PHP Quick Reference Sheet

SERVER REQUIRED

Pre-Hypertext Processing (PHP) is a server-side scripting language. This means that it resides on and is executed by the server. In order to develop PHP, one must have a server (locally installed or remotely accessed).

LOOSELY TYPED

PHP is a loosely typed language. This means that variable values are treated as entered. They do not have to be declared first and then stored only as that type.

DATA TYPES

PHP support eight primitive data types:
[Reference: http://www.php.net/manual/en/language.types.intro.php]
String – alpha-numeric characters
Boolean – TRUE or FALSE
Float – Numbers with decimal places
Integer – Numbers without decimal places
Array – A group of values identified using keys
Object – Used in Object Oriented Programming
NULL – Devoid of value
Resource – A special type of variable

CODE BLOCKS AND FILE EXTENSIONS

PHP scripts can be created in a PHP page – one which only contains PHP or embedded in a web page. In either case, the PHP scripting is enclosed in a PHP code block – `<?php ?>` and the page must have a .php extension. The extension is a trigger to the server that it must process the page, looking for and executing the PHP code.

COMMENTS

PHP allows three ways of adding comments to your code:

```
// - a single line comment
# - a single line comment
/* … */ - multiline comment
```

VARIABLES

As with every programming language PHP uses variables to store information. A PHP variable starts with a dollar sign ($) and must be followed by an alphabetic character or an underscore (_). Variable names should describe the information stored inside. [Reference: http://us3.php.net/manual/en/language.variables.basics.php]

```
$firstname – valid
$_firstname – valid
$FirstName – valid
$2names – invalid
```

CONSTANTS

A constant is a special type of variable. Its value does NOT change. It is set to one value and left. An example of using a constant is Pi. Pi's value is set and can therefore be assigned to a constant to be used as needed later.
[Reference: http://us3.php.net/manual/en/function.define.php]

```
define('PI', 3.14159);
```

The define function creates the constant by name (the first argument in the function, notice there is no $ in the name) and the value of the constant as the second argument; the two are separated by a comma. If the value of the constant is a string it must be quoted. Best practice dictates that constants be all uppercase. To use the constant, simply state the name. The example below calculates the circumference of a circle:

```
$circumference = 2 * PI * $radius;
```

QUOTES

PHP allows both single ' and double " quotes to be used. However, using single quotes around a value (e.g. 'Today is Wednesday') produces a string literal. This means that what is inside the single quotes is exactly what will be seen if the string is shown on the screen or used in some way.

However, the use of double quotes, particularly when a variable is placed inside the double quotes allows the value of the variable to be displayed. This is called interpolation. See the examples below:

```
$today = 'Today is Wednesday';
echo '$today';
```

This would produce *$today* on the screen.

```
echo "$today";
```

This would produce *Today is Wednesday* on the screen.

Best practice is to use single quotes in most situations except when placing a variable inside the quotes.

ARRAYS

An array is a variable that holds more than 1 value. Arrays are typically described as numeric, associative or multidimensional. Regardless of type you must remember that all values stored in an array have a 'key' that serves as a mechanism for identifying each value in the array.
[Reference: http://us2.php.net/manual/en/language.types.array.php]

NUMERIC ARRAY

A numeric array is one that uses a number as the label or "key" for the corresponding value. The key can be assigned by PHP or manually by the programmer. There are two ways of creating the array, shown below:

```
$numericarray = array(0=>'first',
                      1=>'second',
                      2=>'third');
```

In the example above, the key's (0, 1, 2) have been manually assigned. In the example below, PHP will automatically assign the key – it too will begin with zero and continue on. It is important to remember that numeric arrays always start with zero!

```
$numericarray[] = 'first';
$numericarray[] = 'second';
$numericarray[] = 'third';
```

ASSOCIATIVE ARRAY

An associative array uses a string as the key for each value. Because the key is a string it must be quoted. The key must be manually set or can be set as a result of a SQL query and use the database field name as the label. In the latter case, PHP provides ways of using the field names as the keys automatically.

Example 1

```
$assocarray = array('one'=>'first',
                    'two'=>'second,
                    'three'=>'third');
```

Example 2

```
$assocarray['one'] = 'first';
$assocarray['two'] = 'second';
$assocarray['three'] = 'third';
```

MULTIDIMENSIONAL ARRAY

A multidimensional array is literally an array of arrays. The multidimensional array can be either numeric or associative or mixed, although the mixed is not recommended. The arrays within the multidimensional array may also be one or the other or both. Below is an example of a numeric multidimensional array consisting of numeric arrays:

```
$multiarray = array(
      array('one','two','three'),
      array('uno','dos','tres'),
      array('eins','zwei','drei'));
```

Here is the exact same array built differently:

```
$multiarray[] = array('one','two','three');
$multiarray[] = array('uno','dos','tres');
$multiarray[] = array('eins','zwei','drei');
```

ACCESSING THE VALUES

Because $multiarray is a numeric array consisting of numeric arrays and all of these arrays start with zero the table below illustrates the keys:

	0	1	2
0	one	two	three
1	uno	dos	tres
2	eins	zwei	drei

```
echo $multiarray[1][1]; // Dos
echo $multiarray[2][0]; // Eins
```

The key in the first bracket references the small array inside of the larger array and the number in the second bracket references the value in the smaller array.

ADVANTAGES

Numeric arrays and multidimensional arrays have one benefit – they are easy to loop through. Associative arrays have a benefit as well, their keys are usually easier to read because they are labels and we as humans deal with labels easier than numbers.

LOOPS

A loop is a means of moving in a repetitious manner in order to perform a series of tasks. Loops are useful for working with arrays or for inserting or retrieving data to or from a database during a function execution.

FOR LOOP

A ***for*** loop is the fastest loop because all the factors are known in advance. It requires a counter, a condition to see if the loop should continue, an increment to the counter and the code to be executed within the loop. [Reference: http://www.php.net/manual/en/control-structures.for.php]

```
for($i = 1; $i <= 10; $i++) {
    Code to be executed… }
```

In the second example below, an array is stored in the variable. The loop is to continue as long as the array has data. The count function determines the size (number of items inside) of the array. By assigning the count to the $x variable, it only needs to be determined once.

```
for($i=0,$x=count($array); $i<$x; $i++) {
    Code to be executed… }
```

WHILE LOOP

A ***while*** loop operates much like a ***for*** loop, except that the incrementing of the counter happens at the conclusion of the code within the loop. [Reference: http://www.php.net/manual/en/control-structures.while.php]

```
$i = 1;
while ($i <= 10){
    … Code to be executed…
    $i++;}
```

As with the second example for the ***for*** loop, the following example would represent the ***while*** loop being used with an array.

```
$i = 0;
$x = count($array);
while ($i <$x){
    Code to be executed…
    $i++;}
```

FOREACH LOOP

A ***foreach*** loop is wonderful for small arrays and limited data sets. Because it is self-incrementing it is not good for large data sets because of performance issues. However it is easy to set up. [Reference: http://www.php.net/manual/en/control-structures.foreach.php]

```
foreach($array as $item) {
    … do something with the item…}
```

If the $array is multidimensional then the $item itself is a smaller array and the individual components of the small array can be worked with, like so:

```
foreach ($array as $item) {
    … do something with $item[#]…}
```

DO-WHILE

PHP also supports the ***do-while*** loop, but because it always runs at least once before a condition is checked, its use is discouraged.

OPERATORS

An operator performs some action, usually in regard to a variable. The chart below illustrates common operations and their functions.

Operator	Function	Example
=	Assigns a value on the right to the variable on the left.	$pi = 3.14
Mathematic Operators		
+	Adds two values together.	$total = $num1 + $num2
-	Subtracts a value from another.	$discount = $price - $discount
*	Multiplies two values.	$charge = $price * $salestax
/	Divides two values.	$average = $total / $numOfItems
%	What is left after division.	$balance = $cards % 10
++	Increments (increases) the value of a variable	$visits++
--	Decrements (decreases) the value of a variable	$inventory--
Comparison Operators (*Yields a Boolean – TRUE or FALSE – as a result of the comparison*)		
==	Compares two variables for equal value	$item1 == $item2
===	Compares two values for equal value and type	$item1 === $item2
>	Determines if the item on the left is greater than that on the right.	$value1 > $value2
<	Determines if the item on the left is less than that on the right.	$value1 < $value2
>=	Determines if the item on the left is greater than or equal to that on the right.	$discount >= .10
<=	Determines if the item on the left is less than or equal to that on the right.	$coupon <= $discount
!=	Determines if the item on the left is not equal to item on the right.	$markDown != $discount
<>	Determines if the item on the left is not equal to item on the right.	$markDown <> $discount

CONTROL (FLOW) STRUCTURES

Control structures determine what happens as the PHP code is read. PHP is a sequential language, meaning the code is read from top to bottom.

IF

An ***if*** is the simplest structure It consists of a comparison test that if TRUE the code within the ***if*** is executed. [Reference: http://www.php.net/manual/en/control-structures.if.php]

```
if(comparison test){
  … execute if comparison is true …
}
```

IF – ELSE

Adding an ***else*** to an ***if*** structure simply indicates that if the comparison test is FALSE, the ***else*** structure will be executed since the true code was not. [Reference: http://www.php.net/manual/en/control-structures.else.php]

```
if(comparison test){
  … execute if comparison is true …
} else {
  … execute if comparison is false …
}
```

ELSEIF

An ***elseif*** can be added after an ***if*** in order to provide a follow-up test. As many ***elseif***'s can be added as needed to test for multiple conditions. [Reference: http://www.php.net/manual/en/control-structures.elseif.php]

```
if(comparison test){
  … execute if comparison is true …
} elseif(second comparison test){
  … execute if 2nd comparison is true …
} … repeat as needed …
```

COMBINE EVERYTHING

```
if(comparison test){
  … execute if comparison is true …
} elseif(second comparison test){
  … execute if 2nd comparison is true …
} else {
  … execute if all comparisons are false
}
```

SWITCH

A ***switch*** (sometimes called a case) statement is another form of control structure. In this case a variable is tested for a variety of possible values. If a particular value is matched then code is executed. It is important to note that following the executable code is a "break". The "break" exits the switch. [Reference: http://php.net/manual/en/control-structures.switch.php]

```
switch($variable) {
  case "a possible variable value":
      Code to be executed…
      break;

case "another possible variable value":
    Code to be executed…
    break;

case "another possible variable value":
    Code to be executed…
    break;
}
```

As with the combined ***if – elseif- else***, the switch can have a default added to the end that allows for something to happen even if none of the variable values were matched.

```
switch($variable) {
      case "a possible variable value":
        Code to be executed…
        break;
      case "another possible value":
        Code to be executed…
        break;
      case "another possible value":
        Code to be executed…
        break;
      default:
        Execute if nothing matched…
        break;
}
```

FUNCTIONS

PHP has over 5000 functions to perform a huge range of programmatic tasks. However, there will be times when custom functions, written by you, will be needed to perform tasks within your own application. There are three key concepts to know when writing your own functions:

1. To create a function, use the keyword `function`.
2. The name of the function should reflect what it does.
3. The scope of the function is within its curly braces.

The function below will 1) remove any extra spaces from the beginning or end of a string, 2) remove any HTML tags from the string and 3) send back the resulting string:

```
function stripHTML($string){
  $string = trim($string);
  $string = strip_tags($string);
  return $string;
}
```

The $string variable inside the parentheses is knows as a parameter. It is sent into the function, where the trim function removes any extra spaces and sends the rest of the string back to the same variable. Then, the HTML tags are removed and what is left is assigned back to the same variable. Finally, the value of the variable is returned to wherever it was called from. For example:

```
$firstname = '  <b>Johnny Appleseed</b>';
$firstname = stripHTML($firstname);
echo $firstname;
```

Johnny Appleseed would be seen, minus the empty spaces and it would not be bold or differentiated from normal text in any way.

SCOPE

Scope refers to the context in which variables can be used and in which functions can be called. There are a number of special variables, referred to as "Super Globals" that are available anywhere in PHP. They are: [Reference: http://php.net/manual/en/language.variables.superglobals.php]

```
$_SERVER
$_SESSION
$_REQUEST
$_GET
$_POST
$_FILES
$_COOKIE
```

Local variables (meaning created and populated by the coder as part of PHP) have a scope typically only in the context in which they are defined. This would include within any required or included files. Variables declared outside of a function are NOT within the scope of that function unless specifically brought into scope using the ***global*** declaration. In the example below, the $link variable holds the connection to a database. The function is inserting registration information into the database. In order for the function to work, it must have the ability to connect to the database. Therefore, the connection must be brought into the function's scope.

```
function registerUser($firstname,
$lastname, $email){
  global $link; // brings the outside
variable into scope
  … other code would appear here …
}
```

ECHO AND PRINT

When the results of PHP need to be written to the browser screen the ***echo*** or ***print*** commands are used. They work identically and which one to use is personal preference. [Reference: http://us3.php.net/manual/en/function.echo.php]

```
$lastname = 'Appleseed';
echo $lastname;
print $lastname;
```

Both would produce *Appleseed* on the screen.

EXIT

If a php command occurs that is the end of a process, it is necessary to insure that nothing after it happens. To insure this, the `exit` command is used. For example: [Reference: http://us3.php.net/manual/en/function.exit.php]

```
$today = 'Friday';
if($today == 'Friday'){
echo 'Hooray, the weekend is here';
exit;
} else {
echo 'Sorry, get back to work';
}
```