). In PHP 4, its value is always lowercase.
`__CLASS__`	The class name. (Added in PHP 4.3.0.) As of PHP 5, returns the class name as it was declared (case-sensitive). In PHP 4, its value is always lowercased.
`__METHOD__`	The class method name. (Added in PHP 5.0.0.) The method name is returned as it was declared (case-sensitive).
`__NAMESPACE__`	The name of the current namespace. (Added in PHP 5.3.0.) This constant is defined at compile time (case-sensitive).

Basic Syntax

Functions

- Functions separate sections of code that perform a particular task. For example, maybe you often need to look up a date and return it in a certain format.
- That would be a good example to turn into a function.
  - The code doing it might be only three lines long,
  - but if you have to paste it into your program a dozen times, you're making your program unnecessarily large and complex if you don't use a function

```
<?php
  function longdate($timestamp)
  {
    return date("l F jS Y", $timestamp);
  }
?>
```

Basic Syntax

Functions

- Functions separate sections of code that perform a particular task. For example, maybe you often need to look up a date and return it in a certain format.
- That would be a good example to turn into a function.
  - The code doing it might be only three lines long,
  - but if you have to paste it into your program a dozen times, you're making your program unnecessarily large and complex if you don't use a function

```php
<?php
  function longdate($timestamp)
  {
    return date("l F jS Y", $timestamp);
  }
?>
```

```php
echo longdate(time());
```

Basic Syntax

Variable Scope

- Local Variables:
  - Local variables are variables that are created within, and can be accessed only by, a function. They are generally temporary variables that are used to store partially processed results prior to the function's return.

```php
<?php
  function longdate($timestamp)
  {
    $temp = date("l F jS Y", $timestamp);
    return "The date is $temp";
  }
?>
```

Basic Syntax

Variable Scope

- Local Variables:
  - Now, to see the effects of variable scope, let's look at some similar code in the example. Here $temp has been created before we call the longdate function
  - However, because $temp was neither created within the longdate function nor passed to it as a parameter, longdate cannot access it. **Therefore, this code snippet outputs only the date, not the preceding text**

```
<?php
  $temp = "The date is ";
  echo longdate(time());

  function longdate($timestamp)
  {
    return $temp . date("l F jS Y", $timestamp);
  }
?>
```

Basic Syntax

Variable Scope

- Global Variables:
  - There are cases when you need a variable to have global scope, because you want all your code to be able to access it.
  - Also, some data may be large and complex, and you don't want to keep passing it as arguments to functions.

```php
<?php
$x = 75;
$y = 25;

function addition() {
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>
```

Basic Syntax

Variable Scope

- Super Global Variables:
  - Starting with PHP 4.1.0, several predefined variables are available.
  - These are known as super global variables,
    - which means that they are provided by the PHP environment
    - but are global within the program, accessible absolutely everywhere.

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

Basic Syntax

Variable Scope

- Super Global Variables:
  - Starting with PHP 4.1.0, several predefined variables are available.
  - These are known as super global variables,
    - which means that they are provided by the PHP environment
    - but are global within the program, accessible absolutely everywhere.

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

```
$came_from = $_SERVER['HTTP_REFERER'];
```