LEARN PHP IN A DAY

THE ULTIMATE CRASH COURSE TO LEARNING THE BASICS OF PHP IN NO TIME

ACODEMY

By Acodemy

© Copyright 2015

All rights reserved. No portion of this book may be reproduced – mechanically, electronically, or by any other means, including photocopying without the permission of the publisher

LEARN PHP IN A DAY

The Ultimate Crash Course to Learning the Basics of PHP in No Time

Disclaimer

The information provided in this book is designed to provide helpful information on the subjects discussed. The author's books are only meant to provide the reader with the basics knowledge of a certain topic, without any warranties regarding whether the student will, or will not, be able to incorporate and apply all the information provided. Although the writer will make his best effort share his insights, learning is a difficult task and each person needs a different timeframe to fully incorporate a new topic. This book, nor any of the author's books constitute a promise that the reader will learn a certain topic within a certain timeframe.

Contents

Preface

Chapter One – Introduction, Setup and “Hello World”

- Introduction to PHP
- Setting up Our Work Environment
- Our Very First PHP file

Chapter Two – Variables

- Introduction to Variables
  - Basics
  - Rules
- Variable Types and Typecasting
  - Boolean
  - Number
  - String
  - Array
  - Objects
  - NULL
- Conclusion

Chapter Three – Logical, Math and other Expressions and Operations

- Introduction to Expressions and Operators
  - Expressions
  - Operators
- Conclusion

Chapter Four – Control Structures

- Introduction
- If statements
  - If Statements
  - If Else Statements
  - If-elseif-else statement
- Switch
- Alternate syntax to control structures
- While

- Do-while
- For
- Foreach
- Break
- Continue
- Return
- Include
- Require
- Require_once
- Include_once
- Conclusion

Chapter Five – Functions
- Introduction
- User Defined Functions
  - Function Arguments
  - Return values
  - Variable Scope
- Pre-defined functions
  - Echo, Print, exit, die
  - String functions
- Conclusion

Chapter Six – Databases
- Introduction
- What is an API?
  - Connectors
  - Drivers
  - Extensions
  - PHP MySQL APIs
  - mysqli Extension
  - PDO Extension
- PHPMyAdmin and getting familiar with MySQL
  - Basics
- MySQLi
  - Dual interface
  - Connections

Executing statements
Prepared statements
Conclusion
Chapter Seven – Form Data
Introduction
Methods for sending Form Data
Referencing information from forms
Security
Conclusion
Chapter Eight – Sessions and Cookies
Introduction
Sessions
Passing the Session ID
Custom Session Handlers
Cookies
Creating and retrieving cookies with PHP
Modifying a cookie using PHP
Deleting a cookie using PHP
Check if Cookies are enabled using PHP
Conclusion
Chapter Nine – File Handling
Introduction
File Handling
Reading Files
Opening files
PHP Reading files
PHP Closing files
PHP Create File
PHP Write to File
PHP Overwriting
Conclusion
Chapter Ten – Object Oriented Programming
Introduction
Basics
Class

- new
- extends
- Properties
- Constants
- Autoloading Classes
- Constructors and destructors
- Object Inheritance
- Scope Resolution Operator (::)
- Conclusion
- Answers to Exercises
- Chapter 2
- Chapter 3
- Chapter 4
- Chapter 5
- Chapter 6
- Chapter 7
- Chapter 8
- Chapter 9
- Chapter 10

Preface

1. Introduction to the Course

Welcome to this Introductory Course to PHP. Throughout this course, you will learn the basics behind using PHP in your own projects. You will be introduced to some of the most commonly used functions and methods as well as PHP best practices. This course will mostly teach by example, emphasizing a “show, don’t tell” approach. We will guide you through different real-world examples of PHP applications and explain every step of the process so that you understand not only ***how***, but also ***why*** we are writing our application the way we are.

The course is structured around different chapters, where each chapter is focusing on a particular aspect of programming with PHP. At the end of each chapter, there will be a chapter summary and a chapter exercise that will test all the skills you have learned in the chapter thus far, as well as build on top of previous knowledge.

You will be presented with code at every step of the way. Code will be easily recognized as it will be separated from the rest of the text, and will be formatted differently. Here is an example of a code block:

```
if(!isset($user) || $user == '' || !isset($msg) || $msg == '') {
    $error = "Please fill in the form.";
    header('Location:
index.php?error='.urlencode($error));
    exit();
```

With all of that out of the way, let’s get started with the course!

Chapter One – Introduction, Setup and “Hello World”

Introduction to PHP

PHP stands for ***PHP: Hypertext Preprocessor*** (originally it meant ***Personal Home*** Page) and is one of the most widely used web programming languages (installed on more than 250 million websites). It is similar to languages such as C. But let’s not spend too much time in historical references and get to work!

Setting up Our Work Environment

In order to start working with PHP on your local computer, you need to install a development environment. Why is this so? PHP is a server-side language; so in essence, you need to install a "local server" on your computer in order to run PHP code. There are many solutions that provide PHP packages, but the most common form is known as a LAMP bundle. There are many LAMP bundles out there that fit different operating systems. The most common are:

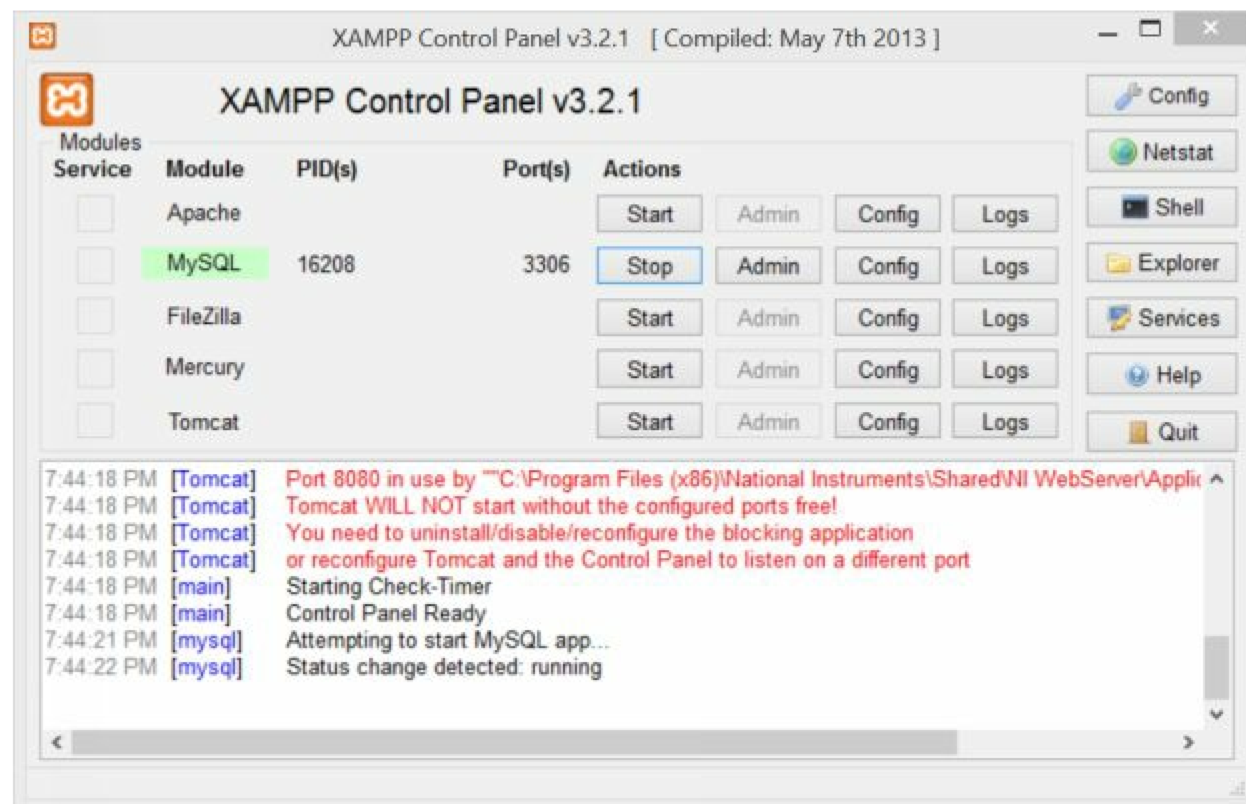
- XAMPP – an Apache distribution with PHP, MySQL and Perl. Support available for Windows, Linux and Mac;
- WAMP – specific for Windows;
- MAMP – specific for Mac;

Pick one, type the name in Google, follow the download link and instructions and you'll be all set to go.

Also, you will need some form of text editor. There are many free and paid programs you can find. The one we are using for this course is called *Notepad++*.

Our Very First PHP file

After having installed your LAMP bundle, you undoubtedly want to create your very first PHP file. If you haven't already, make sure that you have Apache and MySQL active. To check, go to the control panel of your LAMP setup and you should see something like this:



Navigate to the installation directory of your LAMP bundle. In my case it looks something like: " C:\xampp\htdocs ". " htdocs " is the directory that contains all of your websites. Generally, for the sake of clarity, I like creating a " websites " directory inside of " htdocs " and working from there. In either case, create a new folder for your website and give it a name. Try to refrain from using spaces (use _ instead). Inside that folder, create an " index.php " file. Open the file in your text editor and type the following lines of code:

```
<?php

echo 'Hello World!';

?>
```

All PHP code is wrapped within the PHP tags (<?php ?>). Notice that we say all PHP *code*, not all PHP *files*. That is, you can have things such as HTML inside a .php file. More on that in another chapter, though.

Some more rules to keep in mind in terms of syntax:

- In PHP every statement ends with a semicolon.
- Comments:

Comments in PHP can be single-line or multi-line. Single line comments start with two backslashes (//). Anything placed after these slashes is considered a comment and will not be executed by the compiler. Multi-line comments start with a forward slash and a star and end with a star and forward slash (/* Comment goes here*/). Again, the compiler will ignore anything placed inside of these symbols.

Here is an example of comments in code:

```
<?php
/*
This is an example of a multi-line comment;
*/
functio myFunc($arg1, $arg2) {
   returı $arg1*$arg2; //This is an example of a
single-line comment;
}
```

Multi-line comments are usually used for longer descriptions of blocks of code, while single-line comments are used (as in the example above), when you want to describe what a certain line of code does. Multi-line comments can be also useful for debugging. For example, if you have a portion of code that is not working the way you want it to, you can "comment it off" by using multi-line comments and tests bits of it to see what exactly goes wrong.

There is no rule in PHP that tells you whether to use comments or not to, but it is considered good practice to use comments. If you are going to share your code with other people, you should ***definitely*** (and by definitely, we mean always!) use comments. What seems obvious to you will not be so obvious to another coder. Furthermore, even if you are not going to be sharing your code, but are instead just writing some file that you know will be for yourself, you should still make use of comments (even if you don't comment as extensively as you would if you were sharing). Very often you will find yourself needing to revisit old code and when that happens you would want to quickly be able to remember what you were doing and why you were doing it. Although they don't add functionality to your code, comments are just as important as the code you write, so don't ignore them!

Okay, after that digression, let's return to our first PHP file. Open your web browser of choice and enter the following URL: " localhost/websites/your_folder_name_here/index.php " or remove the " websites " if your new website is in the " htdocs " folder.

You should see the words "Hello World!" being displayed at the top of the web page.

Hurray! Our very first PHP file is done! But that wasn't very interesting, was it? Let's move on to chapter two where we learn about the different types of variables inside PHP and start having some more fun!

Chapter Two – Variables

Introduction to Variables

Basics

In PHP, variables are declared using the dollar sign($):

```
<?php
$a = 'My First Variable!';
?>
```

This declares, or creates, the variable. Important to note is that before you create the variable, it is considered unset. It doesn't exist. Kind of intuitive, but this can sometimes lead to some unexpected errors as we are going to see later on.

Here we have created a string variable (note that we don't declare the type). We'll look at the different variable types a bit later in this chapter.

Rules

There are rules to keep in mind when creating variables inside of PHP. We'll list all of them and after that we will take a look at code that shows these. Here are the basic rules:

- Variable names start with a letter or underscore;
- Variable names cannot start with a number;
- A variable name can have any number of letters, underscores of numbers;
- Valid letters are all ASCII hexadecimal letters from 00 to FF;

Let's take a look at an example of code declaring variables:

```
<?php

$a = 'This var'; //Perfectly valid variable declaration;
$_var2 = 42;  // This is also valid;
$124 = 'foo' ; //Invalid;
$äbc = array(); //Valid variable declaration;
```

If we test our file we will get the following error message:

```
Parseerror syntax error, unexpected '124'
(T_LNUMBER), expecting variable (T_VARIABLE) or
'$' in {pathto filehere} on line 5
```

This tells us that we have an error in declaring our variable.

Variable Types and Typecasting

Unlike many other languages, PHP does not require you to explicitly declare the variable type, rather, it determines it from the context. In other words, you don't have to explicitly tell PHP whether you want a variable to be a double precision number or a string. To help you imagine this, you can think of variables as boxes. Imagine a row of lockers (like the kind you would see at an airport or a train station). Each locker represents a variable. Each locker has its own unique number, which differentiates it from all of the other lockers. If someone tells you to open locker number 42, you would be able to find it without a problem and you wouldn't confuse it with any other locker (for example 24). In a similar way, PHP variable names are unique and each PHP variable is unique. Now, you open locker number 42, and you see that it is empty. You move on to locker 43 and see that there is a package inside it. You take the package from 43 and place it in 42. Before opening the lockers you didn't know what was in the lockers. After checking, you know what is contained inside that locker and you can manipulate it as desired. In a similar fashion, variables in PHP can contain anything you place in them. Essentially, when you are checking the type of a variable, you are checking the type of the content inside of the variable.

That explanation may be a bit confusing, but you'll get used to it very quickly and you will not have problems dealing with variables in PHP. Let's take a look at some of the variables inside of PHP:

Boolean

Booleans are the simplest type of variables that exist in PHP. Booleans are used for logical statements and to express a truth value. Booleans are binary, meaning that they take one of two values: TRUE or FALSE . Having said that, let's look at an example of a Boolean variable:

```
<?php

$bool = TRUE; // Assigns the value of TRUE to the variable
'bool';
```

Like many other languages, PHP is flexible in its use and interpretation of Boolean variables. In other words, we can create Booleans without explicitly defining them as either being TRUE or FALSE. IN fact, here is the list of rules that determines which values are considered to be FALSE by PHP:

- The Boolean FALSE ;
- The integer 0 ;
- The float 0.0 ;
- The empty string and the string "0" ;
- An array with zero elements;
- The special type NULL ;

- A SimpleXML object created from empty tags;

Everything else will return true ;

This means that if we can write the following code and it will be valid:

```
<?php
if($var == 1) { // will return TRUE if the value of $var is
equal to 1;
   // execute some code
}
```

Even though we still haven't covered if statements, you can pretty much figure out what will happen here. The implicit Boolean variable is defined inside the parenthesis (). Once the logical operation executes it will return either a TRUE or a FALSE.

A couple of important things to note about the above example:

PHP considers the integer 1 to be the same as TRUE. That means, if we have defined the variable 'var' to be TRUE, and we check it against the integer one, the logical statement will return TRUE:

```
<?php
$var = TRUE;
if($var == 1) { // will return TRUE
   // execute some code
}
?>
```

The second thing to note:

We are using a double equals sign to check whether one value is equal to another. Don't make the mistake of using a single equal sign.

After all of that, we should introduce ***typecasting***. Typecasting is when we take the value a variable and we convert it from one type to another. For example, we can have a variable with a value of 42 and we want to cast it to a Boolean. Another example is if we have a variable with the value of 42 and we want to cast it to a string (more on that later). For Booleans, type casting is for the most part unnecessary (i.e. you will almost never ever do it). This is because values will automatically be converted to Booleans if an operator, function or control structure requires a Boolean as argument. However, for the sake of consistency with the other type casts, this is how to cast a variable to a Boolean:

```
<?php
$var = (bool)$var;
$var2 = (boolean)$var;
?>
```

Number

Numbers in PHP, and most programming languages, come in different flavors. The two subtypes of numbers in PHP are Integers and Floating point numbers. Let's take a look at each one.

Integers

By definition, integers are all numbers in the set:

$Z = \{..., -3, -2, -1, 0, 1, 2, 3...\}$

In other words, integers are all whole, real numbers (both positive, negative and zero). PHP allows us to define integers in a number of ways. You can define integers in base 10, base 8 (octal), base 16 (hexadecimal) or binary (base 2).In order to use octal notation for an integer, you need to precede the number with a `0` . To use hexadecimal notation for an integer, you precede the number with an `0x` . Lastly, if you want to use binary notation, you precede the number with `0b` . Here is an example:

```
<?php
$num1 = 42; // a decimal number
$num2 = -42; // a negative decimal number;
$num3 = 012; // octal number;
$num4 = 0x1B; // hexadecimal number;
$num5 = 0b1010; //binary number
var_dump($num1);
?>
```

Take note of the last function we use (don't worry if you don't know what functions are).

Helpful Tip!

You can check the type and contents of a variable by using `var_dump()`. *The function takes one input as an argument (again, don't worry if you don't fully understand. This means that you should only put one thing inside the parenthesis). The function will print the type of the variable and the contents to the screen. In the above example, the output to the screen will look like:*

```
int(10)
```

Now try to use `var_dump` on the other variables. Do you notice anything interesting? Do you see the value that is being printed to the screen for each variable? What base is it in?

The largest number that can be stored on a 32-bit system as an integer is of magnitude 21474783647. Anything larger than that will automatically be converted to a type of `float` . For a 64-bit system the number is of magnitude 9223372036854775807. Anything larger than that will automatically be converted to a type of `float` .

If you want to convert a variable to an integer, in other words cast it as an integer, you would do the following:

```
<?php
$num1 = (int)$num1; // casts the variable to an integer;
$num2 = (integer)$num2; // casts the variable to an integer;
?>
```

This type of casting is useful in a number of situations, but the most common usage is for security. One of the exercises at the end of the chapter will show you a particular usage of integer casting for security purposes.

Floating Point Numbers

Integers was the set of real and whole numbers, so logically, floating point numbers (also referred to as “floats”, “doubles” or “real numbers”) will include everything else. That is, the numbers that in between integers. Thus, we get the set of all real numbers. With that being said, it is crucial to note that all computers have a limitation as to the precision with which they display numbers (this is known as a ***machine epsilon***). Furthermore, there are some weird discrepancies that may occur when dealing with floats. We won’t go too much into that since it is beyond the scope of this book, but if you find any non-intuitive results when dealing with floats, think about the precision with which PHP handles floats.

Floats can be represented in the following ways in PHP:

```
<?php
$num1 = 42.42; // stores as the number 42.42;
$num2 = 42e3; // equivalent to 42*10^3 = 42000;
$num3 = 42E-3;  // equivalent to 42*10^-3 = 0.042;
// Try doing a var_dump() on these to see what you get;
?>
```

String

Stings in PHP are series of characters. You use these to store pieces of text and words. PHP handles each character as a byte. With that being said, we should note that the largest string that PHP can support is of size 2GB. The fun bits come when defining strings.

Single quotes

The easiest way to define a string in PHP is to use single quotes.

```
<?php
$str = 'This is a string';
?>
```

What happens if we want to incorporate new lines in our string?

```
<?php
echo $str = 'This is a
string';
?>
```

This is valid syntax inside of PHP, but the newline break will not be incorporated into the final output. Not the use of the `echo` function. What does the above code output?

Helpful Tip!
You can use the `echo` *command to output strings to the screen. In fact,* `echo` *can be used with any of the variable types to output their contents other than the* `array` *and* `object` *types. This is very useful tool for debugging and you should often use it when you need to check something quickly or to find errors in your code.*

OK, now imagine you want to input a string that contains a single quotation mark. What do you think the output is going to be?

```
<?php
$str = 'I can't output this';
?>
```

You can probably tell even from the syntax highlighting that this is not going to work. Every time PHP encounters a single quotation mark, it will either start or end a string variable. In order to store a single quotation mark, we need to escape it. In PHP, this is done using the backslash. The same thing will happen if we want to include a backslash. We have to escape it with another backslash:

```
<?php
$str = 'I can\'t output that.';
$str2 = 'This is a backslash \\';
?>
```

Use `echo` to see the results above. Keep in mind that a backslash is used for escaping.(This means that `\n` will not render as a newline)

Another thing that cannot be used inside single quotation marks is variables. For examples, the following will not expand inside of single quotation marks.

```
<?php
$str = 'Katherine';
$str2 = 'Her name is $str';
?>
```

This will not display “Her name is Katherine” as you would expect.

In order to make the above example work, we need to ***concatenate*** the variable and the string. In PHP we would do the following:

```
<?php
$str = 'Katherine';
$str1 = 'Her name is '.$str;
?>
```

Test out the above code and confirm that it works.If you want to continue the string after the variable, you just concatenate the next string to the variable. In other words you use $str.'your string here' . This works as long as the variable can be cast into a string (PHP does that automatically for you in this case).

Double quotes

This is the more complex way of defining strings in PHP, but it gives you a lot more control and flexibility. In the previous examples, we saw that \n, \t, \r and similar will not work in single quotes. This is not the case with double quotes. Double quotes allow you to use those special symbols.

The most useful thing about double quotes, however, is their ability to parse variables. There are two ways to parse variables inside of double quotation marks.

Simple variable parsing

The simple way to parse variables is by just using the dollar sign inside of your string:

```
<?php
$color = 'red';
echc $str = "Mark just bought a new $colo car.";
?>
This will output:
Mark just bought a new red car.
```

You can use this same concept if you want to reference elements of an array. (You should probably come back to this section after reading about the next variable type).

```
<?php
$people = array("Anna", "Jill", "Dom");
$subjects = array("engineering" , "art", "math");
echc $str1 = "$peopl[0]'s favorite subject is $subject[0]. ";
echc $str1 = "$peopl[1]'s favorite subject is $subject[1]. ";
echc $str1 = "$peopl[2]'s favorite subject is $subject[2]. ";
?>
This will output:
Anna's favorite subject is engineering. Jill's favorite subject is
art. Dom's favorite subject is math.
```

Complex variable parsing

This is called complex because it allows you to create more complex expressions, not because the

method itself is complex. This method requires you to use curly braces around your variables.

```php
<?php
$str = 'Katherine';
$str1 = "Her name is {$str}";
?>
```

You can expand this and test it out on things such as arrays, objects and functions once you learn about those.

String casting

As with all other variables, we can cast variables to strings. To do this you use the (string) function. PHP will automatically convert variables to strings depending on the scope of the expression and if a string is required. For example, if you use echo on a variable, it will first convert it to a string.Numbers will be converted to their textual representation. This means that 10 will become "10" . Arrays and objects behave a bit differently, though. If you use echo on an array , you will get the string "Array" . Similarly, if you use echo on an object, you will get the string "Object" .

Array

An array is a type of ordered map. What this means is that arrays will associate values to specific keys. Arrays in PHP can be used to represent dictionaries, vectors, lists, stacks, queues, trees and multidimensional arrays (arrays of arrays).

Arrays are created in the following way:

```php
<?php
$arr = array(
    "object"    => "keys",
    "fruit"    => "apples",
    "food"     => "candy",
);
?>
```

Note how we have used indentation for the sake of clarity. The extra whitespace is ignored by PHP, but it makes our code easier to read and manage.

Each line of the array represents a key-value pair. Each line ends with a comma to indicate the next key-value pair in the array. The last entry of the array does not need to have a comma at the end of it, but most people like to include it. This convention is adopted by many people dealing with content management systems such as WordPress, Drupal and others. (The idea of leaving the comma is that you can take the entire line of key-value pairs and move it around in the array without having to worry about including a comma).

The keys of an array can be either integers or strings. If you use something else for the array key, it will automatically be cast into either of these types. (NOTE: arrays and objects cannot be used as array keys as this will result in an error).

Alternatively, you do not need always need to supply the key. You can do the following:

```
<?php
$arr = array('PHP', 'HTML', 'CSS', 'MySQL');
?>
```

This will create keys for you automatically by incrementing each key, starting from 0. We can check the contents of the array:

```
<?php
$arr = array('PHP', 'HTML', 'CSS', 'MySQL');
var_dump($arr);
?>
This outputs:
array(4) { [0]=> string(3) "PHP" [1]=> string(4) "HTML" [2]=>
string(3) "CSS" [3]=> string(5) "MySQL" }
```

You can see that the first entry of the array starts from key 0. It is important that you remember that the first value in an array with default keys is always going to be stored under key 0.

You can reference elements of an array using the square brackets.

```
<?php
$arr = array('PHP', 'HTML', 'CSS', 'MySQL');
echo $arr[2];
?>
This outputs: CSS
```

Play around with this syntax and try to create arrays of arrays (multidimensional arrays). In other words, try to have an array as the value of some key. This looks like:

```
<?php
$menu = array(
    'salads'  => array('green' , 'fruit' ),
    'pizzas' => array(
        '1 topping'      => array('cheese' , 'sausage' ),
        '2 toppings'     => array('cheese  and sausage' ),
    ),
);
?>
```

You will get a chance to practice this in one of the exercises.

If you want to create new elements to an existing array, or to modify elements of an existing array you use the square bracket syntax:

```
<?php
$arr = array('pizza', 'olives' , 'mushrooms' ); // creates the
initial array;
$arr[3] = 'cheeseburger' ; // adds a fourth element with the
key of 3 and the value of cheeseburger;
$arr[0] = 'apples' ; // modifies the first element to the value of
apples;
$arr[] = 'candy' ; // adds a new element to the array and the
key will be incremented automatically;
unse($arr[2]); // unsets (deletes) the 3rd key => value pair of
the array;
```

The above outputs:

```
array(4) {
  [0]=>
  string(6) "apples"
  [1]=>
  string(6) "olives"
  [3]=>
  string(12) "cheeseburger"
  [4]=>
  string(5) "candy"
}
```

Objects

Objects will be an object of discussion in Chapter 10 and we shall leave them for now.

NULL

The NULL value is special and it represents a variable with no value. The only possible way of that happening is if the variable is assigned the constant NULL, the variable has not been assigned a value yet, or if the variable has been unset (remember the last example from the array section when we unset the value of the 2nd entry).

Conclusion

This is the end of this chapter! You learned a lot about PHP and how it works. You've taken your first steps, steps that you will use over and over again when writing code. This chapter may have been a little boring in some bits, but it is important you learn these basics well so you can apply them without having to think about it later on. Here are some exercises to get your gears going and get you thinking about using what you've learned.

Exercise 1

You are given the following array of user data. Perform the specified manipulations to it:

```
<?php
$user_data = array(
    'username'  => 'test_user',
    'name'      => array(
        'first' => 'John',
        'last'  => 'Doe',
    ),
    'admin'     => 0,
    'user_id'   => '42',

);
?>
```

1. Create a string that consists of the first and last name of the user by concatenating the values from the array.
2. Suppose that this array will be stored into a database of other users. We want to make sure that all values will be of the correct type.
    1. Cast the user ID to a type of integer;
    2. Cast the admin field to a type of Boolean;
3. Add a new field with a key of 'email' and set it to the value of: ' john.doe@example.com '.

Exercise 2

Create a menu for a restaurant. To do this, you have to use a multidimensional array. Your menu has to include three main subcategories (name your array menu and the keys inside this array will be your main subcategories). Each subcategory will include 4 meals/foods. For each meal/food you need to include a list of ingredients and a list of nutritional information (carbs, sugars, protein, etc.). You have to figure out the most efficient way to do this inside of PHP. (*Hint: Think about how you want to create and organize the information for each meal. Do you want to make the key the name of the meal, or do you want to set the key to be automatically incremented and include the name of the meal inside the array? Which makes the most sense to you? Which causes the least confusion? Sketch it out on paper if need be*).

Exercise 3

Imagine you are creating an online application for a restaurant and you are tasked with creating the section for deliveries. Use the array from the previous exercise to provide yourself with the data for the menu. Create a variable that represents one of the three main subcategories (it should have the same exact name). Now create a variable that corresponds to one of the foods you made. Suppose that these variables represent the choice made by a user when selecting their meal.

Create a string that would be a message displayed to the user with important information about their order. It should look like this:

```
You picked: {name of meal here}.
This meal is made from: {list ingredients here}.
The nutritional information for this meal is: {list nutritional information here}.
```

Think about concatenation, single vs. double quotations, array referencing.

Chapter Three – Logical, Math and other Expressions and Operations

Introduction to Expressions and Operators

Expressions

PHP revolves around expressions. Expressions are the basic building blocks of the language and pretty much anything you write is considered an expression. A more formal definition of an expression is “anything that has a value”.

In their most basic forms, expressions are constants and variables.So for example, if you want to write `$var = 42` , you are essentially assigning the value of `42` into `$var` .If in some later part of your code you write `$var2 = $var` , you expect the value of `$var2` to be the same as the value of `$var` , `42` .Taking this one step further, functions (which we haven’t covered yet), are also considered expressions. A function is an expression with the value of whatever value that function returns. (Even if you are not familiar with functions inside of PHP, you should have some experience with mathematical functions. You can imagine all of this in terms of `y=f(x)` and you will get an idea of what we are talking about.)

It’s easy to confuse operators with expressions in PHP, but to give you a general comparison between the two you can think of them like this. Operators are used inside of expressions. There are comparison expressions and comparison operators. The operators are the symbol that you use (i.e. >, <, ==), while the expression is what the comparison evaluates to (i.e. whether it evaluates to TRUE or FALSE).

We won’t go into more details about expressions here, as we want to keep this book more about real life applications instead of theory.

Operators

The technical definition of an operator in PHP is “something that takes one or more values (or expressions, in programming jargon) and yields another value (so that the construction itself becomes an expression)”.

Grouping of operators happens according to the number of values that the operator acts upon. There are unary operators, such as the logical NOT operator or the increments operators (`!`, `++`, `--` respectively). Binary operators take two values. These are the familiar math operators that we all know (`+`, `-`, etc.). There is also a single ternary operator that we will go into more detail when discussing logical operators.

Let’s go into analyzing the types of operators.

Operator Precedence

Operator precedence is used to determine how tightly an operator binds two expressions to one another. Operator precedence is used in cases when you want to determine the result of an expression such as `1+2*4` where parentheses are not used to force precedence.

When operators have equal precedence, their associativity decides how the operators are to be grouped. For example, if we write `1-2-4` this will give us `-5` because the minus sign (`-`) is left-associative.

Like in regular everyday math, you can use parentheses in PHP to force the precedence of an operator. It is generally advised to use parentheses even if you have written your expression in a way so you don't need them. Using parenthesis is a best practice that makes code more human readable and easier to understand.

Here is a comprehensive list of all operator precedence:

<table>
<tr><th>Associativity</th><th>Operators</th><th>More Information</th></tr>
<tr><td>Non-associative</td><td>clone new</td><td>Used to create a clone of an object or a new instance of an object respectively</td></tr>
<tr><td>Left</td><td>[</td><td>Used when reading/writing to elements of an array.</td></tr>
<tr><td>Right</td><td>**</td><td>Used for exponentiation in PHP</td></tr>
<tr><td>Right</td><td>++ -- (int) (float) (string) (array) (object) (bool) @</td><td>These are the increment and decrement operators and the type casting operators and the error control operator (more info about these later)</td></tr>
<tr><td>Right</td><td>!</td><td>Logical negation operator</td></tr>
<tr><td>Left</td><td>* / %</td><td>Arithmetic operators for multiplication, division and modulo (remainder).</td></tr>
<tr><td>Left</td><td>+ - .</td><td>Arithmetic operators for addition and subtraction as well as string concatenation.</td></tr>
<tr><td>Non-associative</td><td>== != === !== < ></td><td>Comparison operators</td></tr>
<tr><td>Left</td><td>&&</td><td>Logical AND operator</td></tr>
<tr><td>Left</td><td>||</td><td>Logical OR operator</td></tr>
<tr><td>Left</td><td>? :</td><td>Ternary operator</td></tr>
<tr><td>Right</td><td>= += -= *= **= /= .= %= &= |= ^= <<= >>=</td><td>Assignment operators</td></tr>
<tr><td>Left</td><td>and</td><td>Logical operator</td></tr>
<tr><td>Left</td><td>or</td><td>Logical operator</td></tr>
</table>

With that we close our discussion of operator precedence.

Arithmetic Operators

Arithmetic operators work just like the basic arithmetic operators in regular math. Let's begin by taking a look at a list of all the arithmetic operators and then we will look at a few examples.

Example	Name	Result
-$var	Negation	Opposite (negative) of the variable.
$var + $var2	Addition	The sum of the two variables.
$var - $var2	Subtraction	The difference of the two variables.
$var * $var2	Multiplication	The product of the two variables.
$var / $var2	Division	The quotient of the two variables
$var % $var2	Modulus	The remainder the first variable divided by the second.
$var ** $var2	Exponentiation	The result obtained by raising the first variable to the power corresponding to the second variable.

When using the division operator, it is good to keep in mind that the result will be a float unless the two operands are integers that are evenly divisible. If the two operands are integers that are not evenly divisible, the returned value will be of type float.

The operands of the modulus operator are converted to integers before processing. The decimal part of the value is stripped and the remaining integer is used.

The result of the modulus operator takes its sign from the dividend.

Let's take a look at a couple of examples.

```php
<?php

echo (4 % 3); // prints the value 1 ;
echo (4 % -3); // prints the value 1 ;
echo (-4% 3); // prints the value -1 ;
echo (-4% -3); // prints the value -1 ;
$a = 3;
$b = 2;

echo $a + $b; // prints out the value of $a + $b, which is 5;
echo $a - $b; // prints out the value of $a - $b, which is 1;
echo $a * $b; // prints out the value of $a * $b, which is 6;
echo $a / $b; // prints out the value of $a / $b, which is 1.5 of
type float;
echo $a ** $b; // prints out the value of $a ** $b, which is 9 ;

?>
```

Assignment Operators

The most basic assignment operator is " = ". You might be tempted to think of this as an "equals to", but that would be a mistake. This operator actually means that the left operand gets the value of the expression on the right. So essentially you can think of this as "gets sets to". With this knowledge, we can do some tricky and interesting things.

```php
<?php

$var = ($var2 = 2) + 3; // $var is equal to 5 now, and $var2
has been set to 2.
```

When dealing with arrays, assigning a value to a named key is done using the " => " operator. The precedence of this operator is the same as the precedence of all assignment operators.

In addition to the basic assignment operator, there are a number of "combined operators" that come in handy when we want to use shorthand for binary arithmetic operations. Let's say we have a variable $a equal to some value. In a later part of the code we want to set $a to be the previous value of $a plus some other value. To do that, we can write:

```php
<?php
$a = 42; // assign 42 to $a ;
$a = $a + 3; // new value of $a is equal to the old value of $a +
3;
```

A simpler way of doing the above would be to use the combined operator for addition:

```
<?php
$a = 42; // assign 42 to $a ;
$a += 3; // new value of $a is equal to the old value of $a + 3;
?>
```

These combined operators work for all arithmetic operations as well as for string concatenation. Here are a few examples:

```
<?php
$a = 42; // assign 42 to $a ;
$a += 3; // new value of $a is equal to the old value of $a +
3;
$a -= 3 // new value of $a is $a = $a - 3. So $a is 42 again;
$b = "Hello "; // create a string with value "Hello " (note the
space at the end) ;
$b .= "World!"; // concatenate the string "World!" to the
previous value of $b ;
$a /= 6; // new value of $a is $a = $a/6 = 7 ; (note that $a will
be an integer this time since the values divide evenly;
?>
```

Comparison Operators

Comparison operators allow us to compare two values. Let's take a look at a list of comparison operators.

Example	Name	Result
$var == $var2	Equal	Returns **TRUE** if the two variables are equal. Type juggling will be implemented if necessary.
$var === $var2	Identical	Returns **TRUE** if the two variables are equal and of the same type.
$var != $var2	Not equal	Returns **TRUE** if the two variables are not equal after type juggling.
$var <> $var2	Not equal	Returns **TRUE** if the two variables are not equal after type juggling.
$var !== $var2	Not identical	Returns **TRUE** if the two variables are not equal or they are not of the same type.

$var < $var2	Less than	Returns **TRUE** if the first variable is strictly less than the second.
$var > $var2	Greater than	Returns **TRUE** if the first variable is strictly greater than the second.
$var <= $var2	Less than or equal to	Returns **TRUE** if the first variable is less than or equal to the second.
$var >= $var2	Greater than or equal to	Returns **TRUE** if the first variable is greater than or equal to the second.

If you are comparing a string with a number or the comparison is between numerical strings, the strings are converted to numbers and the comparison is performed numerically.The type conversion is not done when using === or !== as these operators check for type as well as value.

```
<?php
var_dump(0 == "abc"); // 0 == 0 -> true ;
var_dump("2" == "02"); // 2 == 2 -> true ;
var_dump("20" == "2e1"); // 20 == 20 -> true ;
var_dump(100 == "1e2"); // 100 == 100 -> true ;
var_dump(100 === "1e2"); // 100 === "100" -> false ;
?>
```

Another conditional operator to be aware of is the ternary operator. The ternary operator is essentially a short hand for the if/else control structure (more on that later). The ternary operator has the form (some conditional expression) ? {do this if true} : {do this if false} . (The curly brackets are not used, they just there for clarity). Here is an example of a ternary operator as you would use it in an application and the corresponding if/else statement that it represents:

```
<?php

$welcome_msg = ($user_data['admin'] === 0) ? 'Hello
User!' : 'Hello Admin!';
/* Suppose we have an array of user data called
$user_data. In that
   array we have a field named 'admin' that takes the
values 0 or 1.
   We want to create a welcome message that would
display somewhere
   on our site. If the user is not an admin (value is 0), then
we simply
   say "Hello User", otherwise we say "Hello Admin".
*/
//This is the corresponding if/else statement:

if ($user_data['admin'] === 0) {
   $welcome_msg = 'Hello User!';
} else {
```

You can see that using the ternary operator is a lot shorter than writing out the entire if/else statement.

This concludes our overview of comparison operators. We have not extensively covered the comparison between variables of different types, but such information can be found in the PHP documentation available online so we have decided not to include it.

Error Control Operators

The error control operator is something that should be used cautiously. In PHP, the error control operator is the at sign (@). When an expression is prepended with the error control operator, any error messages associated with that expression that would normally be generated will be ignored.

Using this operator has it’s downsides because if you prepend it to an expression that generates an error that causes script termination, the script will stop without telling you why, which is bad for debugging.

We will not go into examples of using the error control operator, as you would rarely have to use it and should try to form your code so that you do not rely on it.

Incrementing and Decrementing Operators

The incrementing and decrementing operators are the ones you would see in C-style languages. These operators affect numbers and strings but not arrays and objects. Decrementing NULL values has no effect, but incrementing a NULL value results in 1.

Here is a list of the incrementing and decrementing operators.

Example	Name	Result
++$var	Pre-increment	Increments the variable by

		one, then returns the value.
$var++	Post-increment	Returns the value of the variable, then increments the variable by one.
--$var	Pre-decrement	Decrements the variable by one, then returns the value.
$var--	Post-decrement	Returns the value of the variable, then decrements the variable by one.

Let’s look at a few examples of how these work:

```
<?php
// Test to show you a simple example of post increment
$a = 42;
echo 'Value is 42: '. $a++.'<br>';
echo 'Value is 43: '. $a.'<br>';
// Test to show you a simple example of pre increment
$a = 42;
echo 'Value is 43: '.++$a.'<br>';
echo 'Value is 43: '.$a.'<br>';
// Test to show you a simple example of post decrement
$a = 42;
echo 'Value is 42: '.$a--.'<br>';
echo 'Value is 41: '.$a.'<br>';
// Test to show you a simple example of pre decrement
$a = 42;
echo 'Value is 41: '.--$a.'<br>';
echo 'Value is 41: '.$a.'<br>';
?>
```

We can also increment characters. We cannot decrement characters.

```
<?php

$a = 'a';
echo ++$a; // Outputs 'b';
echo --$a; // Outputs 'b'. Decrementing characters does not
work;

?
```

Logical Operators

PHP supports logical operators that allow us to evaluate logical statements. Here is a list of logical operators

Example	Name	Result
$a and $b	And	Returns **TRUE** if both variables are true.
$a or $b	Or	Returns **TRUE** if either one of the two variables is true.
$a xor $b	Xor (either)	Returns **TRUE** if either one of the variables is true, but not both.
!$a	Not	Returns **TRUE** if he variable is not **TRUE**.
$a && $b	And	Returns **TRUE** if both variables are true.
$a || $b	Or	Returns **TRUE** if either one of the two variables is true.

There are two logical operators for "AND" and for "OR" because they have a different order of precedence. The most common usage of logical operators is within an if/else statement or within the first expression of a ternary operator. Let's look at some examples of logical operators and how they are used.

```
<?php

// The result of the expression (false || true) is assigned to $e
// Acts like: ($e = (false || true))
$e = false || true;

// The constant false is assigned to $f and then true is ignored
// Acts like: (($f = false) or true)
$f = false or true;

var_dump($e, $f); // $e = TRUE, $f = FALSE
?>
```

String Operators

There are only two types of string operators and we have already seen both of them. The first operator is the dot operator which concatenates two strings. The second operator is the combined

concatenating assignment operator which appends the argument on the right side to the argument on the left side. You will use both very frequently.

```
<?php
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"

$a = "Hello ";
$a .= "World!";     // now $a contains "Hello World!"
?>
```

Conclusion

This concludes our overview of the different types of operators and expressions inside of PHP. You are getting closer and closer to writing your first real PHP applications! You need two more basic building blocks, covered in the next two chapters and you will be on your way to coding intermediate and advanced applications!

Exercise 1

Arithmetic-assignment operators perform an arithmetic operation on the variable at the same time as assigning a new value. For this PHP exercise, write a script to reproduce the output below. Manipulate only one variable using no simple arithmetic operators to produce the values given in the statements.

Hint: In the script each statement ends with "Value is now $variable."

```
Value is now 8.
Add 2. Value is now 10.
Subtract 4. Value is now 6.
Multiply by 5. Value is now 30.
Divide by 3. Value is now 10.
Increment value by one. Value is now 11.
Decrement value by one. Value is now 10.
```

Exercise 2

For this PHP exercise, write a script using the following variable:

```
$around="around";
```

Single quotes and double quotes don't work the same way in PHP. Using single quotes (' ') and the concatenation operator, echo the following to the browser, using the variable you created:

What goes around comes around.

Exercise 3

PHP allows several different types of variables. For this PHP exercise, you will create one variable and assign it different values, then test its type for each value.

Write a script using one variable “$whatsit” to print the following to the browser. Your echo statements may include no words except “Value is”. In other words, use the function that will output the variable type to get the requested text. Use simple HTML to print each statement on its own line and add a relevant title to your page. Include line breaks in your code to produce clean, readable HTML.

```
Value is string.
Value is double.
Value is boolean.
Value is integer.
Value is NULL.
```

Chapter Four – Control Structures

Introduction

PHP scripts are built out of a series of statements. A statement can be an assignment, a function call, a loop, a conditional statement or even empty statements. Statements can be grouped into statement groups by encapsulating them with curly braces. A statement group is considered a statement itself.

In this chapter we will observe the behavior of different types of statements.

If statements

If Statements

The “if” statement is one of the most important structures inside any language.This control structure allows you to have conditional execution of code fragments. PHP’s “if” structure is similar to that of C. That is, each if statement at its most basic looks like `if (expression) statement` . The expression is evaluated to its Boolean value. If the expression is TRUE, PHP will execute the statement. If it is FALSE, it will not execute it. A simple example is:

```
<?php
$a = 2;
$b = 1;

if($a > $b)
   echo $a.' is greater than '.$b;

?>
```

Very often you would need to execute more than one statement if a certain condition is true. You don’t need to wrap each statement in its own if clause. Instead, you can use curly braces to wrap the code you want to be executed conditionally.

```
<?php
$a = 2;
$b = 1;

if($a > $b) {
   echo $a.' is greater than '.$b;
   $b++;
}
?>
```

You can even take it one step further and nest if statements inside of other if statements.

If Else Statements

In some cases you want to execute a piece of code when a condition is met and a different piece of code if that condition is not met. In that case, you would have to use and `if-else` statement. `else` extends an if statement to execute an expression if the statement evaluates to FALSE. Here is an example that extends the one above:

```php
<?php
$a = 2;
$b = 1;

if($a > $b) {
   echo $a.' is greater than '.$b;
} else {
   echo $a.' is smaller than '.$b;
}
?>
```

If-else if-else statement

This is yet another modification to the if statement and it presents the most generalized form of an if statement. It includes and if statement that executes a block of code if the statement if TRUE. After this if statement, you have a number of elseif statements. Each else if statement has a condition associated with it and a block of code to be executed if true. The block of code is executed only if the condition evaluates to true. After all of the elseif statements there is a closing else statement. You can think of this as a kind of default case. If none of the conditions evaluate to true, the else statement will be executed. Here is an example:

```php
<?php
$a = 2;
$b = 1;

if($a > $b) {
   echo $a.' is greater than '.$b;
} elseif ($a == $b) {
   echo $a.' is equal to '.$b;
} else {
   echo $a.' is smaller than '.$b;
}
?>
```

Keep in mind that as soon as a condition is evaluated to TRUE, the script executes the associated block of code and breaks out of the if-elseif-else structure and will not go down to any of the other conditions. That is, only one condition will be executed when using an if-elseif-else.A structure that is related to the if-elseif-else structure is the switch statement which is the subject of discussion next.

Switch

The switch statement is similar to a bunch of if statements operating on the same variable. Imagine you have some variable that can take some values. In your code, you want to compare if that variable corresponds to some specific values. For each value, there is a code segment that has to be executed. Also, if the variable does not meet any of those cases, you want to execute some default piece of code. You could do this using a long if-elseif-else structure, but you could also do it using a switch statement.

A switch statement does the same thing as an if-else-if structure, but saves you some typing. The syntax for a switch statement is:

```
switch ($variable) {
     case {value you want to compare against}:
          {code here}
          break;
     case {different value}:
          {another code here};
          break;
     default:
          {default code}
          break;
}
```

You can have as many cases as you need for your code. An important thing to keep in mind is the presence of the break statement. The break command breaks you out of the current structure or loop you are in. If you did not have the break command inside each case, PHP would go through each case. That is, PHP checks the variable against each case and executes the specified code within if the comparison returns TRUE. If PHP does not encounter a break command, it continues to the next case and executes the specific code if the comparison of that case returns TRUE.This is one of the differences between using an if-elseif-else structure and a switch statement.

Let's take a look at an example comparison between the if-elseif-else structure and the switch statement:

```
<?php
i1 ($i == 0) {
    echc "i equals 0";
} elsei ($i == 1) {
    echc "i equals 1";
} elsei ($i == 2) {
    echc "i equals 2";
}

switcl ($i) {
    case 0:
        echc "i equals 0";
        breal;
    case 1:
        echc "i equals 1";
        breal;
    case 2:
        echc "i equals 2";
        breal;
}
?>
```

That’s pretty much all there is to switch statements. Learn how to utilize them and you will make your life faster than having to write out all of the if-else statements.

Alternate syntax to control structures

Before going on to each of the other control structures, let's take a moment to go to over the alternate syntax that PHP offers us for control structures. PHP offers us an alternative syntax for some of the control structures. These are: if , while , for , foreach and swtich . In each case, the basic form of the alternate syntax is to change the opening brace to a colon and the closing brace to an endif; , endwhile; , endfor; , endforeach; , or endswitch; , respectively (basically end with the name of the control structure appended.). What this allows us to do is break out of PHP in order to include for HTML, for example.

We haven't really talked much about including HTML inside of your PHP scripts, but it is actually a really easy concept. One way to include the HTML is to save the HTML string to a variable and then echo or print that variable. In fact, this is a method that you are going to find yourself using quite often. This method, however, has its shortcomings. For example, imagine you want to create a table that is not dynamically generated (that is, you are not using PHP to generate the table rows for you). The table has a specific layout and you want to include the value of some PHP variable inside one of the cells for example. If you try to write that inside a variable, you will find that it is very tedious and not efficient. (Unless for some reason you need to store that HTML in a variable, in that case you are stuck using the HTML to variable method). You can break out of PHP and then break back into PHP and include the necessary HTML inside the break. Here is a simple example:

```
<?php
$name = "Desi"; // variable that stores a user's name ;
?> <!--- > Here we break out of PHP and break into HTML <!--
->
<h2>Welcome<?php echc $name;?></h2> <!---> We
create a heading 2 tag that will display a welcome message
to our site. In order to make it personalized, we break into
PHP and echo the user's name ; <!--->
<?php // we break back into PHP
includ 'sidebar.php' ; // we continue witt our script. In this
case we include some PHP file that includes other code.
```

OK, after that digression, let's see the alternate syntax for an if statement:

```
<?php if ($a == 5): ?>
A is equal to 5
<?php endi; ?>
```

This looks exactly like the code above and will work exactly as you would expect it to.

Helpful Tip!

Whenever you are creating control structures that have to be encapsulated inside of curly braces,

always write out the entire control structure skeleton first before going in to fill it up. That is, if you are creating an if statement, you would write out:

```
<?php
if() {

}
?>
```

Always make sure you do this! If you do this, you will never have to keep track of opening and closing brackets and/or parentheses. In fact, you should make it a rule to yourself. **Whenever you open a bracket or parenthesis, close it before you write ANY other code.** *Then go back and fill in the code you need. This will keep you from making stupid errors and spending precious time debugging code while you could have been writing an amazing application.*

Another way to write the alternate syntax is to keep the curly braces and break out of PHP like normal. The following two statements are equivalent:

```
<?php if ($a == 5): ?>
A is equal to 5
<?php endif; ?>

<?php
if($a == 5) { ?>
A is equal to 5
<?
}
?>
```

That does it for the alternate syntax for control structures. You will often find yourself these alternate syntaxes in your code when you are juggling between PHP and HTML. In the next subchapter we will begin out discussion of the different types of looping structures in PHP. Keep in mind that the alternate syntax we discussed in this structure is applicable to some of the next structures as well.

While

This is the first type of loop that we will be talking about and it is also the simplest. The form of a while statement is:

```
while (expression)
      statement
```

The meaning of the while loop is also simple to understand. While loops tell PHP to execute the nested statement (does not need to be a single statement, it could be a block of code and in most cases it will be) as long as the while expression evaluates to TRUE. The expression is evaluated once at the beginning of the loop so even if the value of the expression is to change during the execution of the loop, the execution of the loop does not terminate (unless told otherwise) until the beginning of the next iteration. It is important to note that if the expression is FALSE the first time PHP checks it, the statement will not be executed.

As we mentioned, the statement could be a number of statements wrapped inside of curly braces. As with the if statement, we can use the alternate syntax to form our while loop. This is the general form of a while loop using alternate syntax for control structures:

```
while (expr):
   statement
   ...
endwhile;
```

Let’s run a simple example of this loop to count from 1 to 10.

```
<?php
$i = 1; // Initialize a counter variable ;
while ($i <= 10) { // start the while loop ;
   echo $i++;  /* the printed value would be
               $i before the increment
               (post-increment)  */
}

// Same thing but with the alternate syntax
$i = 1;
while ($i <= 10):
   echo $i++;
endwhile;
?>
```

While loops are very commonly used, especially when you are extracting information from a database

(we'll look at that in a later chapter).

One potential pitfall to lookout for is creating infinite loops, that is, loops that will never terminate (the while condition will always evaluate to TRUE). There are cases in which you would want to create an infinite loop and terminate it on a certain condition, but other times this behavior is not desired. Imagine the following examples.

You want to set up an application that will check the status of something and send an email to a user. Say, you are monitoring the status of a server and you want to know if something is going wrong. You want a script that will be running indefinitely and checking the server status at defined intervals. This can be achieved using an infinite loop that pauses for some defined amount of time before it resumes operation. Another example is if you are setting up a simple game using PHP to test your skills. You want to set up a main loop (you can refer to it as a "game loop" if you wish) that will run once each frame and do the operations that are associated with the game. Another example is if you are waiting for user-defined input. In that case you would set up an infinite loop that takes the user input and operates on it.

At the start infinite loops can be tricky and can cause you some problems, so try to avoid them until you are comfortable with the concept of looping and know what you are doing.

The easiest way to set up an infinite loop is:

```
<?php
whil (1) {
   ech 'INFINITE LOOPING!;
}
?>
```

The while condition will always evaluate to TRUE (in essence, the loop reads: while TRUE, execute {statement}).

Do-while

The do-while loop is similar to the loop we saw in the previous subchapter, with a minor difference. Do-while loops have the following structure:

```
do {
      statement
} while (condition)
```

This means that PHP will execute everything that is inside the do clause first, and then it will check the condition. If it evaluates to TRUE, PHP will loop back. What you will notice is that you evaluate the condition at the end of the loop instead of the beginning. This means that the code inside the do statement is guaranteed to run at least once. For example:

```
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```

This code will execute exactly one time before terminating because the truth expression evaluates to FALSE.

The only major difference between a do-while and a while loop is that the do-while will run the code at least once, whereas the while loop will not. With that being said, it should be noted that the while loop is the loop that you will be using most commonly in practice, but don't forget that the do-while loop exists!

For

For loops are more complex than while and do-while loops, but you should become really comfortable with them because you will be using them all the time! Here is the basic syntax of a for loop:

```
for (expr1; expr2; expr3)
   statement
```

The first expression is evaluated (executed) once at the beginning of the loop.

The second expression is evaluated at the beginning of each iteration. If it evaluates to the TRUE, the loop will continue, and the nested statement(s) are executed. If it evaluates to FALSE, the execution of the loop terminates. At the end of each iteration, the third expression is evaluated (executed).

Each of the expression can be empty or it can contain multiple expressions separated by commas. If the second expression is empty, PHP assumes it to be TRUE. Essentially this will create an infinite FOR loop. Let’s look at the following examples:

```php
<?php
// In this example we count from 1 to 10 using the most basic
and common form of the for loop ;
for ($i = 1; $i <= 10; $i++) { // we set the variable $i (counter
variable) to 1; In each iteration we check if this variable is
smaller than or equal to 10. If it is, the loop continues. At the
end of each iteration we increment $i by 1;
    echo $i; // we print the value of $i to the screen ;
}

// In this example we again count from 1 to 10, but we set it up
as an infinite loop that we break out of on a certain condition ;
for ($i = 1; ; $i++) {
    if ($i > 10) { // Essentially this if statement is the one that is
implicitly set up in the above example;
        break; // we break out of for loop if $i < 10;
    }
    echo $i; // we print $i to the screen;
}

// Again, we count from 1 to 10, but this time we expand the for
loop to see how it functions.
$i = 1; // we initialize $i to 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++; // this is essentially the third expression of the for
loop;
}
```

As with the while and if statements, for loops support the alternate colon syntax for control structures:

```
for (expr1; expr2; expr3):
    statement
    ...
endfor;
```

Very often, people will loop through arrays using for loops. While it is not wrong to do this, it is more convenient to use foreach loops for arrays or objects, because that control structure is specifically setup for arrays and provides you with more control.

Now that we've introduced the two types of looping in PHP, you may be asking yourself: What is the difference between a while and a for loop and when should you use each one? Here is the general rules of thumb that will guide you through the proper usage of for and while loops in your PHP applications.

For loops are for when you know how many iterations of the loop you need. For example, if you want to operate on each element of an array, you use a for loop that loops from the first to the last value of the array. The easiest way is if you have an array where each key is an incremental number. You initialize the counter variable to 0 (this is usually going to be the first value of the array. Remember, PHP counts from 0). After that, you check whether you have reached the last element of the array. To do that you check whether the counter variable is smaller than or equal to the length of the array (you can use the `count()` function to find the number of elements in the array.) The last expression will be incrementing the counter variable.

While loops on the other hand, are to be used when you don't know how many iterations your loop will take until the truth condition evaluates to FALSE.

A closing word about for loops:

For loops can be used on string characters as well. Can you set up a for loop that will print out the letters of the alphabet?

Foreach

The foreach construct provides an easy way to iterate over arrays and objects, but it will only work on arrays and objects and will give you an error if you try to use a foreach on a different kind of variable type. There are two syntaxes to the foreach loop:

```
foreach (array_expression as $value)
   statement
foreach (array_expression as $key => $value)
   statement
```

The first syntax loops over an array (array_expression)and takes the value of each element and assigns it to $value . In the second syntax, you reference both the key and the value of each element of the array. Let's look at an example:

```
<?php
$arr = array(1, 2, 3, 4);
foreacl ($arr as &$value) {
   $value = $value * 2;
}
// $arr is now array(2, 4, 6, 8)
unse($value); // break the reference with the last element
?>
```

Note that the reference to $value remains after the foreach loop, so you should unset it (destroy it) after the loop to prevent any issues with your code.

Foreach loops and while loops are very commonly used in tandem when working with database information, but we shall see that in the chapter concerning databases.

Break

So far we have seen the use of the break command to break out of loops, but we have not defined it explicitly until now.

break ends execution of the current for , foreach , while , do-while or switch structure.

break accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.

Here is an example of how the break command works with the optional argument:

```
<?php
$i = 0;
while (++$i) {
   switch ($i) {
   case 5:
      echo "At 5<br />\n";
      break 1;  /* Exit only the switch. */
   case 10:
      echo "At 10; quitting<br />\n";
      break 2;  /* Exit the switch and the while. */
   default:
      break;
   }
}
?>
```

Continue

The continue command is similar to the break command, but it does not break out of the loop. Instead it continues to the next iteration of the loop. This command can be used inside of switch statements as well, but the results of that would be the same using a break. You can think of it this way:

Continue takes you to just before the last closing curly bracket of your structure. If it is a loop, you will go to the next iteration of the loop. For a switch statement, when the execution is taken to just before the closing curly bracket, you are essentially being taken to the end of the switch statement.

In contrast, break will take you to just outside the last closing curly bracket of your code. For a loop that means that you have broken out of the loop. For a switch statement, that means that you break out of the switch statement and do not go through any of the cases.

As with the break statement, continue takes an optional argument that tells PHP how many levels of enclosing loops it should skip.

Return

The return command returns program control to the calling module. Execution resumes at the statement following the called module's invocation.

If you use return from within a function, the return statement immediately ends execution of the current function, and returns the argument of the return as the value of the function call.

Try to abstain from using return inside of a file that is included in another file as this is bad practice.

Include

Files are included based on the file path given or, if none is given, the include_path specified. If the file isn't found in the include_path, include will finally check in the calling script's own directory and the current working directory before failing. The include construct will emit a warning if it cannot find a file; this is different behavior from require, which will emit a fatal error.

If a path is defined — whether absolute (starting with a drive letter or \ on Windows, or / on Unix/Linux systems) or relative to the current directory (starting with . or ..) — the include_path will be ignored altogether. For example, if a filename begins with ../, the parser will look in the parent directory to find the requested file.

For more information on how PHP handles including files and the include path, see the documentation for include_path.

When a file is included, the code it contains inherits the variable scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file, from that point forward. However, all functions and classes defined in the included file have the global scope.

Require

Require is identical to include, except that upon failure, it will return an error and execution of the script will stop.

Require_once

This is similar to require, except require_once will check whether the file has been included up to that point. If it has, it will not include it again. Hence, it will require the file to be included only once.

Include_once

Works exactly like require_once, except it will not produce an error.

Conclusion

This is probably one of the most important chapters in the book. With the concepts and techniques you learned in this chapter you should be able to fully utilize your PHP skills to create amazing applications. The control structures introduced here are the basis of each PHP script you will write and encounter. Combine these with the next chapters about functions and database connections and usage and you will be well on your way to creating advanced and complicated web applications.

Exercise 1

In this PHP exercise, you will use a conditional statement to determine what gets printed to the browser. Write a script that gets the current month and prints one of the following responses, depending on whether it's August or not:

```
It's August, so it's really hot.
Not August, so at least not in the peak of the heat.
```

Hint: the function to get the current month is ' `date('F', time())` ' for the month's full name.

Exercise 2

In this PHP exercise, you will put all the loops through their paces. Write a script that will print the following to the browser:

```
abc abc abc abc abc abc abc abc abc
xyz xyz xyz xyz xyz xyz xyz xyz xyz
1 2 3 4 5 6 7 8 9
```

1. Item A
2. Item B
3. Item C
4. Item D
5. Item E
6. Item F

Create the 'abc' row with a while loop, the 'xyz' row with a do-while loop, and the last two sections with for loops. Remember to include HTML and source code line breaks in your output. No arrays allowed in this solution.

Exercise 3

Loops are very useful in creating lists and tables. In this PHP exercise, you will use a loop to create a list of equations for squares.

Using a for loop, write a script that will send to the browser a list of squares for the numbers 1-12.

Use the format, "1 * 1 = 1", and be sure to include code to print each formula on a different line.

Chapter Five – Functions

Introduction

Functions in PHP, like in many other languages, give you the power to create extremely versatile and reusable code. Imagine the following situations. You are writing a script and you want to perform a certain set of actions. Later on, you need to perform the same set of actions, but you have to change a variable to another variable, for example. Wouldn't it be convenient to be able to separate those actions out, and just replace the variable as needed? This is where functions come in.

In PHP, functions are broken up into user defined functions and pre-defined functions. User defined functions are custom functions that you create inside of your code. Pre-defined functions are the functions that PHP comes with.

User Defined Functions

A function is basically a block of statements that can be used repeatedly in a program that will only be executed when you call it. Functions can take parameters that you predefine or they can have no parameters. Let's look at the syntax for a function in PHP.

```
<?php
functio functionName({parameters go here}) {
   {code statements go here};
}
?>
```

The function name is defined by you. It is a good idea to give the function a name that reflects what the function will do. The naming convention for functions is pretty much the same as the naming convention for variables (functions can start with letters or underscores but not numbers). An important thing to note is that function names are not case-sensitive. Therefore, the following code will produce an error:

```
<?php
functio functionName($name) {
   echc $name;
}
functio FunctionName($name) {
   echc $name;
}
?>
Gives the following error:
Fatal error: Cannot redeclare FunctionName() (previously
declared...
```

If you want to call the function you created, you just write the name of the function followed by a pair of parentheses. If your function takes parameters, you put the corresponding variable or variables in between the parentheses.

```
<?php
functio msg(){
   echc 'Hello World!';
}
msg();
?>
```

Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many

arguments as you want, just separate them with a comma. Let's modify the previous example to include a name:

```
<?php
functio msg($name){
   echc 'Hello '.$name.'!';
}
$name = "Jon";
msg($name);
?>
```

Functions arguments can take default values. For example, if you call the function without specifying an argument, the function will assume the default argument value;

```
<?php
functio msg($name = 'user' ){
   echc 'Hello '.$name.'!';
}
$name = "Jon";
msg($name); // this will echo out the name Jon
msg(); // this will echo out Hello user!
?>
```

Return values

Often times, you want the function to return a specific value. To do this, you use the return statement:

```
<?php
functio sum($x, $y) {
   $z = $x + $y;
   returı $z;
}
?>
```

Variable Scope

When dealing with variables, it is important to know what their scope is (or the context in which they are defined). For example:

```
<?php
$var = 2;
includ 'functions.php' ;
?>
```

In this case, the variable $var can be used by any code that is included inside of functions.php . The

context of a variable, however, changes when we create our own functions.

Within user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope. For example:

```
<?php
$a = 1; /* global scope */
functio test()
{
   echc $a; /* reference to local scope variable */
}
test();
?>
```

This script will not produce any output because the echo statement refers to a local version of the `$a` variable, and it has not been assigned a value within this scope. You may notice that this is a little bit different from the C language in that global variables in C are automatically available to functions unless specifically overridden by a local definition. This can cause some problems in that people may inadvertently change a global variable. In PHP global variables must be declared global inside a function if they are going to be used in that function.

Let’s look at how to declare variables as global.

```
<?php
$a = 1;
$b = 2;
functio Sum()
{
   globa $a, $b;
   $b = $a + $b;
}
Sum();
echc $b;
?>
```

Pre-defined functions

PHP comes built with many pre-defined functions and this is what makes PHP so efficient and useful. We cannot possible examine all of the PHP functions as this will take another book itself (currently, the PHP documentation lists 9457 built in functions.). Let's start with the most basic functions that can be used inside of PHP.

Echo, Print, exit, die

Echo

Echo is the language construct in PHP that allows us to output string data. PHP will automatically convert numerical variables to strings. We've already used echo in previous examples in this book, so you should be familiar with this function by now.

Print

Print is another language construct that does the same thing as echo, but it is older and a bit slower. Therefore, it is usually better to use echo.

A function that is not exactly related to print but shares enough of the name for us to place it here, is the print_r() function. This function allows us to print the contents of an array to the screen.

Exit and Die

These functions allow you to print out a message to the screen and terminate the script. The two functions are equivalent to one another and you can use whichever. The syntax for both is:

```
<?php
die('error message goes here');
exit('error message goes here');
?>
```

String functions

PHP comes with a wide range of functions for string manipulation.

Word count

PHP comes with a pre-built function that will count the number of words in a string for you. It is of the following form:

```
<?php
$param = 0; // or 1, or 2 depending on what you need ;
$word_count = str_word_count($string, $param); // the return
value of the function is either an integer or an array ;
?>
```

The second argument of the function is optional and can take 3 values. Here are the supported values

and the result returned from each:

- 0 - returns the number of words found
- 1 - returns an array containing all the words found inside the string
- 2 - returns an associative array, where the key is the numeric position of the word inside the string and the value is the actual word itself

The function takes a third optional argument that we didn't specify here. The third parameter is a list of additional characters that you want to be considered as words. You just put the symbols next to each other in a string with no other formatting.

String Shuffle

This function allows you to randomly shuffle the characters inside of a string. This comes in handy if you want to create pseudo-unique names for files or something like that.

```
<?php
$string = "word";
echc str_shuffle($string); // returns something like: dorw
?>
```

Replace parts of a string

PHP has a number of different functions that can help you with this. The simplest one is the str_replace() function. It takes the form:

```
<?php
$string = "string to replace things inside of";
$string_new = str_replac($needle, $replacement , $haystack );
// finds all occurences of the string $needle inside of the string
$haystack and replaces those with $replacement;
?>
```

The arguments can be strings or arrays. If needle and replacement are arrays, then str_replace() takes a value from each array and uses them to search and replace on haystack . If replacement has fewer values than needle , then an empty string is used for the rest of replacement values. If needle is an array and replacement is a string, then this replacement string is used for every value of haystack . The converse would not make sense, though.

There is an optional fourth argument that could be passed which will be set to the number of replacements performed.

There is a modification to the str_replace function that will replace without regard to case. In other words, it will do a case-insensitive replace. This function is called str_ireplace and takes the same arguments.

String length

There are a number of functions in PHP that will return the length of a string. The simplest to use is the following:

```
<?php
$string = "string to replace things inside of";
$length = strlen($string); // Returns the length of a string ;
?>
```

Another function you can utilize to do this is the mb_strlen();

```
<?php
$string = "string to replace things inside of";
$length = mb_strlen($string); // Returns the length of a string ;
?>
```

This function has an optional parameter which is the encoding of the string. For example, if you want to set the encoding to UTF-8, you just provide the string 'UTF-8' . PHP supports numerous encoding types which you can easily find by going to the official PHP manual.

Another way to count the number of characters is to use the iconv_strlen() function.It works very much like the mb_strlen() function and takes the same parameters, except it is a lot stricter when it comes to bad sequences inside your string.

The reason for the different functions is that they actually count different things. strlen() counts the number of bytes inside of a string. Usually a character is equivalent to one byte, but that is not always the case. This is why the other functions exist.

Sometimes you will find that you have a string that has unneeded whitespace at the end of it.To get of whitespace on both sides of a string, you can use the trim() function.A useful feature of the trim() function is that it can trim not only whitespace but any additional characters that you specify.Alternatively, if you only want to trim from the left or the right, there are the ltrim() and rtrim() functions.

```
<?php
$str = "Hello World!";
echc $str . "<br>";
echc trim($str,"Hed!"); // outputs llo Worl
?>
```

Finding substrings in a string

The simplest way to return a part of a string is to use the substr() function. This function takes two required parameters and one optional parameter. The first parameter is the string that you are referencing. The second parameter is the start of the portion of the string. This parameter can be either positive or negative. If positive, the returned string will be the string that starts from the specified location within the main string. If negative, the returned string will start from the end of the main string. The third parameter is optional and is the length of the string you want returned. It can also be positive or negative. If it is positive, the returned string will contain at most, that number of characters. If the value is negative, the value specifies how many characters will be omitted from the

main string. Here are examples:

```
<?php
$rest = substr("abcdef" , -1);    // returns "f"
$rest = substr("abcdef" , -2);    // returns "ef"
$rest = substr("abcdef" , -3, 1); // returns "d"
$rest = substr("abcdef" , 0, -1);  // returns "abcde"
$rest = substr("abcdef" , 2, -1);  // returns "cde"
$rest = substr("abcdef" , 4, -4);  // returns false
$rest = substr("abcdef" , -3, -1); // returns "de"
?>
```

If you want to find the number of times a string occurs inside another string you can use the substr_count() function. This function takes two required parameters and two optional parameters. Here is the syntax for the function and some examples on how to use it in your own code:

```
substr_count(string,substring,{start},{length})
```

The first parameter is the string to check. The second parameter is the string to search for. The third parameter specifies where in the string to start searching. The third parameter specifies the length of the search.

```
<?php
$str = "This is nice";
echc strlen($str)."<br>"; // Using strlen() to return the string
length ; output = 12 ;
echc substr_count($str,"is")."<br>"; // The number of times " is
" occurs in the string ; output = 2 ;
echc substr_count($str,"is",2)."<br>"; // The string is now
reduced to " is is nice " ; output = 2 ;
echc substr_count($str,"is",3)."<br>"; // The string is now
reduced to " s is nice " ; output = 1 ;
echc substr_count($str,"is",3,3)."<br>"; // The string is now
reduced to " s i " ; output = 0 ;
?>
```

Another function that finds a substring inside a larger string is the strpos() function. This function finds the position of the first occurrence of a string inside another string. Let’s look at the syntax for the function as well as a couple of different examples to understand it:

```
strpos(string,find,{start})
```

The first parameter is the string to check. The second parameter is the string to search for. The third parameter specifies where in the string to start searching.

```
<?php
echo strpos("I love php, I love php too!","php"); // The output of this will be 7. This is the position in the string where we find a match.
?>
```

There is an alternative version to this function that is case-insensitive. This function is the stripos() function. It takes the same parameters as the strpos() function and works the same way except it is case-insensitive.

There are two similar functions that will return the last occurrence of a string inside another string. Again, there is a case-sensitive and a case-insensitive version of these functions.These are the strripos() and the strrpos() functions. They take the same parameters as the above two functions and work in the same way.

String case switches

Sometimes you will find yourself having to convert strings from all caps to lowercase or vice versa or even have to capitalize words. Luckily, there are functions in PHP that will save you from having to do that by yourself.

The two most common and reciprocal functions are strtolower() and strtoupper() . Both of these functions take only 1 parameter and that is the string that you want to edit. Here is an example of both functions:

```
<?php
echo strtolower("Hello WORLD.'); // outputs hello world.
echo strtoupper("Hello WORLD.'); // outputs HELLO WORLD.
?>
```

The other case functions that you might find useful arethe ucwords() and the ucfirst() . Here is an example:

```
<?php
echo ucfirst("hello world!"); // outputs the string "Hello world!" ;
echo ucwords("hello world!"); // outputs the string "Hello World!"
;
```

Strings to Arrays and back

There are two very important functions in PHP that allow you to convert a string into an array and an array into a string. We'll take a look at both.

The first function is the implode() function. This function takes an array and joins the array elements with a string. The syntax is the following:

```
implode(separator,array)
```

The separator specifies what to be placed in between the array elements. Here is an example:

```
<?php
$arr = array('Hello','World!','Beautiful','Day!');
echo implode(" ",$arr); // outputs "Hello World! Beautiful Day!" ;
?>
```

This is a very powerful function that, if used properly, can save you from having to write out code and cut down on execution time. Make sure to look at the exercises at the end of the chapter to see an example of how this function can be used to optimized code performance.

Let's look at the other function. We saw how to implode an array into a string, now let's look at how to explode a string into an array. The function is called explode() and takes two parameters.

```
implode(separator,string)
```

The separator is the part of the string that you want to separate by and the string is the string you are exploding. Here is an example:

```
<?php
$str = "Hello world. It's a beautiful day.";
print_r (explode(" ",$str));
?>
```

Conclusion

We've expanded our PHP knowledge even more in this chapter by covering the topics of functions in PHP. We covered user-defined functions and a small amount of the pre-defined functions in PHP. Here are some practice exercises to make you comfortable with functions.

Exercise 1

In chapter 2, exercises 2 and 3 you created an associative array that represented a menu and had to output some string based on the contents of the array. We revisit this array in this exercise. Using the same array, create a function that takes three parameters: the menu array, the category selection and meal selection. The function has to output an HTML string. The string should be a paragraph tag containing the content in the following format:

You picked: **{name of meal here (in bold)}**.
This meal is made from: {list ingredients here (comma-separated list)}.
The nutritional information for this meal is: {list nutritional information here (comma-separated list)}.

Do not use loops! Use the aforementioned string functions to do this.

Exercise 2

This PHP exercise has two parts. For the first, you will create a function to accept two arguments, perform a calculation using them, then return a sentence with the result to the browser. The function will calculate the area of a rectangle, with the two arguments being width and height. (Reminder: area = width * height.) The sentence to be returned is "A rectangle of length $l and width $w has an area of $area.", where $l and $w are the arguments and $area is the result.

Exercise 3

For this PHP exercise, first create an array called $months. Use the names of the months as keys, and the number of days for each month as values. For February, use the following for your value: "28 days, if leap year 29".

Next, write a function to create an option element for a form's select field. Make sure each option will be upper case. Both the array and the function should precede the HTML for the page.

Once again, you will be requesting user input. Create a form for the user with the request, "Please choose a month." Next, provide a select field with the months as options, looping through the array you created and using the function to create the option elements.

When the user clicks the submit button, return the statement "The month of $month has $number days.", where $month is the name of the month the user chose, and $number is the number of days. Be sure to include a different response for February.

Chapter Six – Databases

Introduction

In this chapter we will begin our discussion of using databases inside of our PHP applications. For this we will be using the MySQL database engine and the PHPMyAdmin interface that comes with your LAMP package of choice. We assume you have those installed and activated and are ready to continue.

To be able to manipulate databases, we will be using the mysqli API provided with PHP version 5 and later as well as the PDO abstraction layer. Let’s start by looking into what APIs are. Some of the following content is referenced directly from the PHP Manual.

What is an API?

An API (Application Programming Interface) defines the classes, methods, functions and variables that your application will need to call in order to carry out a desired task. In the case of PHP applications that need to communicate with databases the necessary APIs are usually exposed via PHP extensions.

APIs can be procedural or object-oriented. With a procedural API you call functions to carry out tasks, with the object-oriented API you instantiate classes and then call methods on the resulting objects. Of the two the latter is usually the preferred interface, as it is more modern and leads to better organized code.

We have not yet covered OOP (object-oriented programming) in PHP, so if you are not familiar with the concept, you can skip to chapter 10 of this book and then return to this chapter after you have gotten a hang of OOP!

There are several APIs available that will allow you to connect to the MySQL server. We will have a discussion of the options and which to choose depending on your application.

Connectors

MySQL provides documentation that explains all its terms. The term *connector* refers to a piece of software that allows your application to connect to the MySQL database server. MySQL has a variety of connectors depending on your language, including connectors for PHP.

When your application requires a database, you need to write your code to perform activities such as connecting to the database server, querying the database, updating the database, removing entries and other database-related functions. Software is required to provide the API that your PHP application will use, and also handle the communication between your application and the database server, possibly using other intermediate libraries where necessary. This software is known generically as a connector, as it allows your application to *connect* to a database server.

Drivers

A driver is a piece of software designed to communicate with a specific type of database server. The driver may also call a library, such as the MySQL Client Library or the MySQL Native Driver. These libraries implement the low-level protocol used to communicate with the MySQL database server.

By way of an example, the PHP Data Objects (PDO) database abstraction layer may use one of several database-specific drivers. One of the drivers it has available is the PDO MYSQL driver, which allows it to interface with the MySQL server.

Sometimes people use the terms connector and driver interchangeably, this can be confusing. In the MySQL-related documentation the term "driver" is reserved for software that provides the database-specific part of a connector package.

Extensions

In the PHP documentation you will come across another term - extension. The PHP code consists of a core, with optional extensions to the core functionality. PHP's MySQL-related extensions, such as

the mysqli extension, and the mysql extension, are implemented using the PHP extension framework.

An extension typically exposes an API to the PHP programmer, to allow its facilities to be used programmatically. However, some extensions which use the PHP extension framework do not expose an API to the PHP programmer.

The PDO MySQL driver extension, for example, does not expose an API to the PHP programmer, but provides an interface to the PDO layer above it.

The terms API and extension should not be taken to mean the same thing, as an extension may not necessarily expose an API to the programmer.

PHP MySQL APIs

PHP offers three main API options to connect to a MySQL server. One of those options, however, is deprecated and we will not include it in our discussion. Here are the three API options:

- MySQL Extension;
  - This is deprecated in newer version of PHP;
- MySQLi Extension;
- PHP Data Objects (PDO);

Each has its own advantages and disadvantages. The following discussion aims to give a brief introduction to the key aspects of each API.

mysqli Extension

The mysqli extension, or as it is sometimes known, the MySQL *improved* extension, was developed to take advantage of new features found in MySQL systems versions 4.1.3 and newer.
The mysqli extension is included with PHP versions 5 and later.

The mysqli extension has a number of benefits, the key enhancements over the mysqlextension being:

- Object-oriented interface
- Support for Prepared Statements
- Support for Multiple Statements
- Support for Transactions
- Enhanced debugging capabilities
- Embedded server support

PDO Extension

PHP Data Objects, or PDO, is a database abstraction layer specifically for PHP applications. PDO provides a consistent API for your PHP application regardless of the type of database server your application will connect to. In theory, if you are using the PDO API, you could switch the database server you used, from say Firebird to MySQL, and only need to make minor changes to your PHP code.

Other examples of database abstraction layers include JDBC for Java applications and DBI for Perl.

While PDO has its advantages, such as a clean, simple, portable API, its main disadvantage is that it doesn't allow you to use all of the advanced features that are available in the latest versions of MySQL server. For example, PDO does not allow you to use MySQL's support for Multiple

Statements.

PHPMyAdmin and getting familiar with MySQL

Let's start by discussing MySQL and the PHPMyAdmin graphical user interface. We presume that you would prefer to use this more intuitive option to the command line version of MySQL (unless you really want to have more control, in which case, go for it!).

Basics

MySQL is a database server that allows you to create and store multiple databases for use. A database is a place where you store all of the data for a given application. A database consists of multiple tables with entries. Each row of a table represents a database entry and each column of that table represents some particular information. Below is a picture of the PHPMyAdmin interface with a database open showing three tables inside:

Below is a picture of a single table showing multiple rows and columns:

MySQLi

Dual interface

The mysqli extension features a dual interface. It supports the procedural and object-oriented programming paradigm.

Users migrating from the old mysql extension may prefer the procedural interface. The procedural interface is similar to that of the old mysql extension. In many cases, the function names differ only by prefix. Some mysqli functions take a connection handle as their first argument, whereas matching functions in the old mysql interface take it as an optional last argument.

```
<?php
$mysqli = mysqli_connect ("example.com" , "user", "password",
"database" );
$res = mysqli_query($mysqli, "SELECT 'Please, do not use ' AS
_msg FROM DUAL");
$row = mysqli_fetch_assoc ($res);
echc $row['_msg' ];

$mysql = mysql_conne("example.com" , "user", "password" );
mysql_select_("test");
$res = mysql_quei("SELECT 'the mysql extension for new
developments.' AS _msg FROM DUAL", $mysql );
$row = mysql_fetch_ass($res);
echc $row['_msg' ];
?>
Output:
Please, do not use the mysql extension for new developments.
```

In addition to the classical procedural interface, users can choose to use the object-oriented interface. The documentation is organized using the object-oriented interface. The object-oriented interface shows functions grouped by their purpose, making it easier to get started. The reference section gives examples for both syntax variants.

There are no significant performance differences between the two interfaces. Users can base their choice on personal preference.

```
<?php
$mysqli = mysqli_connect("example.com", "user", "password",
"database");
if (mysqli_connect_errno($mysqli)) {
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$res = mysqli_query($mysqli, "SELECT 'A world full of ' AS _msg
FROM DUAL");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];

$mysqli = new mysqli("example.com", "user", "password",
"database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: " . $mysqli-
>connect_error;
}

$res = $mysqli->query("SELECT 'choices to please everybody.'
AS _msg FROM DUAL");
$row = $res->fetch_assoc();
echo $row['_msg'];
```

Connections

The MySQL server supports the use of different transport layers for connections. Connections use TCP/IP, Unix domain sockets or Windows named pipes.

The hostname localhost has a special meaning. It is bound to the use of Unix domain sockets. It is not possible to open a TCP/IP connection using the hostname localhost you must use 127.0.0.1 instead.

```
<?php
$mysqli = new mysqli("localhost", "user", "password",
"database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli-
>connect_errno . ") " . $mysqli->connect_error;
}
echo $mysqli->host_info . "\n";

$mysqli = new mysqli("127.0.0.1", "user", "password",
"database", 3306);
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli-
>connect_errno . ") " . $mysqli->connect_error;
}

echo $mysqli->host_info . "\n";
?>
Outputs:
```

```
Outputs:
Localhost via UNIX socket
127.0.0.1 via TCP/IP
```

Depending on the connection function used, assorted parameters can be omitted. If a parameter is not provided, then the extension attempts to use the default values that are set in the PHP configuration file.

```
mysqli.default_host=192.168.2.27
mysqli.default_user=root
mysqli.default_pw=""
mysqli.default_port=3306
mysqli.default_socket=/tmp/mysql.sock
```

The resulting parameter values are then passed to the client library that is used by the extension. If the client library detects empty or unset parameters, then it may default to the library built-in values.

Built-in connection library defaults

If the host value is unset or empty, then the client library will default to a Unix socket connection on localhost. If socket is unset or empty, and a Unix socket connection is requested, then a connection to the default socket on /tmp/mysql.sock is attempted.

On Windows systems, the host name . is interpreted by the client library as an attempt to open a Windows named pipe based connection. In this case the socket parameter is interpreted as the pipe

name. If not given or empty, then the socket (pipe name) defaults to \\.\pipe\MySQL.

If neither a Unix domain socket based not a Windows named pipe based connection is to be established and the port parameter value is unset, the library will default to port 3306.

The mysqlnd library and the MySQL Client Library (libmysqlclient) implement the same logic for determining defaults.

Connection options

Connection options are available to, for example, set init commands which are executed upon connect, or for requesting use of a certain charset. Connection options must be set before a network connection is established.

For setting a connection option, the connect operation has to be performed in three steps: creating a connection handle with mysqli_init(), setting the requested options using mysqli_options(), and establishing the network connection with mysqli_real_connect().

Executing statements

Statements can be executed with the mysqli_query(), mysqli_real_query() and mysqli_multi_query() functions. The mysqli_query() function is the most common, and combines the executing statement with a buffered fetch of its result set, if any, in one call. Calling mysqli_query() is identical to calling mysqli_real_query() followed by mysqli_store_result() .

```
<?php
$mysqli = new mysqli("example.com" , "user", "password",
"database" );
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli-
>connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " .
$mysqli->error;
}
```

After statement execution results can be retrieved at once to be buffered by the client or by read row by row. Client-side result set buffering allows the server to free resources associated with the statement results as early as possible. Generally speaking, clients are slow consuming result sets. Therefore, it is recommended to use buffered result sets. mysqli_query() combines statement execution and result set buffering.

PHP applications can navigate freely through buffered results. Navigation is fast because the result sets are held in client memory. Please, keep in mind that it is often easier to scale by client than it is to scale the server.

```php
<?php
$mysqli = new mysqli("example.com", "user", "password",
"database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1), (2),
(3)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " .
$mysqli->error;
}

$res = $mysqli->query("SELECT id FROM test ORDER BY id
ASC");

echo "Reverse order...\n";
for ($row_no = $res->num_rows - 1; $row_no >= 0; $row_no--
) {
    $res->data_seek($row_no);
    $row = $res->fetch_assoc();
    echo " id = " . $row['id'] . "\n";
}

echo "Result set order...\n";
$res->data_seek(0);
```

The above example will output:

```
Reverse order...
 id = 3
 id = 2
 id = 1
Result set order...
 id = 1
 id = 2
 id = 3
```

Prepared statements

The MySQL database supports prepared statements. A prepared statement or a parameterized statement is used to execute the same statement repeatedly with high efficiency.

Basic workflow

The prepared statement execution consists of two stages: prepare and execute. At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use.

The MySQL server supports using anonymous, positional placeholder with ?.

```
<?php
$mysqli = new mysqli("example.com" , "user", "password",
"database" );
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli-
>connect_errno . ") " . $mysqli->connect_error;
}

/* Non-prepared statement */
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli-
>query("CREATE TABLE test(id INT)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " .
$mysqli->error;
}

/* Prepared statement, stage 1: prepare */
if (!($stmt = $mysqli->prepare("INSERT INTO test(id) VALUES
(?)"))) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli-
```

Prepare is followed by execute. During execute the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values to execute it using the previously created internal resources.

```
<?php
/* Prepared statement, stage 2: bind and execute */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
    echo "Binding parameters failed: (" . $stmt->errno . ") " .
$stmt->error;
}
if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
?>
```

A prepared statement can be executed repeatedly. Upon every execution the current value of the bound variable is evaluated and sent to the server. The statement is not parsed again. The statement

template is not transferred to the server again.

```php
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " .
$mysqli->connect_error;
}

/* Non-prepared statement */
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli-
>query("CREATE TABLE test(id INT)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

/* Prepared statement, stage 1: prepare */
if (!($stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)"))) {
     echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}

/* Prepared statement, stage 2: bind and execute */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
    echo "Binding parameters failed: (" . $stmt->errno . ") " . $stmt->error;
}

if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}

/* Prepared statement: repeated execution, only data transferred from
client to server */
for ($id = 2; $id < 5; $id++) {
    if (!$stmt->execute()) {
        echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
    }
}

/* explicit close recommended */
$stmt->close();

/* Non-prepared statement */
$res = $mysqli->query("SELECT id FROM test");
var_dump($res->fetch_all());
?>
```

The above example will output:

```
array(4){
 [0]=>
 array(1){
  [0]=>
  string(1)"1"
 }
 [1]=>
 array(1){
  [0]=>
  string(1)"2"
 }
 [2]=>
 array(1){
  [0]=>
  string(1)"3"
 }
 [3]=>
 array(1){
  [0]=>
  string(1)"4"
 }
}
```

Every prepared statement occupies server resources. Statements should be closed explicitly immediately after use. If not done explicitly, the statement will be closed when the statement handle is freed by PHP.

Using a prepared statement is not always the most efficient way of executing a statement. A prepared statement executed only once causes more client-server round-trips than a non-prepared statement. This is why the SELECT is not run as a prepared statement above.

Conclusion

In this chapter we introduced some concepts related to creating databases and handling database connections using the MySQLi API in PHP. You should now be able to create your own databases using the PHPMyAdmin interface and connect to them with PHP.

Chapter Seven – Form Data

Introduction

One of the most powerful features of PHP is the way it handles HTML forms. The basic concept that is important to understand is that any form element will automatically be available to your PHP scripts.

In order to be able to use form data inside of PHP, you need to be familiar with HTML forms. We won't spend too much time on reviewing HTML forms. Here is an example HTML form:

```
<form action="action.php" method="post">
 <p>Your name: <input type="text" name="name" /></p>
 <p>Your age: <input type="text" name="age" /></p>
 <p><input type="submit" /></p>
</form>
```

There is nothing special about this form. It is a straight HTML form with no special tags of any kind. When the user fills in this form and hits the submit button, the action.php page is called. Alternatively, you could leave the action parameter empty and the information from the page will be sent to the same page you are on. (Essentially, it will refresh the page and make the form data available to you).

Methods for sending Form Data

There are two methods for sending form data – POST and GET.

GET is a more limited and less secure method of sending information (although there are ways, such as using unique tokens, to make GET a bit more secure). GET sends the form data to the URL of the page. Anytime you see a URL that ends with something that looks something like the following: ?key=value&key=value&key=value, you know that you are dealing with GET data.

POST is a bit more secure as users don't actually get to see the information being sent and it allows you to send more information.

Both of these methods are available to PHP through the $_GET and $_POST superglobal arrays.

Referencing information from forms

You will notice that each input element of the HTML form has its own name. This name corresponds to a key in the corresponding superglobal array. The "method" parameter of the form identifies whether you are using POST of GET.

Once we have the form data submitted we can just reference the corresponding elements of the array. For example, if we want to display all the information from the array we can do:

```
<?php
print_($_POST);
?>
```

We will get all of the key-value pairs inside of that array.

Security

Important!!

Anytime you are dealing with user submitted data, you must assume that the user using your site has bad intentions. Always assume that you are writing a site that someone is trying to hack. Never trust that your users are not going to exploit a vulnerability in your site. It's better to be overly secure than to not be secure.

When it comes to form data, you have to always be cautious. Think of it this way. When you give a user a field for them to type their name, who will guarantee that they will type their name in that field? What happens if they type some PHP code? Or maybe an SQL query? Or maybe some JavaScript? You have to be the one that makes sure you prevent XSS (cross-site scripting) attacks and SQL injection. Those are the two most common types of "web hacking" and will cause you intense headaches if you don't think ahead of time.

Think of the form from the previous example. When the user fills in this form and hits the submit button, the action.php page is called. In this file you would write something like this:

```
Hi <?php echc htmlspecialchars ($_POST['name' ]); ?>.
Youare <?php echc (in!)$_POST['age' ]; ?> yearsold.
```

The output will be:

```
Hi Joe. You are 22 years old.
```

Apart from the htmlspecialchars() and (int) parts, it should be obvious what this does. htmlspecialchars() makes sure any characters that are special in html are properly encoded so people can't inject HTML tags or Javascript into your page. For the age field, since we know it is a number, we can just convert it to an integer which will automatically get rid of any stray characters. You can also have PHP do this for you automatically by using the filter extension. The $_POST['name'] and $_POST['age'] variables are automatically set for you by PHP. Aabove we just introduced the $_POST superglobal which contains all POST data. Notice how the method of our form is POST. If we used the method GET then our form information would live in the $_GET superglobal instead.

Form validation is a wide topic that we will not cover in this book, but hopefully this example gives you an idea of what to do. Generally the page that is your form processing page, you will have a number of if statements that check a number of things. Here is a general outline of what you should do when validating a form:

- If a field is important, meaning that is must have a value (e.g. a password field for a user registering to your site) you have to check whether that value is set. For example, use something like: if(!isset($_POST['password'])) {do sth here};
- Generally it is a good idea to create an errors array. This array will contain all of the errors that occur with the form. When you validate and a validation of a field fails, you add an error. At the end, you display these errors to your page so the user knows what they did wrong

and can fix their input;

- Whenever you are sending information to a database, ALWAYS, (ALWAYS ALWAYS ALWAYS!) use the htmlspecialchars() function to make sure you don't get HTML/JS/PHP code injected into your code. After that you should use a mysqli_real_escape_string() function. This will escape any SQL characters (such as the apostrophes) so that you don't get SQL injection.
- If you know a field has to be an integer, or some other number, cast it to a number. That way any string inputs will be prevented;
- Always test and debug your form with different types of input. Think of yourself as a hacker. What would you try putting in the form field to make the site break?

Conclusion

In this brief chapter we covered some important concepts about dealing with form data and security issues. Hopefully you have understood the importance of security in your web applications.

Exercise 1

In the next PHP exercise, you will request input from the user, then move the user's response from one file to another and do something with it.

Create two separate files. The first will contain a form with one input field asking for the user's favorite city. Use the post method for the form. Although this file contains no PHP code, on my localhost, it needs the .php extension to successfully call the second file.

The second file will contain PHP code to process the user's response. (In this case, something very simple.) After the user clicks the submit button, echo back Your favorite city is $city., where $city is the input from the form.

Hint: the variable that contains the user's input is an array. Arrays will be addressed in future exercises, but this particular array needs to come into play here. The array variable is $_POST['name'], where 'name' is the name of your input field.

Exercise 2

One very useful thing you can do with PHP is include the request for user input and the response in the same file, using conditional statements to tell PHP which one to show. For this PHP exercise, rewrite the two files of the previous exercise into one file using an if-else conditional statement.

Hint: You'll need some way to tell if the form has been submitted. The function to determine if a variable has been set and is not null is isset().

Exercise 3

For this PHP exercise, you will use the same format as the previous exercise, requesting input in the first part, and responding in the second, through the magic of PHP's if-else statement. In the first section, give the user an input field and request that they enter a day of the week.

For the second section, you'll need the following poem:

```
Laugh on Monday, laugh for danger.
Laugh on Tuesday, kiss a stranger.
Laugh on Wednesday, laugh for a letter.
Laugh on Thursday, something better.
Laugh on Friday, laugh for sorrow.
Laugh on Saturday, joy tomorrow.
```

Using the else-elseif-else construction, set each line to output in response to the day the user inputs, with a general response for any input that is not in the poem.

Chapter Eight – Sessions and Cookies

Introduction

Session support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and increase the appeal of your web site.

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Sessions

Sessions are a simple way to store data for individual users against a unique session ID. This can be used to persist state information between page requests. Session IDs are normally sent to the browser via session cookies and the ID is used to retrieve existing session data. The absence of an ID or session cookie lets PHP know to create a new session, and generate a new session ID.

Sessions follow a simple workflow. When a session is started, PHP will either retrieve an existing session using the ID passed (usually from a session cookie) or if no session is passed it will create a new session. PHP will populate the $_SESSION superglobal with any session data after the session has started. When PHP shuts down, it will automatically take the contents of the $_SESSION superglobal, serialize it, and send it for storage using the session save handler.

By default, PHP uses the internal files save handler which is set by session.save_handler . This saves session data on the server at the location specified by the session.save_path configuration directive.

Sessions can be started manually using the session_start() function. If the session.auto_start directive is set to 1, a session will automatically start on request startup.

Sessions normally shutdown automatically when PHP is finished executing a script, but can be manually shutdown using the session_write_close() function.

```
<?php
$mysqli = mysqli_connect ("example.com" , "user", "password",
"database" );
if (mysqli_connect_errno ($mysqli )) {
    echo "Failed to connect to MySQL: " .
mysqli_connect_error ();
}

$res = mysqli_query($mysqli , "SELECT 'A world full of ' AS
_msg FROM DUAL");
$row = mysqli_fetch_assoc ($res);
echo $row['_msg' ];

$mysqli = new mysqli ("example.com" , "user", "password",
"database" );
if ($mysqli ->connect_errno) {
    echo "Failed to connect to MySQL: " . $mysqli -
>connect_error;
}

$res = $mysqli ->query("SELECT 'choices to please
everybody.' AS _msg FROM DUAL");
$row = $res->fetch_assoc ();
echo $row['_msg' ];
```

```
<?php
/* Unregistering a variable using Sessions */
session_sta();
unse($_SESSION['count']);
?>
```

Caution

Do NOT unset the whole $_SESSION ***with*** unset($_SESSION) ***as this will disable the registering of session variables through the*** $_SESSION ***superglobal.***

Passing the Session ID

There are two methods to propagate a session id:

- Cookies
- URL parameter

The session module supports both methods. Cookies are optimal, but because they are not always available, we also provide an alternative way. The second method embeds the session id directly into URLs.

PHP is capable of transforming links transparently. Unless you are using PHP 4.2.0 or later, you need to enable it manually when building PHP. Under Unix, pass --enable-trans-sid to configure. If this build option and the run-time option session.use_trans_sid are enabled, relative URIs will be changed to contain the session id automatically.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id . Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
<?php

session_sta();

if (empt($_SESSION['count'])) {
  $_SESSION['count'] = 1;
} else {
  $_SESSION['count']++;
}
?>

<p>
Hello visitor, you have seen this page <?php echo
$_SESSION['count']; ?> times.
</p>

<p>
To continue, <a href="nextpage.ph<?php echo
htmlspecialchars (SID); ?>">click
here</a>.
</p>
```

Custom Session Handlers

To implement database storage, or any other storage method, you will need to use session_set_save_handler() to create a set of user-level storage functions. As of PHP 5.4.0 you may create session handlers using the SessionHandlerInterface or extend internal PHP handlers by inheriting from SessionHandler .

The callbacks specified in session_set_save_handler() are methods called by PHP during the life-cycle of a session: open, read, write and close and for the housekeeping tasks: destroy for deleting a session and gc for periodic garbage collection.

Therefore, PHP always requires session save handlers. The default is usually the internal 'files' save handler. A custom save handler can be set using session_set_save_handler() . Alternative internal save handlers are also provided by PHP extensions, such as sqlite, memcache and memcached and can be set with session.save_handler .

When the session starts, PHP will internally call the open handler followed by the read callback which should return an encoded string exactly as it was originally passed for storage. Once the read callback returns the encoded string, PHP will decode it and then populate the resulting array into the $_SESSION superglobal.

When PHP shuts down (or when session_write_close() is called), PHP will internally encode the $_SESSION superglobal and pass this along with the session ID to the write callback. After the write

callback has finished, PHP will internally invoke the close callback handler.

When a session is specifically destroyed, PHP will call the `destroy` handler with the session ID.

PHP will call the `gc` callback from time to time to expire any session records according to the set max lifetime of a session. This routine should delete all records from persistent storage which were last accessed longer than the `$lifetime` .

Cookies

PHP transparently supports HTTP cookies. Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. You can set cookies using the setcookie() or setrawcookie() function. Cookies are part of the HTTP header, so setcookie() must be called before any output is sent to the browser. This is the same limitation that header() has. You can use the output buffering functions to delay the script output until you have decided whether or not to set any cookies or send any headers.

Any cookies sent to you from the client will automatically be included into a $_COOKIE auto-global array if variables_order contains "C". If you wish to assign multiple values to a single cookie, just add [] to the cookie name.

Depending on register_globals, regular PHP variables can be created from cookies. However it's not recommended to rely on them as this feature is often turned off for the sake of security.

For more details, including notes on browser bugs, see the setcookie() and setrawcookie() function.

A cookie is created with the setcookie() function.

```
setcookie(name,value,expire,path,domain,secure,httponly);
```

Creating and retrieving cookies with PHP

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 *
30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

The setcookie() function must appear BEFORE the <html> tag.

Modifying a cookie using PHP

To modify a cookie, just set (again) the cookie using the setcookie() function:

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 *
30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>

<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); //
86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Deleting a cookie using PHP

To delete a cookie, use the setcookie() function with an expiration date in the past:

```php
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

Check if Cookies are enabled using PHP

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

```php
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
} else {
    echo "Cookies are disabled.";
}
?>

</body>
</html>
```

Conclusion

In this chapter we covered creating and destroying Cookies and Sessions in PHP

Chapter Nine – File Handling

Introduction

File handling is an important part of all web applications. You will sometimes find it necessary to open and process a file or files for different reasons and tasks. With that being said, we must issue a word of caution. When you are manipulating files you must be very careful. You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

File Handling

Reading Files

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

```
<?php
echo readfile("webdictionary.txt"); // Note that breaklines in the
.txt file will be ignored ;
?>
```

Opening files

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, "webdictionary.txt", during the lessons:

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open
file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

Modes	Description
r	Open a file for read only. File pointer starts at the beginning of the file
w	Open a file for write only.

	Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	Creates a new file for write only. Returns FALSE and an error if file already exists
r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

PHP Reading files

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

PHP Closing files

The fclose() function is used to close an open file.

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

Tip!

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

PHP Create File

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

```
<?php
$myfile = fopen("testfile.txt", "w");
?>
```

PHP Write to File

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open
file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

```
John Doe
Jane Doe
```

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

```php
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open
file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

```
Mickey Mouse
Minnie Mouse
```

Conclusion

With that we conclude our discussion of PHP File Handling. You would find file handling to be useful if you want to store information on your server, but you don’t need a construct like a database because you are storing a small amount of information that will not change frequently, for example. There are many other examples for when File Handling would be useful, but we shall not go into analyzing those.

Chapter Ten – Object Oriented Programming

Introduction

Starting with PHP 5, the object model was rewritten to allow for better performance and more features. This was a major change from PHP 4. PHP 5 has a full object model.

Among the features in PHP 5 are the inclusions of visibility, abstract and final classes and methods, additional magic methods, interfaces, cloning and typehinting.

Object-oriented programming is a style of coding that allows developers to group similar tasks into classes. This helps keep code following the tenet "don't repeat yourself" (DRY) and easy-to-maintain.

One of the major benefits of DRY programming is that, if a piece of information changes in your program, usually only one change is required to update the code. One of the biggest nightmares for developers is maintaining code where data is declared over and over again, meaning any changes to the program become an infinitely more frustrating game of “Where's Waldo?” as they hunt for duplicated data and functionality.

OOP is intimidating to a lot of developers because it introduces new syntax and, at a glance, appears to be far more complex than simple procedural, or inline, code. However, upon closer inspection, OOP is actually a very straightforward and ultimately simpler approach to programming.

Basics

Before you can get too deep into the finer points of OOP, a basic understanding of the differences between objects and classes is necessary. This section will go over the building blocks of classes, their different capabilities, and some of their uses.

Right off the bat, there's confusion in OOP: seasoned developers start talking about objects and classes, and they appear to be interchangeable terms. This is not the case, however, though the difference can be tough to wrap your head around at first.

A class, for example, is like a blueprint for a house. It defines the shape of the house on paper, with relationships between the different parts of the house clearly defined and planned out, even though the house doesn't exist.

An object, then, is like the actual house built according to that blueprint. The data stored in the object is like the wood, wires, and concrete that compose the house: without being assembled according to the blueprint, it's just a pile of stuff. However, when it all comes together, it becomes an organized, useful house.

Classes form the structure of data and actions and use that information to build objects. More than one object can be built from the same class at the same time, each one independent of the others. Continuing with our construction analogy, it's similar to the way an entire subdivision can be built from the same blueprint: 150 different houses that all look the same but have different families and decorations inside.

Class

Basic class definitions begin with the keyword class, followed by a class name, followed by a pair of curly braces which enclose the definitions of the properties and methods belonging to the class.

The class name can be any valid label, provided it is not a PHP reserved word. A valid class name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: ^[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*$.

A class may contain its own constants, variables (called "properties"), and functions (called "methods").

```
<?php
class SimpleClass
{
    // property declaration
    publi $var = 'a default value';

    // method declaration
    publi functio displayVar() {
        echo $this->var;
    }
}
?>
```

The pseudo-variable $this is available when a method is called from within an object context. $this is a reference to the calling object (usually the object to which the method belongs, but possibly another object, if the method is called statically from the context of a secondary object).

```
<?php
clas A
{
   functio foo()
   {
      i (isse($this)) {
         ech '$this is defined (';
         ech get_class($this);
         ech ")\n";
      } els {
         ech "\$this is not defined.\n";
      }
   }
} // continued on next page

clas B
{
   functio bar()
   {
      // Note: the next line will issue a warning if E_STRICT
is enabled.
      A::foo();
   }
}

$a = nev A();
$a->foo();

// Note: the next line will issue a warning if E_STRICT is
enabled.
A::foo();
$b = nev B();
$b->bar();

// Note: the next line will issue a warning if E_STRICT is
enabled.
B::bar();
```

```
class B
{
    function bar()
    {
        // Note: the next line will issue a warning if E_STRICT is
enabled.
        A::foo();
    }
}

$a = new A();
$a->foo();

// Note: the next line will issue a warning if E_STRICT is
enabled.
A::foo();
$b = new B();
$b->bar();

// Note: the next line will issue a warning if E_STRICT is
enabled.
B::bar();
?>
Output:
$this is defined (A)
$this is not defined.
```

New

To create an instance of a class, the new keyword must be used. An object will always be created unless the object has a constructor defined that throws an exception on error. Classes should be defined before instantiation (and in some cases this is a requirement).

If a string containing the name of a class is used with new, a new instance of that class will be created. If the class is in a namespace, its fully qualified name must be used when doing this.

```
<?php
$instance = new SimpleClass();

// This can also be done with a variable:
$className = 'Foo';
$instance = new $className(); // Foo()
?>
```

In the class context, it is possible to create a new object by new self and new parent.

When assigning an already created instance of a class to a new variable, the new variable will access

the same instance as the object that was assigned. This behaviour is the same when passing instances to a function. A copy of an already created object can be made by cloning it.

```
<?php

$instance = new SimpleClass();

$assigned   =  $instance;
$reference  =& $instance;

$instance->var = '$assigned will have this value';

$instance = null; // $instance and $reference become null

var_dump($instance);
var_dump($reference);
var_dump($assigned);
?>
NULL
NULL
object(SimpleClass)#1 (1) {
   ["var"]=>
     string(30) "$assigned will have this value"
}
```

Extends

A class can inherit the methods and properties of another class by using the keyword extends in the class declaration. It is not possible to extend multiple classes; a class can only inherit from one base class.

The inherited methods and properties can be overridden by redeclaring them with the same name defined in the parent class. However, if the parent class has defined a method as final, that method may not be overridden. It is possible to access the overridden methods or static properties by referencing them with parent::.

When overriding methods, the parameter signature should remain the same or PHP will generate an E_STRICT level error. This does not apply to the constructor, which allows overriding with different parameters.

```
<?php
clas: ExtendClass extend SimpleClass
{
   // Redefine the parent method
   functio displayVar()
   {
      echc "Extending class\n";
      paren::displayVar();
   }
}

$extended = new ExtendClass();
$extended->displayVar();
?>
Outputs:
Extending class
a default value
```

Properties

Class member variables are called "properties". You may also see them referred to using other terms such as "attributes" or "fields", but for the purposes of this reference we will use "properties". They are defined by using one of the keywords public, protected, or private, followed by a normal variable declaration. This declaration may include an initialization, but this initialization must be a constant value--that is, it must be able to be evaluated at compile time and must not depend on run-time information in order to be evaluated.

Within class methods non-static properties may be accessed by using -> (Object Operator): $this->property (where property is the name of the property). Static properties are accessed by using the :: (Double Colon): self::$property.

The pseudo-variable $this is available inside any class method when that method is called from within an object context. $this is a reference to the calling object (usually the object to which the method belongs, but possibly another object, if the method is called statically from the context of a secondary object).

```
<?php
clas: SimpleClass
{
   // invalid property declarations:
   publi $var1 = 'hello ' . 'world';
   publi $var2 = <<<EOD
hello world
EOD;
   publi $var3 = 1+2;
   publi $var4 = self::myStaticMethod();
   publi $var5 = $myVar;

   // valid property declarations:
   publi $var6 = myConstant;
   publi $var7 = array(true, false);

   // This is allowed only in PHP 5.3.0 and later.
   publi $var8 = <<<'EOD'
hello world
EOD;
}
?>
```

Constants

It is possible to define constant values on a per-class basis remaining the same and unchangeable. Constants differ from normal variables in that you don't use the $ symbol to declare or use them.

The value must be a constant expression, not (for example) a variable, a property, a result of a mathematical operation, or a function call.

As of PHP 5.3.0, it's possible to reference the class using a variable. The variable's value cannot be a keyword (e.g. self, parent and static). Let’s look at some examples:

```
<?php
class MyClass
{
    const CONSTANT = 'constant value';

    function showConstant() {
        echo  self::CONSTANT . "\n";
    }
}

echo MyClass::CONSTANT . "\n";

$classname = "MyClass";
echo $classname::CONSTANT . "\n"; // As of PHP 5.3.0

$class = new MyClass();
$class->showConstant();

echo $class::CONSTANT."\n"; // As of PHP 5.3.0
?>
```

Autoloading Classes

Many developers writing object-oriented applications create one PHP source file per class definition. One of the biggest annoyances is having to write a long list of needed includes at the beginning of each script (one for each class).

A good way to avoid having to write multiple includes is to use the spl_autoload_register() function. Here is an example:

```
<?php
/*
You would include this top part of the file in some initilization
file that is included in all other pages.
*/
spl_autoload_register('myAutoloader');

functio myAutoloader($className)
{
    $path = '/path/to/class';

    includ $path.$className.'.php';
}

//-----------------------------------

$myClass = new MyClass();
?>
```

In the example, above, "MyClass" is the name of the class that you are trying to instantiate, PHP passes this name as a string to spl_autoload_register(), which allows you to pick up the variable and use it to "include" the appropriate class/file. As a result, you don't specifically need to include that class via an include/require statement...

Just simply call the class you want to instantiate like in the example above, and since you registered a function (via spl_autoload_register()) of your own that will figure out where all your class are located, PHP will use that function.

Constructors and destructors

PHP 5 allows developers to declare constructor methods for classes. Classes which have a constructor method call this method on each newly-created object, so it is suitable for any initialization that the object may need before it is used.

Important!

Parent constructors are not called implicitly if the child class defines a constructor. In order to run a parent constructor, a call to parent::__construct() *within the child constructor is required. If the child does not define a constructor then it may be inherited from the parent class just like a normal class method (if it was not declared as private).*

```
<?php
clas: BaseClass {
   functio __constru() {
      prin "In BaseClass constructor\n";
   }
}

clas: SubClass extend BaseClass {
   functio __constru() {
      paren::__constru();
      prin "In SubClass constructor\n";
   }
}

clas: OtherSubClass extend BaseClass {
    // inherits BaseClass's constructor
}

// In BaseClass constructor
$obj = nev BaseClass();

// In BaseClass constructor
// In SubClass constructor
$obj = nev SubClass();

// In BaseClass constructor
$obj = nev OtherSubClass();
?>
```

PHP 5 introduces a destructor concept similar to that of other object-oriented languages, such as C++. The destructor method will be called as soon as there are no other references to a particular object, or in any order during the shutdown sequence.

```
<?php
class MyDestructableClass {
  function __construct() {
     print "In constructor\n";
     $this->name = "MyDestructableClass";
  }

  function __destruct() {
     print "Destroying " . $this->name . "\n";
  }
}

$obj = new MyDestructableClass();
?>
```

Like constructors, parent destructors will not be called implicitly by the engine. In order to run a parent destructor, one would have to explicitly call parent::__destruct() in the destructor body. Also like constructors, a child class may inherit the parent's destructor if it does not implement one itself.

The destructor will be called even if script execution is stopped using exit() . Calling exit() in a destructor will prevent the remaining shutdown routines from executing.

Object Inheritance

Inheritance is a well-established programming principle, and PHP makes use of this principle in its object model. This principle will affect the way many classes and objects relate to one another.

For example, when you extend a class, the subclass inherits all of the public and protected methods from the parent class. Unless a class overrides those methods, they will retain their original functionality.

This is useful for defining and abstracting functionality, and permits the implementation of additional functionality in similar objects without the need to reimplement all of the shared functionality.

```
<?php

clas: Foo
{
    publi functio printItem($string)
    {
        echc 'Foo: ' . $string . PHP_EO;_
    }

    publi functio printPHP()
    {
        echc 'PHP is great.' . PHP_EO;_
    }
}

clas: Bar extend Foo
{
    publi functio printItem($string)
    {
        echc 'Bar: ' . $string . PHP_EO;_
    }
}
```

```
$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Output: 'Foo: baz'
$foo->printPHP();       // Output: 'PHP is great'
$bar->printItem('baz'); // Output: 'Bar: baz'
$bar->printPHP();       // Output: 'PHP is great'

?>
```

Scope Resolution Operator (::)

The Scope Resolution Operator (also called Paamayim Nekudotayim) or in simpler terms, the double colon, is a token that allows access to static, constant, and overridden properties or methods of a class.

When referencing these items from outside the class definition, use the name of the class.

As of PHP 5.3.0, it's possible to reference the class using a variable. The variable's value can not be a keyword (e.g. self, parent and static).

```
<?php
clas MyClass {
    cons CONST_VALUE = 'A constant value';
}

$classname = 'MyClass';
ech $classname::CONST_VALUE; // As of PHP 5.3.0

ech MyClass::CONST_VALUE;
?>
```

Three special keywords self, parent and static are used to access properties or methods from inside the class definition.

```
<?php
clas OtherClass extend MyClass
{
    publi stati $my_static = 'static var';

    publi stati functio doubleColon() {
        ech paren::CONST_VALUE . "\n";
        ech self::$my_static . "\n";
    }
}

$classname = 'OtherClass';
ech $classname::doubleColon(); // As of PHP 5.3.0

OtherClass::doubleColon();
?>
```

When an extending class overrides the parents definition of a method, PHP will not call the parent's method. It's up to the extended class on whether or not the parent's method is called. This also applies to Constructors and Destructors, Overloading, and Magic method definitions.

```
<?php
clas MyClass
{
    protecte functio myFunc() {
        ech "MyClass::myFunc()\n" ;
    }
}

clas OtherClass extend MyClass
{
    // Override parent's definition
    publi functio myFunc()
    {
        // But still call the parent function
        paren::myFunc();
        ech "OtherClass::myFunc()\n" ;
    }


}

$class = ne OtherClass();
$class->myFunc();
?>
```

Conclusion

Let’s look at a few exercises for Classes and Objects before we close this book.

Exercise 1
In this PHP exercise, you will build the beginnings of a user registration form. To do this, you will create a class for making the select field, then use an object derived from the class in the form.

First of all, write an array that includes browser types: Firefox, Chrome, Internet Explorer, Safari, Opera, Other.

Then begin to write the class Select. You will need two properties, $name for the name of the select field, and $value , an array to provide the option values. You will also need four methods in addition to the two methods you will adapt: setName, getName, setValue, getValue. Checking to be sure the value is an array belongs in the setValue method, so write that here, and delete it from makeSelect .

Now we come to the two functions you wrote to generate the select field. Change the makeOptions function to iterate over the array argument's values rather than keys. This will be your fifth method. Then revise the makeSelect function to be the sixth method in your class.

Next comes the HTML. Write a user registration form asking for name, username, email, browser. Use text fields to collect the user data for the first three, then instantiate an object based on your class for the select field. When the user clicks the submit button, return the data as confirmation.

If you were creating a registration form to use on the Web, you would want to collect the data in a database. However, using PHP with mySQL or other databases is beyond the scope of this website.

Exercise 2
In the last PHP exercise, the Select class may have seemed like an awful lot of code to write for a simple select field. The real value of classes and objects doesn't become apparent until you have reason to reuse the code. So this time, you will expand your user registration form to use several select fields.

Assume that you have good reason to need data about your users' browsing capabilities. Either you want to tune your site, the content of your site concerns these issues, or something similar. Using your select class, you can reuse the class code as often as you like to create select fields.

To build this new version of the registration form, start with the script you wrote for Classes Ex. #1. Add the value of None as the first value in the $browsers array. Write two more arrays: $speeds, including values Unknown, DSL, T1, Cable, Dialup, Other; and $os, including Windows, Linux, Macintosh, Other. (Of course, these could be screen resolution or flash version or any other relevant capability.)

You want data for how the user browses both at home and work. Above the browser select field, add the subheading Work Access, and rename the browser label Primary Browser. (We all know that many people use more than one.) Below that, add labels and select field objects for Connection

Speed and Operating System. Next, add the subheading Home Access, with three new select fields corresponding to the ones you created for Work Access.

Since you are using so many objects in this script, it's a good idea to destroy each one after it has done its work. This will free up the memory the object occupied.

When the user hits the submit button, return the user's select field choices in two bulleted lists under the same headings (Work Access, Home Access).

Exercise 3

If you completed PHP Classes Ex. #1 and #2, you have now written a working user registration form. Time to tweak it and make it better.

First of all, it would be preferable to have the message --Select one-- at the top of each select field. Add a line to the makeSelect() method to accomplish this. The value should be No response. You won't need the "None" value at the top of the $browsers array, so delete that. With this change to class Select, you can see how using a class can simplify your work. One line of code, and all the select fields update.

Your user responses won't be very useful without some basic information, so the next task is to make three of the fields required. Above the form, add * Indicates required field. Then add an asterisk to the Name, Username, and Email fields.

Next, add code to validate the data in those three fields. This code will appear in the second half of the script, after you have retrieved data from the $_POST[] variable. The function empty() will let you know if there is data in the field. To help the user supply missing information, include a back button with the error message. (If you completed Forms Ex. #3, you have already written one of those.)

The email field is a special case. Not only can you check for the presence of data, you can check for an @(at symbol), which would be included in any valid email address. So here the data must satify two conditions to be acceptable. You can use the strpos() function to confirm the presence of the @ character.

Congratulations! You did it. This was the last chapter in our PHP course. You should now be able to create your own applications and tackle all sorts of projects. We hope you enjoyed this book as much as we did writing it. Hopefully you found it useful and you took away some useful concepts and techniques.

Answers to Exercises

These are the answers to exercises after each chapter. Keep in mind that many of these exercises can be solved in different ways, so your code does not need to be exactly like the one provided below to work properly. In many cases the code provided is just the general template from you can develop your own code for the exercise.

Chapter 2

Exercise 1

```php
<?php
// Solution to CH 2, EX 1
$user_data = array(
    'username'  => 'test_user',
    'name'      => array(
        'first'  => 'John' ,
        'last'   => 'Doe',
    ),
    'admin'     => 0,
    'user_id'   => '42',
);

//1.
// Concatenates the first name, a space and the last name.
$name = $user_data['name']['first'].' '.$user_data['name']['last'];

//2.
    //a.
    $user_data['user_id'] = (int)$user_data['user_id'];
    //b.
    $user_data['admin'] = (boo)$user_data['admin'];
//3
$user_data['email'] = 'john.doe@example.com' ;
?>
```

Exercise 2

```php
<?php
// Solution to CH 2, EX 2
$menu = array(
   'cat1' => array(
      'meal1' => array(
         'ingredients'  => array('ingredient  1', 'ingredient  2', 'ingredient  3'),
         'nutritional'  => array('nutrient1:  50g', 'nutrient2:  40g', 'nutrient3:  20g'),
      ),
      'meal2' => array(
         'ingredients'  => array('ingredient  1', 'ingredient  2', 'ingredient  3'),
         'nutritional'  => array('nutrient1:  32g', 'nutrient2:  42g', 'nutrient3:  21g'),
      ),
      'meal3' => array(
         'ingredients'  => array('ingredient  1', 'ingredient  2', 'ingredient  3'),
         'nutritional'  => array('nutrient1:  53g', 'nutrient2:  13g', 'nutrient3:  6g'),
      ),
      'meal4' => array(
         'ingredients'  => array('ingredient  1', 'ingredient  2', 'ingredient  3'),
         'nutritional'  => array('nutrient1:  78g', 'nutrient2:  12g', 'nutrient3:  0g'),
      ),
   ),
);
?>
```

Exercise 3

```
<?php
// Solution to CH 2, EX 3
$cat = 'cat1';
$meal = 'meal2';
echc $msg = "You picked: $mea <br/>
This meal is made from: {$menu[$cat][$meal]['ingredients'][0]} ,
{$menu[$cat][$meal]['ingredients'][1]} , {$menu[$cat][$meal]['ingredients'][2]}
<br/>
The nutritional information for this meal is: {$menu[$cat][$meal]['nutritional'][0]} ,
{$menu[$cat][$meal]['nutritional'][1]} , {$menu[$cat][$meal]['nutritional'][2]} ";
?>
```

Chapter 3

Exercise 1

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1"/>
<title> Arithmetic-Assignment Operators</title>
</head>

<body>

<?php

$num = 8;
echo "Value is now $num.<br/>";

$num += 2;
echo "Add 2. Value is now $num. <br/>";

$num -= 4;
echo "Subtract 4. Value is now $num. <br/>";

$num *= 5;
echo "Multiply by 5. Value is now $num. <br/>";

$num /= 3;
echo "Divide by 3. Value is now $num. <br/>";

$num++;
echo "Increment value by one. Value is now $num.<br/>";

$num--;
echo "Decrement value by one. Value is now $num.";

?>

</body>
</html>
```

Exercise 2

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
<title> Concatenation of Strings</title>
</head>

<body>

<?php

$around = "around";
echo 'What goes ' . $around . ' comes ' . $around . '.';

?>

</body>
</html>
```

Exercise 3

```
<html>
<head>
<meta http-equiv ="content-type"content ="text/html;charset=iso-8859-1">
<title> VariableDataTypes</title>
</head>

<body>
<h2> PHPVariableData Types</h2>
<p>
<?php

$whatsit = 'George' ;
echo "Value is ".gettype ($whatsit ).".<br/>\n" ;

$whatsit = 88.9 ;
echo "Value is ".gettype ($whatsit ).".<br/>\n" ;

$whatsit = true;
echo "Value is ".gettype ($whatsit ).".<br/>\n" ;

$whatsit = 8;
echo "Value is ".gettype ($whatsit ).".<br/>\n" ;

$whatsit = null;
echo "Value is ".gettype ($whatsit ).".<br/>\n" ;

?>
</p>
</body>
</html>
```

Chapter 4

Exercise 1

```
<html >
<head>
<meta http-equiv ="content-type" content ="text/html;charset=iso-8859-1">
<title> If-ElseStatement</title>
</head>

<body>
<h2>If-ElseStatement</h2>

<?php

$currMonth =date('F', time());
if ($currMonth  == 'August' ){
 echo "<p>It's  August, so it's really  hot.</p>" ;
}else{
 echo "<p>Not  August, so at least  not in the peak  of the heat.</p>" ;
}

?>

</body>
</html>
```

Exercise 2

```
<html>
<head>
<meta http-equiv ="content-type" content ="text/html;charset=iso-8859-1" />
<title> SimpleLoops</title>
</head>

<body>
<h2>SimpleLoops</h2>

<?php

echo "<p>\n" ;

$counter = 1;
while ($counter < 10){
  echo 'abc ';
  $counter ++;
}

echo "</p>\n" ;
echo "<p>\n" ;

$counter = 1;
do{
  echo 'xyz ';
  $counter ++;
} while ($counter < 10) ;

echo "</p>\n" ;

for ($x=1; $x<10; $x++){
  echo "$x ";
}
//The space inside the "" is necessary to separate the numbers.

//Generate ordered list.
echo "\n<ol>" ;
for ($x='A'; $x<'G'; $x++){
  echo "<li>Item $x</li>\n" ;
  }
echo "\n</ol>" ;
//Note that letters may be used in the for loop in place of numbers.

?>

</body>
</html>
```

Exercise 3

```
<html >
<head>
<meta http-equiv ="content-type"content ="text/html;charset=iso-8859-1">
<title> Squaresforthe Numbers1-12</title>
</head>

<body>
<h2> Squaresforthe Numbers1-12</h2>

<?php

for ($x=1; $x<=12; $x++){
  $result  = $x * $x;
  echo "$x * $x = $result <br />\n";
}

?>

</body>
</html>
```

Chapter 5

Exercise 1

```
<?php
// Solution to CH 2, EX 2
$menu = array(
   'cat1' => array(
      'meal1' => array(
         'ingredients' => array('ingredient 1', 'ingredient 2', 'ingredient 3'),
         'nutritional' => array('nutrient1: 50g', 'nutrient2: 40g', 'nutrient3: 20g'),
      ),
      'meal2' => array(
         'ingredients' => array('ingredient 1', 'ingredient 2', 'ingredient 3'),
         'nutritional' => array('nutrient1: 32g', 'nutrient2: 42g', 'nutrient3: 21g'),
      ),
      'meal3' => array(
         'ingredients' => array('ingredient 1', 'ingredient 2', 'ingredient 3'),
         'nutritional' => array('nutrient1: 53g', 'nutrient2: 13g', 'nutrient3: 6g'),
      ),
      'meal4' => array(
         'ingredients' => array('ingredient 1', 'ingredient 2', 'ingredient 3'),
         'nutritional' => array('nutrient1: 78g', 'nutrient2: 12g', 'nutrient3: 0g'),
      ),
   ),
   // Repeat for other categories here ;
);

$category = 'cat1'; // We pick which category we want ;
$meal = 'meal1'; // We pick our meal ;

$meal_choice = 'You picked: <b>'.$meal.'</b> <br> ';
// we create our output string and add the info for which meal we picked. Not that we are
using HTML bold tags in our string ;

$made_from = 'This meal is made from: '.trim(implode(",
",$menu[$category][$meal]['ingredients']), ',').'.<br>';
/*
We define a string that stores our information for ingredients.
We add the static part of the string and then we add the dynamically generated end.
To do that we select the 'ingredients' array from the 'meal1' array.
We implode that array using a comma and a space as a separator.
If we leave it like this, we will be left with a string that has a trailing comma at the end of
it though.
In order to avoid that, we use the trim function to trim the whitespace and specify a
comma in a charset so that it will be trimmed as well.
We then concatenate this imploded value (which is now a string value) to the static
string. At the end we append a period to end the sentence and a break tag.
*/
$nutritional = 'The nutritional information for this meal is: '.trim(implode(",
",$menu[$category][$meal]['nutritional']), ',').'.';
/*
We are essentially doing the same operation here as we did above.
We define a string that stores our information for nutrients.
We add the static part of the string and then we add the dynamically generated end.
To do that we select the 'nutritional' array from the 'meal1' array.
We implode that array using a comma and a space as a separator.
```

```
$nutritional = 'The nutritional information for this meal is: '.trim(implode(",
",$menu[$category][$meal]['nutritional' ]), ',').'.';
/*
We are essentially doing the same operation here as we did above.
We define a string that stores our information for nutrients.
We add the static part of the string and then we add the dynamically generated end.
To do that we select the 'nutritional' array from the 'meal1' array.
We implode that array using a comma and a space as a separator.
If we leave it like this, we will be left with a string that has a trailing comma at the end of
it though.
In order to avoid that, we use the trim function to trim the whitespace and specify a
comma in a charset so that it will be trimmed as well.
We then concatenate this imploded value (which is now a string value) to the static string.
At the end we append a period to end the sentence and a break tag.
*/
$meal_choice .= $made_from.$nutritional ; // We concatenate the three strings together in
the correct order.
echo $meal_choice ; // we echo out our string to test the results;
?>
```

Exercise 2

```
<html>
<head>
<meta http-equiv ="content-type"content ="text/html;charset=iso-8859-1">
<title> RectangleArea Function</title>
</head>

<body>
<h2>RectangleAreaFunction</h2>

<?php
//Define function.
functioı recArea($l, $w){
 $area = $l * $w;
 echc "A rectangle of length $l and width $w has an area of $are."; 
}

//Call function.
recArea(2, 4);
?>

</body>
```

```
<?php
//Create array.
$months = array(
  'January' => 31,
  'February' => '28 days, if leap year 29',
  'March' => 31,
  'April' => 30,
  'May' => 31,
  'June' => 30,
  'July' => 31,
  'August' => 31,
  'September' => 30,
  'October' => 31,
  'November' => 30,
  'December' => 31
);

//Define function. Use built in string function to make each option upper case.
//Don't forget to escape the quotes in your HTML.
function option($str){
  echo "<option value=\"$str\">" .ucfirst($str). "</option>\n" ;
}

?>
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
<title>Days in a Month</title>
</head>

<body>
<h2>Days in a Month</h2>

<?php
//If form not submitted, show form.
if(!isset($_POST['submit'])){
?>
<form method="post" action="yourfile.php">
<p>Please choose a month.</p>
<select name="month">
<?php
//Create options using the array and the function.
  foreach($months as $k => $v){
    option($k);
  }
?>
</select>
<p />
<input type="submit" name="submit" value="Go" />
</form>
```

Exercise 3

```
<?php
//If form submitted, respond to user.
} else {
//Retrieve user input.
  $month = $_POST['month' ];
//Allow for February using a conditional statement.
  if ($month == 'February' ){
    echo "The month of February has " . $months['February' ] . ".";
  }else{
    echo "The month of $month has $months[$month] days." ;
  }
}
?>

</body>
</html>
```

Chapter 7

Exercise 1

```
<?php
/* yourfile.php */
?>
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
<title>SimpleResponse- FavoriteCity</title>
</head>

<body>
<h2>FavoriteCity</h2>

<?php

//Retrieve string from post submission
$city = $_POST['city'];
echo "Your favorite city is $city.";

?>
```

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
<title>SimpleForm- FavoriteCity</title>
</head>

<body>
<h2>FavoriteCity</h2>

<form method="post" action="yourfile.php">
Pleaseenteryour favoritecity: <br />
<input type="text" name="city" />
<p />
<input type="submit" name="submit" value="Go" />
</form>

</body>
</html>
```

Exercise 2

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1" />
<title> Form Response with If-Else Statement</title>
</head>

<body>
<h2>Favorite City</h2>

<?php
//If form not submitted, display form.
if (!isset($_POST['submit'])){
?>

<form method="post" action="yourfile.php">
<!--Make sure you have entered the name you gave the file as the action.-->
Please enter your favorite city: <br />
<input type="text" name="city" />
<p />
<input type="submit" name="submit" value="Go" />
</form>

<?php
//If form submitted, process input.
}else{

//Retrieve string from form submission.
$city = $_POST['city'];
echo "Your favorite city is $city.";

}
?>

</body>
</html>
```

Exercise 3

```
<!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" xml:lang= "en" lang="en">
<head>
<meta http-equiv ="content-type" content ="text/html;charset=iso-8859-1" />
<title> If-Elseif-Else Days of the Week</title>
</head>

<body>
<h2> Days of the Week</h2>

<?php
//If form not submitted, display form.
if (!isset($_POST['submit' ])){
?>

<form method="post" action="yourscript.php">
Please enter a day of the week: <br />
<input type="text" name="day" />
<p />
<input type="submit" name="submit" value ="Go" />
</form>

<?php
//If form submitted, process input.
}else{
//Retrieve string from post submission
$day = $_POST["day" ];
if ($day == 'Monday' ){
  echo "Laugh on Monday, laugh for danger." ;
} elseif ($day == 'Tuesday' ){
  echo "Laugh on Tuesday, kiss a stranger." ;
} elseif ($day == 'Wednesday' ){
  echo "Laugh on Wednesday, laugh for a letter." ;
} elseif ($day == 'Thursday' ){
  echo "Laugh on Thursday, something better." ;
} elseif ($day == 'Friday' ){
  echo "Laugh on Friday, laugh for sorrow." ;
} elseif ($day == 'Saturday' ){
  echo "Laugh on Saturday, joy tomorrow." ;
} else {
  echo "No information for that day." ;
}

}
?>

</body>
</html>
```

Chapter 10

```php
<?php
//Create array.
$browsers = array(
    "Firefox" ,
    "Chrome" ,
    "Internet Explorer" ,
    "Safari" ,
    "Opera" ,
    "Other"
  );

class Select {
  //Property
  private $name;   //String variable.
  private $value ;  //Array variable.

  //Methods
  //The string set by this method will be the name of the select field.
  public function setName($name){
     $this ->name = $name;
  }

  public function getName(){
     return $this ->name;
  }

  //This method provides the values used for the options
  //in the select field and checks to be sure the value is an array.
  public function setValue ($value ){
     if (!is_array ($value )){
        die ("Error: value is not an array." );
     }
     $this ->value = $value ;
   }

  public function getValue (){
     return $this ->value ;
  }

  //This method creates the actual select options. It is private,
  //since there is no need for it outside the operations of the class.
  private function makeOptions ($value ){
     foreach($value as $v){
        echo "<option value=\" $v\">" .ucfirst ($v). "</option>\n" ;
      }
```

Exercise 1

```
  //This method puts it all together to create the select field.
  publi functio makeSelect (){
    echo "<select name=\"" .$this ->getName(). "\">\n" ;
    //Create options.
    $this ->makeOptions ($this ->getValue ());
    echo "</select>" ;
  }
}//end class

?>


<!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"xml:lang= "en" lang="en">
<head>
<meta http-equiv ="content-type"content ="text/html;charset=iso-8859-1"/>
<title> ClassSelect- Browsers</title>
</head>

<body>
<h2>UserRegistration Browser<br /></h2>

<?php
//If form not submitted, display form.
  if(!isset($_POST['submit' ])){
?>

<form method="post"action ="yourfile.php">
<p>Name:<br />
<input type="text"name="name"size="60" /> </p>
<p>Username:<br />
<input type="text"name="username"size="60" /></p>
<p>Email:<br />
<input type="text"name="email"size="60" /></p>
<p>Browser:


<?php
//Instantiate object.
$browser = new Select();
//Set properties.
$browser ->setName('browser' );
$browser ->setValue ($browsers );
//The object has the data it needs from the preceding commands.
//Tell it to make the select field.
$browser ->makeSelect ();
?>
</p>
<input type="submit"name="submit"value ="Go" />
```

```
<?php
 //If form submitted, process input.
 }else{
  //Could include code to send data to database here.
  //Retrieve user responses.
  $name=$_POST['name'];
  $username=$_POST['username'];
  $email=$_POST['email'];
  //The following variable has an altered name to avoid confusion.
  $selBrowser=$_POST['browser'];
  //Confirm responses to user.
  echc "The following data has been saved for $name: <br />";
  echc "Username: $usernam<br />";
  echc "Email: $emai<br />";
  echc "Browser: $selBrowse<br />";

}
?>

</body>
</html>
```

Exercise 2

```
<?php
//Create arrays.
$browsers = array(
    "None",
    "Firefox" ,
    "Chrome" ,
    "Internet Explorer" ,
    "Safari" ,
    "Opera" ,
    "Other"
  );

$speeds = array(
    "Unknown" ,
    "DSL",
    "T1" ,
    "Cable" ,
    "Dialup" ,
    "Other"
);

$os = array(
    "Windows" ,
    "Linux" ,
    "Macintosh" ,
    "Other"
);

class Select {
  //Properties
  private $name;  //String variable.
  private $value ; //Array variable.

  //Methods
  //The string set by this method will be the name of the select field.
  public function setName($name){
    $this->name = $name;
  }

  public function getName(){
    return $this->name;
  }

  //This method provides the values used for the options
  //in the select field and checks to be sure the value is an array.
  public function setValue($value ){
    if (!is_array ($value )){
      die ("Error: value is not an array." );
    }
    $this->value = $value ;
```

```
  publi functio getValue(){
    returr $this->value;
  }

  //This method creates the actual select options. It is private,
  //since there is no need for it outside the operations of the class.
  privat functio makeOptions($value){
    foreacl($value as $v){
      echc "<option value=\"$v\">" .ucfirst($v). "</option>\n" ;
    }
  }

  //This method puts it all together to create the select field.
  publi functio makeSelect(){
    echc "<select name=\"" .$this->getName(). "\">\n" ;
    //Create options.
    $this->makeOptions($this->getValue());
    echc "</select>" ;
  }
}//end class

?>


<!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"xml:lang= "en" lang="en">
<head>
<meta http-equiv ="content-type"content ="text/html;charset=iso-8859-1"/>
<title> MultipleSelectObjects</title>
</head>

<body>
<h2>UserRegistration<br /></h2>

<?php
//If form not submitted, display form.
  if(!isset($_POST['submit'])){
?>

<form method="post"action="yourfile.php">
<p>Name:<br />
<input type="text"name="name"size="60" />  </p>
<p>Username:<br />
<input type="text"name="username"size="60" /></p>
<p>Email:<br />
<input type="text"name="email"size="60" /></p>
```

```
<p>PrimaryBrowser:


<?php
//Instantiate object.
$browserWork = new Select();
//Set properties.
$browserWork ->setName('browserWork' );
$browserWork ->setValue($browsers );
//The object has the data it needs from the preceding commands.
//Tell it to make the select field.
$browserWork ->makeSelect();
//Destroy the object.
unset($browserWork );

echo "</p>\n<p>Connection Speed: ";
$speedWork = new Select();
$speedWork ->setName('speedWork' );
$speedWork ->setValue($speeds);
$speedWork ->makeSelect();
unset($speedWork );

echo "</p>\n<p>Operating System: ";
$osWork = new Select();
$osWork ->setName('osWork' );
$osWork ->setValue($os);
$osWork ->makeSelect();
unset($osWork );

?>
</p>
<p><strong> HomeAccess</strong></p>
<p>PrimaryBrowser:
<?php
$browserHome = new Select();
$browserHome ->setName('browserHome' );
$browserHome ->setValue($browsers );
$browserHome ->makeSelect();
unset($browserHome );

echo "</p>\n<p>Connection Speed: ";
$speedHome = new Select();
$speedHome->setName('speedHome');
$speedHome->setValue($speeds);
$speedHome->makeSelect();
unset($speedHome);

echo "</p>\n<p>Operating System: ";
$osHome = new Select();
$osHome->setName('osHome');
```

```
$osHome->makeSelect();
unset($osHome);
?>
</p>

<p />
<input type="submit" name="submit" value="Go" />

</form>

<?php
  //If form submitted, process input.
  }else{
    //Could include code to send data to database here.
    //Retrieve user responses.
    $name=$_POST['name'];
    $username=$_POST['username'];
    $email=$_POST['email'];
    $browserWork =$_POST['browserWork' ];
    $speedWork =$_POST['speedWork' ];
    $osWork =$_POST['osWork' ];
    $browserHome =$_POST['browserHome' ];
    $speedHome=$_POST['speedHome'];
    $osHome=$_POST['osHome'];
    //Confirm responses to user.
    echo "<p>The following data has been saved for $name: </p>\n";
    echo "<p>Username: $username<br />\n";
    echo "Email: $email</p>\n";
    echo "<p>Work Access:</p>\n";
    echo "<ul>\n<li> $browserWork</li>\n";
    echo "<li> $speedWork</li>\n";
    echo "<li> $osWork</li>\n</ul>\n";
    echo "<p>Home Access:</p>\n";
    echo "<ul>\n<li> $browserHome</li>\n";
    echo "<li> $speedHome</li>\n";
    echo "<li> $osHome</li>\n</ul>\n";
}
?>

</body>
</html>
```

```php
<?php
//Create  array.
$browsers = array(
    "Firefox" ,
    "Chrome" ,
    "Internet  Explorer" ,
    "Safari" ,
    "Opera" ,
    "Other"
  );

$speeds = array(
    "Unknown" ,
    "DSL" ,
    "T1" ,
    "Cable" ,
    "Dialup" ,
    "Other"
);

$os = array(
    "Windows"  ,
    "Linux" ,
    "Macintosh" ,
    "Other"
);

class Select {
  //Property
  private $name;   //String variable.
  private $value ;  //Array variable.

  //Methods
  //The string set by this method will be the name of the select field.
  public function setName($name){
    $this->name = $name;
  }

  public function getName(){
    return $this->name;
  }

  //This method provides the values used for the options
  //in the select field and checks to be sure the value is an array.
  public function setValue ($value ){
    if (!is_array ($value )){
      die ("Error: value is not an array." );
    }
    $this->value = $value ;
   }

  public function getValue (){
    return $this->value ;
```

Exercise 3

```
  //This method creates the actual select options. It is private,
  //since there is no need for it outside the operations of the class.
  private function makeOptions($value){
    foreach($value as $v){
      echo "<option value=\"$v\">" .ucfirst($v). "</option>\n" ;
    }
  }

  //This method puts it all together to create the select field.
  //This method puts it all together to create the select field.
  public function makeSelect(){
    echo "<select name=\"" .$this->getName(). "\">\n" ;
    //Create options.
    echo "<option value=\"No response\">--Select one--</option>\n" ;
    $this->makeOptions($this->getValue());
    echo "</select>" ;
  }
}//end class

?>


<!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" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1" />
<title>Form Modified and Validated</title>
</head>

<body>
<h2>User Registration<br /></h2>

<?php
//If form not submitted, display form.
  if(!isset($_POST['submit'])){
?>
* Indicates required field.
<form method="post" action="yourfile.php">
<p>*Name:<br />
<input type="text" name="name" size="60" />  </p>
<p>*Username:<br />
<input type="text" name="username" size="60" /></p>
<p>*Email:<br />
<input type="text" name="email" size="60" /></p>

<p><strong>Work Access</strong></p>
<p>Primary Browser:


<?php
//Instantiate object.
$browserWork = new Select();
//Set properties.
$browserWork ->setName('browserWork' );
$browserWork ->setValue($browsers );
//The object has the data it needs from the preceding commands.
//Tell it to make the select field
```

```
//Destroy the object.
unset($browserWork );

echo "</p>\n<p>Connection Speed: ";
$speedWork = new Select();
$speedWork ->setName('speedWork' );
$speedWork ->setValue($speeds);
$speedWork ->makeSelect();
unset($speedWork );

echo "</p>\n<p>Operating System: ";
$osWork = new Select();
$osWork ->setName('osWork' );
$osWork ->setValue($os);
$osWork ->makeSelect();
unset($osWork );

?>
</p>
<p><strong> HomeAccess</strong></p>
<p>PrimaryBrowser:
<?php
$browserHome = new Select();
$browserHome ->setName('browserHome' );
$browserHome ->setValue($browsers );
$browserHome ->makeSelect();
unset($browserHome );

echo "</p>\n<p>Connection Speed: ";
$speedHome = new Select();
$speedHome->setName('speedHome');
$speedHome->setValue($speeds);
$speedHome->makeSelect();
unset($speedHome);

echo "</p>\n<p>Operating System: ";
$osHome = new Select();
$osHome->setName('osHome');
$osHome->setValue($os);
$osHome->makeSelect();
unset($osHome);
?>
</p>

<p />
<input type="submit"name="submit"value="Go" />

</form>

<?php
  //If form submitted, process input.
  }else{
    //Could include code to send data to database here.
    //Retrieve user responses.
    $name=$_POST['name'];
    $username=$_POST['username'];
    $email=$_POST['email'];
    $browserWork =$_POST['browserWork' ];
    $speedWork =$_POST['speedWork' ];
```

```
    $browserHome =$_POST['browserHome' ];
    $speedHome=$_POST['speedHome' ];
    $osHome=$_POST['osHome'];
    //Check input.
    if (empty($name)){
      die('Error: Please enter your name. <br />
      <input type="submit" name="back" value="Back to form"
      onclick="self.location=\'yourfile.php\'"     /></body></html>  ');
    }
    if (empty($username )){
      die('Error: Please choose a username. <br />
      <input type="submit" name="back" value="Back to form"
      onclick="self.location=\'yourfile.php\'"     /> </body></html>  ');
    }
    $char = strpos($email, '@');
    if (empty($email) || $char === false ){
      die('Error: Please enter a valid email address. <br />
      <input type="submit" name="back" value="Back to form"
      onclick="self.location=\'yourfile.php\'"     /> </body></html>  ');
    }
    //Confirm responses to user.
    echo "<p>The following  data has been saved for $name: </p>\n";
    echo "<p>Username: $username<br />\n";
    echo "Email: $email</p>\n";
    echo "<p>Work  Access:</p>\n" ;
    echo "<ul>\n<li> $browserWork</li>\n" ;
    echo "<li> $speedWork</li>\n" ;
    echo "<li> $osWork</li>\n</ul>\n"  ;
    echo "<p>Home Access:</p>\n" ;
    echo "<ul>\n<li> $browserHome</li>\n" ;
    echo "<li> $speedHome</li>\n" ;
    echo "<li> $osHome</li>\n</ul>\n"  ;
}
?>

</body>
</html>
```

Conclusion

This book has found you because you have the ultimate potential.

It may be easy to think and feel that you are limited but the truth is you are more than what you have assumed you are. We have been there. We have been in such a situation: when giving up or settling with what is comfortable feels like the best choice. Luckily, the heart which is the dwelling place for passion has told us otherwise.

It was in 2014 when our team was created. Our compass was this – the dream of coming up with books that can spread knowledge and education about programming. The goal was to reach as many people across the world. For them to learn how to program and in the process, find solutions, perform mathematical calculations, show graphics and images, process and store data and much more. Our whole journey to make such dream come true has been very pivotal in our individual lives. We believe that a dream shared becomes a reality.

We want you to be part of this journey, of this wonderful reality. We want to make learning programming easy and fun for you. In addition, we want to open your eyes to the truth that programming can be a start-off point for more beautiful things in your life.

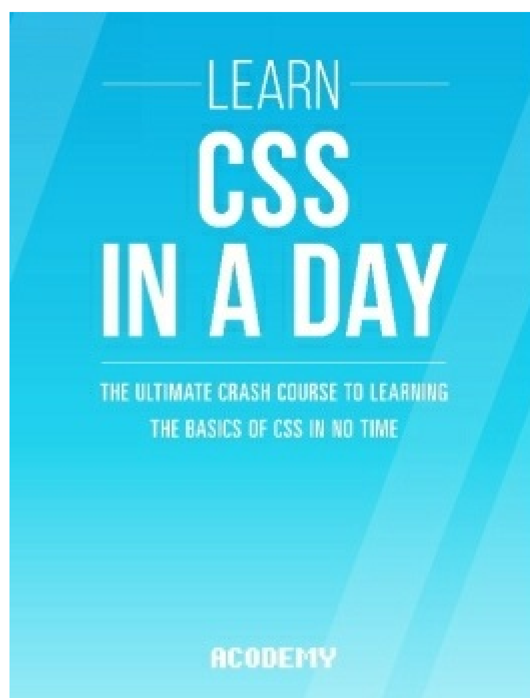
Programming may have this usual stereotype of being too geeky and too stressful. We would like to tell you that nowadays, we enjoy this lifestyle: surf-program-read-write-eat. How amazing is that? If you enjoy this kind of life, we assure you that nothing is impossible and that like us, you can also make programming a stepping stone to unlock your potential to solve problems, maximize solutions, and enjoy the life that you truly deserve.

This book has found you because you are at the brink of everything fantastic!

Thanks for reading!

You can be interested in:

“CSS: Learn CSS In A DAY!”



Here is our full library: http://amzn.to/1HPABQI

To your success,

Acodemy.