

*

A Summary of the
PHP
Programming Language

By

Shyam M. Guthikonda

May 2004

*

CONTENTS

Section 1	Introduction	1
	1.1	Origins of PHP 1
	1.2	Development Environment 2
	1.3	Commenting 2
Section 2	Data Types	3
	2.1	Scalar Types 3
	2.2	Compound Types 4
	2.3	Special Types 6
	2.4	Constants 6
	2.5	Scope 6
Section 3	Expressions and Control	8
	3.1	Operators and Precedence 8
	3.2	Control Structures 10
Section 4	Functions	13
	4.1	Built-In 14
	4.2	User-Defined 15
Section 5	Files	16
	5.1	File I/O 16
	5.2	Parsing and Pattern Matching 17
	5.3	Sessions and Cookies 19
Section 6	Databases	20
	6.1	MySQL 20
	6.2	Database Operations 21
	Conclusion	23
	References	24

SECTION 1 – Introduction

Section 1.1 – Origins of PHP:

PHP is an HTML-embedded server side scripting language. It is purely interpreted, in that it doesn't need to be compiled. A PHP file is run through an interpreter, which produces the corresponding HTML code for the browser to display.

Here is an example of what PHP code looks like. It is contained in a file with the extension, .php. HTML, JavaScript, etc. can be embedded anywhere around it (outside of the tags).

```
<?php printf("Hello World."); ?>
```

PHP was developed in 1995 by Rasmus Lerdorf. He created this language to track the number of people who viewed his online resumé. The only other alternative at the time was Perl, and his computer didn't have the RAM or CPU available to run a CGI Script (a small program that is run on demand to produce the content of the webpage) every time his page was viewed.

The acronym PHP originally stood for 'Personal Home Page Tools', but nowadays it is known to mean 'HyperText Preprocessor'. The first couple versions of PHP (known as PHP/FI – Personal Home Page / Forms Interpreter) were written almost entirely by Lerdorf. He released the code as open source so that others could help in the development process. However, most of the work was done by just him. In 1997, the first real 'fully powered' version of PHP was released by Lerdorf, Andi Gutmans and Zeev Suraski. It was known as PHP 3.

Currently, PHP 4 is the most widely used version, with PHP 5 in development (some Beta versions are already out). PHP 4 was based on the Zend Engine (derived from **Zeev** and **Andi**). Zend is a scripting engine that sits below the PHP-specific modules. It is optimized to significantly improve performance (Zandstra 2002, p. 10). PHP was running on more than 1 million hosts in November 1999. This number jumped to 6 million in September 2001. Also, the number of PHP sites running on Windows has doubled in the past year. It is expected to become to most popular non-Microsoft scripting language used on Windows by the end of 2004. (Currently it is in 2nd place, behind Cold Fusion) (<http://www.netcraft.com>).

Section 1.2 – Development Environment:

Due to its pure interpretation implementation, PHP code does not need to be compiled. This can prove to be beneficial in that no special compilers are required. An entire PHP driven system can be written completely in Notepad. However, this is not necessarily recommended. A good text editor used to create PHP driven websites is Crimson Editor (<http://www.crimsoneditor.com>). It is a free text editor that supports hundreds of languages. There are advantages to using a specialized text editor. They provide many special features that you cannot get in Notepad, including the color-coding of special keywords, and the highlighting of matching brackets.

Although PHP does not require a compiler, it does require a server with an interpreter installed (hence the name, server side scripting language). Most servers nowadays have PHP installed, so you can simply upload your .php files and they will work. It is often an inconvenience to upload a file every time a change is made, though. This is where a personal Apache web server comes in handy. There is a very nice tool available for download called XAMPP (<http://www.apachefriends.org/xampp-en.html>). This is a free program that automatically installs and configures numerous accessories needed for developing PHP applications on a local computer. From PHP to an Apache server to MySQL to phpMyAdmin - this utility has everything needed to get started in a matter of minutes.

Section 1.3 – Commenting:

Commenting in any programming language is essential. They allow for quick and easy documentation so that other programmers can more easily understand the code. PHP supports C/C++ and UNIX shell-style commenting.

```
// this is a one-line comment
/* this is a multi-line comment...
    so is this... */
# this is a one-line comment
```

Overuse of comments is generally considered a safer practice than under-use of them. They can greatly add to the readability of the code, helping even the originating programmer understand what they wrote at a later date.

SECTION 2 – Data Types

Section 2.1 – Scalar Types:

Although PHP started out as a simple parser, it has grown into a full-fledged programming language. To meet these demands, PHP is required to have the standard scalar (single value) data types. These include: Boolean, integer, float, and string.

2.1.1 – *Boolean*:

Boolean values are mostly the same in PHP as they are in other languages. A Boolean variable has the value TRUE or FALSE. If you try to print a Boolean variable, though, the browser will display 1 for TRUE and 0 for false. If you try to add a numeric value to a Boolean, it will do so successfully, but treating the Boolean as either 1 or 0 for TRUE or FALSE, respectively. Boolean values are much like C/C++ in that any value but 0 is considered TRUE, where 0 is FALSE. A Boolean declaration may look like `$someBool = TRUE;`

2.1.2 – *Integer*:

A PHP integer is any non-decimal number, under 32 signed bits (PHP does not support unsigned integers). The standard arithmetic can be performed on integers. If an integer is divided by a number and a remainder occurs, the result will be turned into a floating point number unless explicitly cast to an integer. Integers can be declared in base 8, 10 or 16. An example integer declaration is `$someInt = 32;`

2.1.3 – *Float*:

PHP does not have explicit double or real types like many other languages (Fortran, C, Java, etc.). Every number with a decimal is considered a floating-point number. The range of the PHP float type is 64 bits (much like a other languages corresponding double). Programmers must be careful when using floats, as the same round-off errors and impossibility of accurately representing decimal digits in a finite number of binary digits (Sebesta 2004, p. 236) can arise as they do in other languages. An example of a float is `$someFloat = 16.2311135;`

2.1.4 – *String*:

Strings in PHP are very flexible. They are simply a collection of characters (ASCII). A

string is declared with quotes around it, such as `$someString = "Hello"`; They can be compared, and are considered equal if all of the characters are the same in both strings. PHP supplies many built-in functions that allow string manipulation (discussed later).

The one real oddity with PHP strings is that `$aString = "3"`; is a string, yet it can be added to any other number and result in a number. If the value `"3h"` were assigned instead, it would be treated the same way. (You can tell that PHP still thinks it's a string through the `gettype()` function). For example,

```
$var1 = "333abcdefg";           // $var1 is a string
$temp = 42 + $var1;             // add 42 to $var1
printf("\$temp = %s", $temp);   // prints: $temp = 375
```

If the character appears first, though, such as `$var1 = "abcdefg333"`; this value is considered a string and is treated as such. Adding a number to a string will simply echo the original number value.

Any particular character in a PHP string can be accessed through treating the string as an array. To get the "b" in `$aString = "abc"`; use `$aString{1}`; This is very similar to C's treatment of strings as arrays of chars, except PHP only has the string type.

Section 2.2 – Compound Types:

When data becomes more and more complex, it's often necessary to have a data type that properly reflects this complexity so that the programmer can more easily maintain it. The two types used in PHP are arrays and objects.

2.2.1 – *Array:*

In PHP, an array is defined as an ordered map. A map is a type that maps *values* to *keys* (Bakken 2004). The "key" is the index number and the value is the value assigned to that particular key.

Arrays can be defined in many ways. `$users[] = "Bob"`; will assign the value "Bob" to the key, 0. If no key is explicitly stated inside the brackets, PHP defaults to 0, and will automatically increment it each time a new value is supplied. Another way to define an array is to explicitly state the key: `$users[200] = "Joe"`; This assigns "Joe" to the 201st entry of the `$users` array.

PHP also makes use of associative arrays. This is when the key of the array is a string (this has similarities with a C++ struct). A practical use of an associative array is to keep track of an employee's information.

```
$employee["name"] = "Sarah";  
$employee["age"] = 30;  
$employee["salary"] = 60000;
```

PHP utilizes a special function to help create arrays, `array()`. The parameters are they key and the value to be assigned to that key. (The key is optional. If it's not specified, it simply increments itself).

```
$employee = array( "name" => "Bob",  
                  "age" => 30,  
                  "salary" => 50000);
```

PHP has multi-dimensional arrays as well. This can lead to some very interesting data structures. A two-dimensional array with the first subscript as an index number, and the second subscript as an associative array can store an entire roster of employees. To print all of the employees, simply use the `foreach()` function.

```
foreach($employee as $val) {  
    foreach($val as $key=>$final_val) {  
        printf("%s: %s", $key, $final_val);  
    }  
}
```

This cycles through the `$employee` array, storing each associative array in `$val`. Then it loops through `$val`, storing each key in `$key` and each value in `$final_val`, and prints their values.

2.2.2 - *Object*:

Objects in PHP are like C++ classes. They can contain member functions and variables. They are considered user defined data types. Variables can be defined as objects and they will inherit all of the properties that the object has. An example object is a tree. A tree has various properties: height, width, type, color, etc. An object of type tree can be created with these properties. Then, a variable can be declared to be a tree. `$aTree = new tree;` Now `$aTree` has the properties defined in the tree object.

Perhaps the greatest benefit to objects and object-oriented programming is reusability. Because the classes used to create objects are self-enclosed, they can be easily moved from one project to another (Zandstra 2002, p. 120). Objects can increase the portability of the code, but they can often just add unnecessary complexity and are not typically utilized in basic PHP programming. PHP 5, which is still not fully released, is said to

add many new features to the area of objects in PHP, so this view may change.

Section 2.3 – Special Types:

PHP has two special types: NULL and resource. In a sentence, if NULL is assigned to a variable, that variable is said to have no value. Functions such as `isnull()` can be used to test if the variable has a value assigned to it. If it doesn't, then one can be assigned to it.

The resource data type of PHP is truly unique. A resource is effectively a reference to a third party resource (Zandstra 2002, p. 42). The key example of a third party resource is a database. To access a database, you must first connect to it. This "connection" is returned and stored in a resource. Because of its abstract nature, it is hard to define exactly what a PHP resource truly is. More examples and practical uses of them will be shown later in the database section.

Section 2.4 – Constants:

A constant is a variable that does not change its value from the time it is declared to the time it no longer exists. PHP constants have a global scope. They can be referred to anywhere in the script without explicitly stating `global` (discussed later). To create a constant in PHP, the `define()` function is used. This function takes two required parameters, and a third optional. The first is the name of the constant, and the second is the value of the constant. The third is a Boolean value. This Boolean, if true, says that the constant can be referred to without worrying about case. If this third parameter is left out, it defaults to false, and the constant must be referred to with exact same name as it was defined. An example of a constant is

```
define("USER", "Susan");  
echo "Welcome, ".USER;           // prints Welcome, Susan
```

PHP includes some pre-defined constants as well. `__LINE__` returns the current line number of the file. `__FILE__` returns the full path and filename of the file. Constants can greatly aid a program through readability, as well as making code highly portable.

Section 2.5 – Scope:

The scope of a variable is the range of statements in which the variable is visible. A variable is visible in a statement if it can be referenced in that statement (Sebesta 2004, p. 211). The variable scope in a scripting language such as PHP is a little different concept than one such as C++. In compiled languages, the longest a variable can last is from the start of

program execution, until the program is turned off. With PHP, a variable can last from the moment a certain web site is loaded, until the browser is closed (or even for years at a time while the computer is shut off through the use of cookies, which are discussed later).

Section 2.5.1 – Local Variables:

Most PHP variables are considered to be local. Locality can be viewed in a couple different ways. A variable could be local to a particular page where it is defined, main.php. Or, a variable could be local to a particular function/block on a particular page where it is defined. In a function (which are described later), the only variables that it has access to are variables defined within itself, or variables passed in as parameters. Values outside the function are not known. For a simple page with no functions, a variable defined anywhere on the page (even within include files) are known at any location within the file. However, if a file top.php defines a variable `$a`, and it is included in both index.php and main.php, `$a` will be a “different” in index.php when compared to main.php, since they are both effectively different instances of top.php and therefore different instances of `$a`.

Section 2.5.2 – Global Variables:

To solve the previously mentioned problem of local variables not being defined in functions, global variables are used. A variable defined outside of a function can be referenced by the function if the keyword `global` is used.

```
$x = 7;
$y = 3;
function add_them() {
    global $x, $y;
    $y = $x + $y;
}
```

Since `$x` and `$y` are both declared global within the function, they now have access to the previously defined values. After the function call, `$y` will contain the value 10.

This has advantages and disadvantages. It definitely adds flexibility to the language. Functions can manipulate variables defined anywhere in the code, increasing writability. This could also be a bad thing, though. It can decrease readability and reliability. A function can mistakenly change the value of a globally defined variable. Debugging will then become increasingly difficult, since the error can occur anywhere in code, not just the section where the variable was originally defined.

Although global variables solve the problem of allowing functions to access variables outside of them without passing parameters, they do not solve the problem of allowing a

totally separate page to have access to the variables on another page. This is where the next type of scope comes into play.

Section 2.5.3 – Super-Global Variables:

These variable types are pre-defined. They are available at anytime, anywhere, on any page on a particular server. Some examples of super-global variables are `$PHP_SELF`, `$_POST`, and `$_SESSION`. The latter two are associative arrays. One very common use of the `$_SESSION` variable is for a login script. After a user logs in, their username can be stored in the `$_SESSION['username']` field. Then, to determine what pages the user sees, the function `isset()` can be used to see if the `$_SESSION['username']` variable is set. If it is set, the user must be logged in. If it isn't set, the user is not logged in. The `$_SESSION` super-global will remain until the user closes the browser, or the `session_destroy` function is called. That is a key role for super-global variables, and it is a large part of what makes them so integral to the PHP language.

PHP supports casting from type to type. This is the process of changing the type of a variable to that of a different type. For example,

```
$x = 5.3;  
$y = (int)$x;
```

This will turn the float `$x` into the integer version of itself, `$y`. `$y` will contain the value 5. Since it is an integer, the decimal portion of `$x` is simply dropped off.

SECTION 3 – Expressions, and Control

Section 3.1 – Operators and Precedence:

PHP is an expression-oriented language, in the sense that almost everything is an expression (Bakken 2004). To create these expressions, a language needs basic operators that act on operands. PHP is no different than almost every other language when it comes to these operators. Arguably the most important operator in an imperative language is the assignment operator, `=`. The assignment operator is used to assign and change values of variables. Without this feature, imperative languages could not exist.

PHP has the basic arithmetic operators that are effective on numeric data types: + - * / %, for addition, subtraction, multiplication, division and modulus, respectively. For strings, there is the concatenation operator (.). This is used to attach a string to the end of another string. For example,

```
$aString = "hello"." world";
```

`$aString` now contains the string "hello world". This is an extremely useful operation, since PHP deals with strings so often.

Much like C/C++, PHP includes combined assignment operators, which aid the programmer in writability. Assignments such as `$x = $x + 5;` occur so frequently, that it is very convenient to have an operator that automatically does the operation. For the previous expression, the combined assignment operator would be `$x += 5;` Other "shorthand" operations are defined for all the arithmetic operations and the concatenation symbol by replacing the addition symbol with the appropriate operator.

Another necessity of programming languages are the comparison operators. In PHP, these are `==`, `!=`, `>`, `>=`, `<`, and `<=` for equivalence, non-equivalence, greater than, greater than or equal to, less than, and less than or equal to. These are standard in most other languages, except, PHP has one additional one: `===` (and it's counterpart, `!==`). This is the "identical" operator. The difference between `==` and `===` is that `==` merely tests to see if the two operands being compared are equivalent. `===` tests to see if the two operands are equivalent *and* are of the same type.

```
$x = 5.0;
$y = 5;
($x == $y)           // evaluates to true
($x === $y)         // evaluates to false
```

The expression using the equivalence operator performs a casting operation, converting the two variables into the same type temporarily, and therefore evaluates to true. The expression using the identical operator evaluates to false, since `$x` is a float, while `$y` is an integer and no coercion is performed.

Lastly, are the logical operators. These are identical to those of C/C++: `||`, `&&`, `!`, *or*, *and*, *xor*. These stand for or, and, not, and exclusive or. `||` means one must be true. `&&` means both must be true. *xor* means one must be true but not both.

With all of these operators, confusion can arise when they are put into a single expression, such as `3 + 4 * 7`. Is multiplication or addition done first? This precedence will

determine the result. The following is the official list of precedence as stated in the PHP Manual (Bakken 2004). Higher precedence is indicated by a position closer to the top of the table.

Associativity	Operators
non-associative	new
right	[
right	! ~ ++ -- (int) (float) (string) (array) (object) @
left	* / %
left	+ - .
left	<< >>
non-associative	< <= > >=
non-associative	== != === !==
left	&
left	^
left	
left	&&
left	
left	? :
right	= += -= *= /= .= %= &= = ^= <<= >>=
right	print
left	and
left	xor
left	or
left	,

Associativity is the way that the expression evaluates (from left to right, or from right to left) if it's surrounding operators are of the same precedence level. With this list, the meaning of an expression can be determined without any uncertainty.

Section 3.2 – Control Structures:

Selection and iteration are necessary tools for any programming language, as proven by Böhm and Jacopini (Sebesta 2004, p. 320). They provide power and flexibility. PHP, being a descendent of Perl, and therefore of C, inherits the exact same iteration statements as its ancestor (with one exception). These statements include: **if-else**, **else-if**, **switch**, **while-do**, **do-while**, **for** and **foreach** (which does not exist in C). The semantics of these statements are identical to those of C. The syntax is nearly identical, except for the variable

declaration methods that PHP uses. Also, the control expression can be Boolean or arithmetic. It is not limited to Boolean like Java. Any value other than 0 is considered true, while 0 is considered false. Lastly, a variable declared within a control structure is visible anywhere outside of the block (unlike C).

Section 3.2.1 - *Selection*:

A PHP `if-else` statement is easily understood. An example follows:

```
if ($x == 3) {
    // do something
} else if ($x == 5) {
    // do something
} else {
    // do something
}
```

If `$x == 3`, the code in the first block is executed; if not, it goes on to check the next condition. If `$x == 5`, the second code block is executed. If neither of these is true, the `else`, code block is executed. It is helpful to point out that the curly-brackets are *not* necessary, as they are in Perl, if the statement is not compound. If there is a compound statement, however, they are necessary. Also, the `else-if` and `else` statements are not necessary. They are just shown in the example to demonstrate their existence.

Any sort of selection can be done using the above statements, but it is often easier to use a `switch` statement when there are numerous possible choices. This aids readability. An example of the switch statement with the above example follows:

```
switch ($x) {
    case 3:
        // do something
        break;
    case 5:
        // do something
        break;
    default:
        // do something
}
```

This form will be much more readable if there are even more possible values of `$x`. The value of `$x` is first evaluated. Then, the program looks to match this value to one of the case values. If it matches one, it executes the corresponding code. If none of the cases match the actual value of `$x`, the default statement is executed. `break` statements are used so that the flow of the program will not continue to the next case (although they are not required, they

are almost always used). There is one difference in the PHP `switch` statement from that of C/C++. In C/C++, curly-brackets around the statements of a case are not necessary *unless* a variable is defined within the code. In PHP, curly-brackets are never necessary; with or without variable declaration in a case, it doesn't matter.

Section 3.2.2 – *Iteration*:

For all iterations, the `while-do` loop is almost always sufficient. Adding more loops, while marginally increasing the complexity of the language, can greatly aid readability and writability. The `while-do` loop is a pretest loop.

```
while ($condition) {
    // do something
}
```

While a given condition is true (or not 0), execute the code. If the condition is some numeric variable being used as a counter, the `while-do` will not automatically increment it. This must be done by the programmer. This is a feature that makes the `for` loop so appealing, as discussed later.

The `do-while` loop is very similar to the `while-do`, except it is a post-test loop. The expression is not evaluated until after the loop, which means the loop will always execute at least once.

```
do {
    // do something
} while ($x);
```

This code will execute at least one time, and continue executing while `$x` is true.

As stated earlier, the `for` statement can be constructed with simple `while-do` loops, but it can get very unreadable. The basic syntax is:

```
for ($x = 0; $x < 10; x++) {
    // do something
}
```

This means: Initiate `$x` to 0. While `$x` is less than 10, execute the code. Then each time through, increment `$x` by 1. The integration of the `while-do` can be seen here. The `for` statement is often easier to read, however. Also, it is nice that the counter variable can be incremented in the initialization of the loop, so that it won't be forgotten. Although the three parameters of the `for` loop are all technically optional, it is almost always most efficient to

use them instead of manually testing and/or incrementing within the code body.

Lastly, PHP has a useful looping mechanism (for arrays *only*) called the **foreach** statement. This statement looks like this:

```
foreach($array as $key => $value) {  
    // do something  
}
```

This structure will reset the pointer to the array to the first element. It will cycle through the array, storing the key (index number) in **\$key**, and the value at the given key in **\$value**. The '**\$key =>**' feature is optional, but it serves to demonstrate. This statement is invaluable, and would make operations such as looping through associative arrays extremely difficult without it. C does not include this statement, mainly because it does not have associative arrays. It can loop through arrays easily with the **for** statement, since its keys (index numbers) are only integers.

A few other general features of control structures that are often necessary are the **break** and **continue** statements. The use of **break** was shown in the **switch** statement example. If put in a loop, **break** will halt execution of the loop and the program will jump to the first line following the end of the loop. **continue** is a nice feature that is not included with C. When used in a loop, **continue** will skip the current iteration, and jump to the next iteration (remaining in the loop). This can be useful to test if there will be division by 0. If there is division by 0, go to the next iteration. If there are more important iterations of the loop after the division by 0, **continue** will be a much more viable option over **break**.

Lastly, PHP supports nested loops like C. Selection statements can exist within selection statements existing in loops existing in loops existing in selection statements! Although, the recommended level of nesting is generally considered no more than two levels – for readability sake.

SECTION 4 – Functions

Most, if not all, programming languages contain a type of function or subprogram. Functions increase the readability, writability, and reusability of programs. A frequently

done task can be implemented as a function, and whenever that task needs to be done, the function can simply be called. If the programmer wants to optimize the function, the code just needs to be edited in the one place where the actual function body is. Functions also allow reusability in code. A function can be written, and its purpose can be documented with comments. The same function can be utilized at a later time, and the programmer does not need to have any knowledge about how it works. Although functions are not necessary to programming languages, life would be exponentially more difficult without them.

Function calls are exactly like in C/C++. They are in the form: `function_name (<parameters>)`. Function declaration is different than that of C/C++, however, and it will be discussed later.

Section 4.1 – Built-In:

PHP provides an extensive set of built-in functions. Built-in refers to the fact that these functions are pre-defined within the PHP language – the programmer doesn't construct them; they simply call them. A few extremely useful built-in functions that PHP provides are `stripslashes/add`, `preg_match`, `include_once`, `printf`, and `header`.

`stripslashes` and `addslashes` are self-explanatory, although their use may not be apparent. When a user enters data in a form and submits it, depending on the server configuration, backslash marks, `\`, may or may not be added to the beginning and end of the entry, as well as to the spot preceding any apostrophes or quotation marks. If the data needs to be compared to a constant string, this can lead to problems. These two functions allow the backslashes added by the server configuration to be stripped or added so that they can be properly compared to other strings.

The `preg_match` function is another invaluable built-in function. This function allows for one form of pattern matching, which will be explained in-depth later.

`include`, specifically `include_once`, allows files to be 'global.' To connect to a database, a connection needs to be made. The connection code could be typed out on each page, but if for some reason the password changes, the code will need to be changed on every page. If there are numerous pages, this can be a daunting chore. `include_once` allows the programmer to greatly simplify this task. A file called `connect.inc` can contain the information on how to connect to a database. Then, on each page, the line `include_once ("connect.inc")` can be called. The connection will be automatically made. If any connection information changes, only the code on the one file, `connect.inc`, will need to be changed. `include_once` differs from `include` in that it only allows a file to be included *once*. If the call to include a file that connected to a database happened to be in a loop (for some reason), this would cause strange effects if numerous connections are made. `include_once`

makes sure that the call only happens once.

`printf` is the standard output function. There are others, but this one is especially useful because of the way variables can be displayed. It has a form similar to the corresponding C function. This function was used in earlier examples, but here is a simpler one demonstrating how variables can be displayed to the screen.

```
$x = 3;
$y = 4;
printf("x=%s, y=%s", $x, $y);      // displays: x=3, y=4
```

This function only requires one parameter (a string), but it can have any amount of parameters greater than one, as long as there are equal amount of % symbols. The % means that a variable (passed as a parameter) will be placed here. The 's' means that it will be treated as a string.

`header` is an almost essential function. Although it has other uses, the most commonly used is its feature of redirection. When called, this function sends a raw HTTP header. When used with the 'Location:' keyword, header will send a 302 status code (REDIRECT) to the browser. An example:

```
header("Location: http://www.anyurl.com?id=9");
exit;
```

There are a couple things to note in this example. When the program reaches this line of code, the user will be redirected to the URL given. The `exit` keyword is used so that the code doesn't continue execution upon redirection (which can lead to some problems that become extremely difficult to debug). After being taken to the new URL, the variable `$id` will be equal to 9. This is a very useful way to pass variables to a new page, so that the page can be dynamically generated. Multiple variables can be passed by separating them by &.

Section 4.2 – User-Defined:

Often times the pre-defined functions supplied by a language are not enough. Like any other HOL, PHP allows the user to create their own functions.

```
function sum($a, $b) {
    $c = $a + $b;
    return $c;
}
```

This is the general syntax of a function declaration. This function would be called with the following form: `sum(3, 4)`, which would return 7. PHP functions are different from C/C++ in a few ways, but their general features remain the same. PHP requires all user-defined functions to have the `function` keyword preceding them. This alerts the interpreter of an oncoming function declaration. Then follows the name of the function, and its formal parameters. PHP does not require a return type to be specified like C does. Also, the types of the parameters do not need to be specified. This allows for much more generic code; but with this flexibility, debugging can become more difficult. Any variable declared within a function is local to that function body. Also, functions cannot access variables defined outside of their body, unless the `global` keyword is used. Any value can be returned in a PHP function; including arrays, resources and objects. Function declarations can be nested within one another, unlike C. For example:

```
function a() {
    // do something
    function b() {
        // do something
    }
}
```

The interesting thing to note here is that the function `b` will not be defined unless `a` is first called. If `b` is called before `a`, there will be an error.

SECTION 5 – Files

Section 5.1 – File I/O:

File I/O is an integral part to any HOL. Without files, data manipulation would be a complex and inefficient task – executable files would become enormous, since the data would have to be stored in the actual code. For file I/O to exist, a language must have some built-in functions that allow the creation and manipulation of said files. Some of the functions PHP includes for this task are `touch`, `fopen`, `fgetc`, `fputs`, and `feof`.

When called, `touch("filename.txt")`, will simply create the file specified if it does not already exist. Data can then be entered into the file. If the file already exists and has data in it, however, `fopen("filename.txt", "r")` can be used instead. This will open the specified file, and return a resource data type. The second parameter, `'r'`, is the operation that will be done on the file. This can be `'r'` for reading, `'w'` for writing, or `'a'` for appending.

Once the file is opened, and the resource to it is returned, data can be extracted. An example follows:

```
$fp = fopen("filename.txt", "r") or die("Couldn't open file");
while(!eof($fp)) {
    $char = fgetc($fp);
    printf("%s", $char);
}
```

This demonstrates the use of the previously mentioned functions. First, the file is opened for reading, and the resource is returned to `$fp`. If the file is unable to open, the script will end using the `die` function. The `while` loop will make use of the `eof` function. This function will return true when the end of file character has been reached, and the `while` loop will halt execution. Inside the loop, `fgetc` is used to access a character from the file resource. It will return one character, which will be stored in the variable `$char`. Then, the value of `$char` will simply be printed to the screen using the `printf` function. This is a simple example that probably wouldn't have much use in a real application. Yet, from these simple functions, very complex parsing routines can be created to perform very practical tasks.

Section 5.2 – Parsing and Pattern Matching:

Pattern matching, which takes the form of regular expressions, is an invaluable tool. It was a key feature in the Perl language, and that is one of the reasons it is included in PHP. There are numerous built-in functions that PHP includes for pattern matching. One of the most commonly used is the `preg_match` function, which stands for "Perform Regular Expression Match." This function has the capability to take numerous parameters, but it only requires two: `preg_match(<string_pattern>, <string_subject>)`. The first parameter is the pattern that will be searched for, and the second parameter is the string where the pattern will be searched for. `preg_match` will return an integer, 0 or 1. 0 if the pattern is not found, and 1 if the pattern is found. Even if there are multiple matches of the pattern, 1 will still be returned because the function returns on the first occurrence of the given pattern.

There are a few basic quantifiers that need to be understood before looking at a regular expression, since they are an integral part of pattern matching. (Table from – Zandstra 2002, p. 343).

Symbol	Description	Example
*	Zero or more instances	a*
+	One or more instances	a+
?	Zero or one instance	a?
{ <i>n</i> }	<i>n</i> instances	a{3}
{ <i>n</i> ,}	At least <i>n</i> instances	a{3,}
{, <i>n</i> }	Up to <i>n</i> instances	a{,2}
{ <i>n1</i> , <i>n2</i> }	At least <i>n1</i> instances, no more than <i>n2</i> instances	a{1,2}

Here are a few practical examples of the `preg_match` function.

```
preg_match("/.*@.*\..*/", $_POST['email']);
preg_match("/[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+/", $ip);
```

Note, the `preg_match` function requires `/'` to precede and follow the pattern string. This designates to the interpreter that the characters in between are a pattern and have special meanings, not to be confused with a basic string.

The first example will search the super-global `$_POST` variable (a user-entered e-mail), and make sure it is a valid e-mail address. `.'` signifies any single character; `'@'` signifies the @ symbol; followed by any single character; followed by a `.'` (where `\` means to take the next character as its literal meaning), and finally any single character. This is the basic structure of an e-mail: `someone@someplace.com`

The second example makes use of brackets to signify a range. It will attempt to verify that a given IP address, `$ip`, is valid. The range `[0-9]+` signifies the need for at least one digit, followed by a `.'`, followed by at least one digit, etc. This is the basic structure of an IP address: `136.227.68.17`

These are just some of the most basic instances of pattern matching in PHP. It is a pretty straightforward, but powerful, concept.

Section 5.3 – Sessions and Cookies:

Section 5.3.1 – *Sessions*:

`$_SESSION` is one of PHP's super-global variables. The basic meaning of a session is just like its English definition: "A period of time devoted to a specific activity"

(<http://www.dictionary.com>). When a user logs on to a web site, their session can be set using the `session_start` function. This activates the `$_SESSION` super-global. Since it is an associative array, values can now be assigned to its various fields, such as a username and a password. The various fields of `$_SESSION` will remain set until the user closes the browser window, or the `session_destroy` function is called. The use of sessions is apparent. It provides a simple means of temporarily storing user data while they are visiting a web site. Its main disadvantage is that the information is lost when the user closes the browser window. This problem can be solved by using cookies.

Section 5.3.2 – Cookies:

What is a cookie? There are several “theories” as to where the term originated, but they are all just that: theories.

- In UNIX, files of this type had a name something like kook.z. Thus the strange pronunciation as *cookie* (uh-huh...).
- Because distributing cookies is like leaving crumbs all over (uh...)
- Just because.
- It’s a very old bit of programmer slang (usually) for a piece of data stored to communicate between two processes, typically separated in time, and often as some kind of flag. The fuller form is “magic cookie.” I know I’ve heard it used as far back as the late ‘80s anyway. The real defining point of the magic cookie is that some part of the data is unique to the process(es) so that the receiver can ensure that it’s getting the message from the expected source... I’ve heard that the term “magic cookie” was originally coined in reference to one of those old adventure games (perhaps “Adventure” itself) in which you had to give some character a magic cookie to get something from it (analogous to cookie verification)...

(Burns 1998, p. 422).

Specifically, though, a cookie is a small text file, 4 KB or less. It can store data about a user on their system. A standard cookie consists of a name, value, expiration date, and host and path information. After a cookie is set, only the originating host can read the data (to ensure user privacy). Cookies should be used sparingly. Many users have their browsers set to block incoming cookies, so they should not be an integral part of any web site.

PHP includes a special function to create a cookie. `setcookie(<name>, <value>, <expiration>)` is the simplest form of the function (there are numerous other parameters but they are outside the scope of this paper). Here is an example of the function in action:

```
setcookie("usercookie", $_SESSION['username'], time()+60*60*24*7);
```

This function will set the super-global `$_COOKIE['usercookie']` field to the value of `$_SESSION['username']`, effectively storing their username in a text file on the users system. The third parameter will get the current time, and do some basic mathematical operations on it, in order to make the cookie exist for 1 week. If the user closes their browser, or even shuts of their computer, when they return to the web site, the cookie will still exist (if the 1 week time frame hasn't expired). The `isset` function can be used to see if the 'usercookie' field of `$_COOKIE` is set. If it is set, that means the cookie still exists, and their information can be read and stored temporarily in the `$_SESSION` super-global.

SECTION 6 – Databases

Often times, when vast amount of information needs to be stored dynamically on the internet, files can become too primitive to be of much use. This is where databases come into play. A database is simply a place to store data that can be added to, modified, and deleted at any time. It can store any type of information from sports scores to employee salaries to procedures on how to make a space voyage.

Section 6.1 – MySQL:

There are many different database management systems that are utilized around the world. One of the most widely used is called MySQL – SQL meaning "Structured Query Language", and My being the name of one of the co-founder's daughters (<http://dev.mysql.com/doc/mysql/en/History.html>). Some reasons for MySQL's success in the database world is that it is open source (it's free). It is extremely fast and efficient. It is almost entirely platform independent, running on Windows, UNIX, and more than 20 other platforms.

Another key feature that makes it so appealing to PHP programmers is that PHP has numerous built in MySQL functions. A utility written entirely in PHP, phpMyAdmin (<http://www.phpmyadmin.net>), allows database administration through an entirely graphical process instead of through a command line. These characteristics make MySQL the database management system of choice for PHP users.

`mysql_query` is one of the most crucial functions for database operations. It is used

for reading, deleting, adding, replacing, and updating database fields, specified through the keywords SELECT, DELETE, INSERT, REPLACE, and UPDATE, respectively.

Section 6.2 – Database Operations

Section 6.2.1 – *Connecting to a Database:*

In order to do anything with a database, a connection must first be established. The PHP function that allows this is `mysql_connect`. This function usually takes three parameters. An example follows:

```
$host = "localhost";  
$user = "root";  
$pswd = "secret";  
$link = mysql_connect($host, $user, $pswd);
```

The host is almost always localhost. This is the server that the page will be running on. The user and pswd are the username and password required to log in to the particular host. When `mysql_connect` is called, a resource is created and returned to the `$link` variable.

Now that a connection is established, a database must be selected. The `mysql_select_db` function allows for this operation. This function can take two parameters. The first one is the name of the database to select, and the second is a link (resource) to the server with that database on it.

```
$db = "a_db";  
mysql_select_db($db, $link); // assumes link has already been  
                             // established with mysql_connect
```

Now that the database connection is good to go, operations can be performed on the data contained within.

Section 6.2.2 – *Adding to a Database:*

Once a connection has been established, information can be added to a database. This can be done through phpMyAdmin, but it is not usually a good idea to give every visitor to a web site database administration powers. The standard way to add user-submitted data into a database is through a simple HTML form. When the user fills out the form and presses "submit", the super-global variable `$_POST['submit']` field becomes set, which can be

tested by using `isset`. Adding to a database is very simple.

```
mysql_query("INSERT INTO a_table (field1,
                                field2)
            VALUES ('".$_POST['field1']."',
                    '".$_POST['field2']."')
            ");
```

The above code will simply add (insert) into the specified database table (`a_table`), the user-submitted information from the form into the respective fields. This example requires there to be fields within the `a_table` table with the names `field1` and `field2`.

Section 6.2.2 – Reading from a Database:

Now that there is information in a database, that information can be displayed or read. This operation is very similar to the insert operation, except for one added complication, made simple with the `mysql_fetch_array` function.

```
$result = mysql_query("SELECT * FROM a_table");
$entry = mysql_fetch_array($result);
// loop to display $entry['field1'], $entry['field2'], etc.
```

First, the `mysql_query` function is used to SELECT every field and every entry, `*`, from the table, `a_table`. A resource data type is returned, which is stored in `$result`. The entries are stored as a queue (meaning first-in first-out). The oldest entries will be “on top” and the newest “on bottom”, unless otherwise specified. So, when the results are accessed, the first result will be the oldest entry. After the data has been retrieved from the table, it needs to be accessed one entry at a time. `mysql_fetch_array` will get the entry on the top of the stack, and return it to be stored as an associative array in `$entry`. Now, `$entry` can simply be treated as an array, such as `$entry['field1']`, etc. After its contents have been displayed, `mysql_fetch_array` can be called again, which will access the second entry from the top in the queue. This can be simply put in a loop to display all the contents of a table.

There are numerous other database manipulation functions that are incorporated into PHP, but these few serve to illustrate the point. `mysql_query`, depending on the parameters supplied, is capable of executing most, if not all, of the database operations that a standard user may want to perform.

Conclusion

PHP is one of the most widely used web-based languages today. In the future, it will continue to grow and prosper. It's open-source nature and strong community will ensure its maintained success. Every week, new features are added to the language, which is something unheard of in most other HOL, where it takes years to implement changes.

PHP is simple enough to perform even the most mundane task, yet powerful enough to satisfy the professional web-programmer. It can be learned without spending any money at all. It is forgiving in that it won't crash a system like C/C++ can easily do. These features, coupled with its helpful community of developers, makes PHP a language that will be around for a long time.

References

- Bakken, S. et al. (2004) *PHP Manual*, The PHP Documentation Group,
<http://www.php.net>.
- Burns, J. (1998) *HTML Goodies*, Macmillan Publishing, Indianapolis, IN.
- Sebesta, R. (2004) *Concepts of Programming Languages*,
Addison-Wesley, Boston, MA.
- Zandstra, M. (2002) *Teach Yourself PHP in 24 Hours*, Sams Publishing,
Indianapolis, IN.