PHP Primer

Singapore
PHP
User Group

Introduction

PHP is a general-purpose scripting language especially suited for Web development. It is open-source and can be deployed on all major operating systems and web servers free of charge. It is imperative, reflective and supports object-oriented programming.

Basic Syntax

Similarities with other imperative languages

PHP code should look familiar to anyone who has had experience with C, C++ or Java. Statements are terminated with a ";" and blocks of code are delimited with "{" and "}". The "/*...*/" and "//" notations are both valid for PHP. Control structures such as *if-else* statements, *for* loops and *while* loops work in much the same way as with the abovementioned languages.

However, there are some differences and features that are unique to PHP which will be briefly explored in the rest of this primer.

Variables

Variables in PHP are represented by a "$" character followed by the name of the variable. For example, the variable "foobar" would be represented as "$foobar" in PHP code. Variable names in PHP are case-sensitive and must start with either an underscore or a letter, followed by any number of letters, numbers or underscores. The following code stub demonstrates the assignment of a value to a variable.

```
$foo = 'bar';
```

Note that PHP does not by default enforce the declaration of a variable before its usage, so some care should be taken not to attempt to access the value of a variable before one has been assigned to it.

Escaping HTML

PHP code can embedded within HTML using the "<?php" and "?>" tags. For example, the following code would output "Hello, foobar!".

```
<html>
<body>
<?php $lastname = "bar"; ?>
Hello, foo<?php echo $lastname; ?>!
</body>
</html>
```

Types

PHP supports the following eight primitive data types.

four scalar types: *boolean, integer, float, string*
two compound types: *array, object*
two special types: *resource,* NULL

PHP is a weakly-typed language. This means that the programmer is usually not required to resolve the type of a variable. Instead, the type of a variable is usually resolved during runtime depending on the context in which it is being used. This can sometimes be tricky, as we shall see.

Booleans

The *boolean* type can take the values *true* or *false*. Since PHP is weakly-typed, variables of other types can be implicitly cast as booleans. However, one must be careful when doing this, because some values of other types are evaluated to FALSE, as listed below.

- the integer 0 and the float 0.0
- the string "0" and empty strings
- an array with zero elements
- an object with zero member variables (PHP 4 only)
- the NULL special type

Integers and floats

Integers can be specified in decimal (10-based), hexadecimal (16-based) or octal (8-based) notation, optionally preceded by a sign (- or +). Octal numbers are preceded by *0* and hexadecimals are preceded by *0x*.

Floating point numbers can be specified in any of the following three formats.

```
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
```

Strings

A string in PHP can be specified with single quotes, double quotes or the heredoc syntax. Since this is a just a primer, the heredoc syntax will not be covered for the sake of brevity.

The simplest way to specify a string is to delimit it with single quotes. To specify a single quote within a single-quoted string, it needs to be preceded by a backslash (\). If a backslash is to occur before a single quote, then a double backslash is required. Otherwise, there is normally no need to escape backslashes. For example, the code below would print "This is a backslash \, a single quote ' and a backslash followed by a single quote \'".

```
  echo 'This is a backslash \, a single quote
\' and a backslash followed by a single quote
\\'';
```

For double-quoted strings in PHP, more of the escape sequences found in other languages may be used, such as "\n", "\t" and "\r". However, the most important feature of double-quoted strings in PHP is that variables are expanded. For example, the code below would print "Variable $foo has a value of bar".

```
  $foo = 'ba'.'r'; //'.' is the concatenation
operator
  echo "Variable \$foo has a value of $foo";
```

Arrays

It is beyond the scope of this primer to fully describe the PHP array type, so only a brief summary will be given. An array in PHP is an ordered map. A map is a type that maps *values* to *keys*. An array in PHP may be created using the *array()* function, as shown below.

```
  $arr=array("key1"=>"val1","key2"=>"val2","val
3");
```

The above statement would result in an array with the elements val1, val2 and val3. Since the key of val3 was not specified, it would take the value 0, the smallest available non-negative integer. Alternatively, the above array could have been specified using the following syntax.

```
$arr['key1'] = 'val1'; //$arr is created
$arr['key2'] = 'val2';
$arr[] = 'val3'; //$arr[0] is equivalent
```

Legal values for keys of arrays are any values of primitive types, with boolean true and false being cast as 1 and 0 respectively. Floating point numbers used as keys are truncated and NULL is cast as an empty string. An empty string, of course, remains an empty string.

Objects

Objects in PHP, like arrays, is a very wide topic and hence will not be covered beyond basic class declaration and object instantiation syntax.

```
 //class Declaration
class className [extends parentClassname] {
  var $attribute;
  //constructor
  function className([$params]){statement;}
  //member function
  function functionName([$params]){statements;}
}
 //class instantiation
$ob = new className();
//calling a member function
$result = $ob->functionName();
```

Type functions

Some useful functions for dealing with types are the *var_dump(), gettype()* and *is_** functions. The *var_dump* function recursively dumps the entire variable passed in, including the types of the variable and all its members (for objects) and elements (for arrays). The *gettype* function returns the type of the variable passed in and the *is_** functions test if a variable is of a certain type.

For more information about PHP data types, please refer to "http://php.net/manual/en/language.types.php".

Post, Get, Session and Databases

Since PHP is most commonly used to program web applications, it requires ways to obtain data from the user through HTTP POST and GET methods as well as ways to maintain coherent sessions for users. Finally, most web applications require ways to store user-submitted data.

Useful Superglobals

All POST variables may be obtained from an array called *$_POST* and all GET variables from the *$_GET* array. These variables are called superglobal variables, which means that they are automatically global. PHP provides a few other useful superglobals, including $_COOKIE for cookies and $_FILES for uploaded files.

Session Handling

To start a session, the *session_start()* function must be called before any other output is sent to the browser, after which the *$_SESSION* superglobal may be used to store all session variables. Subsequently, all pages that form a part of the session must also call the *session_start()* function. To end a session, the *session_destroy()* function is called.

Handling Databases

PHP has libraries for interfacing with most of the popular relational database servers in use today. These include MySQL, PostgreSQL, Oracle and MS SQL. It is beyond the scope of this primer to exhaustively explore the rich libraries of functions provided by PHP to access these databases.

However, one of the most useful features about PHP is that it tries to provide a more or less consistent suite of similarly-named functions that are valid for most databases. For instance, to connect to a database server, one would use a **_connect()* function, where * is replaced by the type of server. For a MySQL server, one would use *mysql_connect()*, for a MS SQL server, one would use *mssql_connect()* and so on.

The functions used to query databases are not so consistently named, but after a query has been executed, one would use the **_fetch_assoc()*, **_fetch_array()* and **_fetch_row()* functions to retrieve the data from the queries. Finally, to close the connection to the database, one would call the **_close()* function.

Control Structures and Operators

In most programming tutorials, these would be introduced earlier in the document. However, since this primer is primarily meant for programmers who are new to PHP, and also due to the fact that the syntax of these language constructs in PHP are similar to their counterparts in other languages, we have chosen provide only brief summaries of them at the end of this document.

Conditionals

if-elseif-else:

```
  if (condition){ statement;}
  //yes, elseif is one word
  elseif(condition){ statement; }
  else{ statement; }
```

switch case:

```
  switch (expression){
    case constant1: statement; break;
    case constant2: statement; break;
    default: statement; break;
  }
```

Iteration

while loop:

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

for loop:

```
  for(init expr; terminate expr; iterate
expr){ statement; }
```

```
foreach loop:
  foreach($array as $val){ statement; }
```

or

```
  foreach($array as $key=>$val){ statement; }
```

Operators (decreasing precedence)

Operators	Additional Information
`new`	new
`[]`	array()
`++ --`	increment/decrement
`instanceof`	types
`!`	logical
`* / %`	arithmetic
`+ - .`	arithmetic and string
`<< >>`	bitwise
`< <= > >=`	comparison
`== !=`	comparison
`=== !==`	comparison
`&`	bitwise and references
`^`	bitwise
`|`	bitwise
`&&`	logical
`||`	logical
`? :`	ternary
`= += -= *= /=`	assignment
`.= %= &= |=`	assignment
`^= <<= >>=`	assignment
`and`	logical
`xor`	logical
`or`	logical

PHP Primer v0.1
Created by: Adrian Quek and Chin Yong, Singapore PHP User Group (www.php.com.sg)

With kind support from Ablewise.com