PHP Quick Reference Card 1.02

Copyright©, 2008 BrandsPatch LLC

http://www.explainth.at

Color key on Page 4

Code Structure

```
<?php
$site = 'ExplainThat';

function sayHello(){
 $s = 'Hello from ';
 echo $s;
//single line comment
}

function sayHelloEx(){
 $s = 'Hello from ';
 global $site;
 echo $s.$site;
/*Comment spanning
  two or more lines */
}

sayHello();
print $site;
?>
```

Reusing Code

The **include**, **require**, **include_once** and **require_once** keywords facilitate the reuse of PHP code. For instance

include 'mycode.php';

would cause the contents of mycode.php to be merged into the current file. Failure to find mycode.php results in a warning. **require** behaves similarly but throws a fatal error. The **#_once** versions prevent function redefinition.

Nomenclature Rules

All labels[1] in PHP bear the form $*name*. *name* can consist of upper & lowercase letters a-z, extended ASCII characters from 0x7F to 0xFF, the underscore character, _ and numbers. The first character cannot be a number. Names are case sensitive. $this is a predefined read only variable used to refer to the current object context. There are no limits on name length. Names are case sensitive.

1 Strings used to identify constants, functions, variables & Heredoc

Visibility & Scope

Variables in PHP generally have a single scope – i.e they are always visible. However, user defined functions have their own local scoping context – i.e. do not have access to variables defined outside the function. A reference to $site in **sayHello** would merely create a new empty local variable called $site. To access such variables do one of the following

- Use the **global** keyword as in **sayHelloEx** above.
- Use the **$GLOBALS** array - **$GLOBALS['site']**

Data Types

PHP supports four scalar data types

- boolean - takes the values TRUE & FALSE
- integer - decimal, hexadecimal or octal. e.g. 32, 0x20, 040. The constants PHP_INT_MAX and PHP_INT_SIZE provide platform-dependent information. Integer overflow causes silent transformation of the variable into a float.
- **float** - Typically IEEE 64 bit with 14 decimal digits.
- string - single byte character sequence. See below.

Variables that have not been assigned a value or have been **unset** contain the special value **NULL**. A **NULL** assignment to an object variable destroys the object.

Explicit typecasts are rarely needed in PHP. If required use the code

$x = **(#)**$y; - where # is one of bool, float, int or string.

Operators

Operator	Example	Result
+ «-»	3 + 2	5
.	'Explain'.'That!'	'ExplainThat!'
/ «*»	3/2	1.5
%	7%4	3
=	$i = 2	$i set to 2
$i = 2;		
+= «-=»	$i+=1;	3
s = 'Explain';		
.=	s.='That!'	'ExplainThat!'
==[1]	3=='3' 3==3 3==2	true true false
===[2]	3=='3' 3==3 3==2	false true false
!= or <>	'php'!='PHP' 3!=3	true false
!==	3!=='3'	true
< «>»	2 < 3	true
<= «>=»	2<=3	true
$i = 2;$j = 5;		
&	$i & $j	2
|	$i | $j	7
^	$i ^ $j	5
~	~$i	-3
<< «>>»	$i << 1	4
++ «--»	$i++[3] ;++$i[4]	3
$i = 2;$j = 5		
&&	($i <= 2) && ($j < 7)	true
||	($i%2 > 0) \|\| ($j %2 == 0)	false
!	($i==2) && !($j %2 == 0)	true

1 called **loose** comparison; 2 called **strict** comparison
3 evaluates after use 4 evaluates before use

Constants

define($name,$value,[$ci])

is used to define a constant named $name with the scalar value $value. Case insensitive if $ci = **TRUE**.

- constant references **do not** start with a **$**
- constants cannot be altered
- constants are globally accessible

References to undefined constants are treated as string literals. PHP defines five magic constants whose value depends on where they are used

Name	Description
__LINE__	Current line number
__FILE__	Current file name with path
__DIR__	Path to current file
__CLASS__	Current class name
__METHOD__	Method name as *class:methodname*

Magic constants in included files are evaluated **prior** to inclusion.

Variable Management

Function	Purpose	Return Value
empty	Check if empty?	**boolean**
floatval	Convert to float	**float**
get_defined_vars	List all variables	**array**
gettype	Verify data type	**string**[1]
intval	Convert to int	**integer**
is_#[2]	Verify data type	**boolean**
serialize	Stringify for storage	**string**
settype[3]	Set data type	**boolean**
strval	Convert to string	**string**
unserialize	Regenerate from string	**boolean, integer etc**
unset[4]	Destroy the var	**-**

1 array, boolean, integer, double, string or object
2 # is one of array, bool, float, int, null, object, scalar, string
3 second parameter is a string. See note 1 above
4 behavior inside a function depends on nature of variable being unset

Arrays

Arrays are used to store sequences of related values in a tabular format. PHP has 5 ways of defining an array

```
$lamp = array();
$lamp[]='LINUX';$lamp[]='Apache';
$lamp[]='MySQL';$lamp[]='PHP';

$lamp = array('L'=>'LINUX','A'=>'Apache',
              'M'=>'MySQL','P'=>'PHP');
$lamp = array('LINUX','Apache',
              'MySQL','PHP');
$lamp = array();
$lamp[1]='LINUX';$lamp[2]='Apache';
$lamp[3]='MySQL';$lamp[4]='PHP';[1]

$lamp = array();
$lamp['L']='LINUX';$lamp['A']='Apache';
$lamp['M']='MySQL';$lamp['P']='PHP';
```

PHP arrays can be associations – i.e. a unique key, (e.g. 'L' above) is associated with each value in the array. For multiple dimensions use arrays within arrays.

Array Manipulation

Function	Description
array_change_key_case	Guess!
array_chunk($arr,$size,[$f])	Returns an array of arrays containing $size elements each from $arr. **TRUE** for $f preserves keys.
arra_fill_keys($keys,$values)	Create an associative array using $keys, and $values,.
array_fill($start,$num,$value)	Create an array with $num elements from index $start filled with $value
array_flip($arr)	Flip values and keys in $arr
array_key_exi	Check for $key in $arr

sts($key,$arr)	
array_reverse($arr,[$f])	Reverses element order. $f = TRUE preserves keys.
array_values($arr)	Returns all values in $arr in a numerically indexed array.
count($arr)	Returns element count
ksort($arr)	Sorts array using keys

1By default array indices start at 0. Here we force them to start at 1

Date & Time

Function	Description
getDate([$time])	Associative array with current time or $time exploded into

Key	Value
seconds	0-59
minutes	0-59
hours	0-23
mday	**1**(!)-31
wday	0(Sun)-6(Sat)
mon	1(!)-12
year	e.g 2008
yday	0(!)-365
weekday	Sunday-Saturday
month	January-December
0	UNIX Epoch

checkdate($month, $day,$year)	Validates date for $year between 1 & 32767
date($format, [$time])	Formats current time or **$time** using **$format**.

Format Char	Example
Day	
d	01-31
j	1-31
D	Mon
l(l.c. L)	Monday
N	1(Mon)
S	Suffixes st, nd etc. Use with j
w	0(Sun)
z	0-365
Week	
W	Week of year
Month	
F	January
m	01-12
M	Jan
n	1-12
t	Days in month
Year	
L	(Leap Year)?
	1:0
Y	2008
y	08
Time	
a	am or pm
A	AM or PM
g	1-12
G	0-23
h	01-12
H	00-23
i (Minutes)	00-59
s(Seconds)	00-59
U	UNIX Epoch
Timezone	
e	Europe/Paris
P	Δ to GMT

time()	UNIX Epoch time

Escape Sequences

Sequence	Meaning
\n	Linefeed, 0x0A
\r	Carriage Return, 0x0D
\t	Tab, 0x09
\v	Vertical tab, 0x0B
\f	Form feed, 0x0C
\\	Backslash
\$	Dollar sign
\"	Double quote
\x00 - \xFF	Char in hexadecimal

File System

Function	Description
basename($fname, [$suffix])	Filename minus path – and extension if $suffix is provided
file_exists($fname)	Does $fname exist? Works with files **&** folders
filesize($fname)	Guess?
fileatime($fname)	When was the file accessed? (UNIX time)
chmod($fname, $mode)	Change access rights to file $fname. $mode is an octal number in the format 0OGW where O = Owner, G = User group for Owner & W = the world, i.e. everyone else. Individual digits are made up by adding the desired rights as listed below

Value	Right
1	Execute
2	Write
4	Read

e.g. 0644 means read/write rights for owner & just read for others.

pathinfo($fname, [$options])	Returns associative array with keys 'dirname', 'basename', 'extension' & 'filename'. **OR** PATHINFO_# - # = uppercase keys above - into options for more selective results.
dirname($fname)	Counterpart of **basename** above
glob($pattern, [$flags])	Returns array of all filenames matching $pattern. **OR** GLOB_# flags for more selectivity

# Flag	Purpose
MARK	Add slash
NOSORT	As is list
BRACE	Expand {a,b,c} and match
ONLYDIR	Only folders
ERR	Stop on error

is_#($name) # = dir or file	is $name a folder or a file?
chdir($dname)	Change current directory. FALSE on failure.
closedir($dhandle)	Closes directory opened earlier using **opendir**.
getcwd()	Get current directory
mkdir($dname, [$mode,$recurse])	Makes directory $dname. $mode defaults to 0777 – ignored on Windows. $recurse forces all directories in $dname to be created
opendir($dname)	Opens $dname and returns handle.
readdir($dhandle)	Reads next filename from open directory.
rewinddir($dhandle)	Guess!
rmdir($dname)	Attempts to delete $dname – subject to permissions. FALSE on failure
scandir($dname, [$order])	Returns array of files in $dname. Provide $order = 1 for descending name sort.
disk_free_space($dname)	Guess!
rename($old,$new)	Guess!
fclose($fhandle)	Close **fopen**'d file
fopen($fname, $mode)[1]	Opens$fname. $mode can be

Mode	Meaning
r[1]	Read
r+[1]	Read/Write
w[1,2,3]	Write
w+[1,2,3]	Read/ Write
a[3,4]	Write
a+[3,4]	Read/Write
x[1,3,5]	Write
x+[1,3,5]	Read/Write

1File pointer at BOF 2Truncate file to zero length 3Create file if required 4File pointer

	at EOF 5Fail if file already exists Specify an additional **b** (binary), e.g. 'wb' for all write modes to prevent CR/LF character translation. Always specify **b** with all binary files, e.g. images.
file_get_contents($fname**)**[1]	Reads contents of $fname into a string.
fread($fhandle, $len**)**	Read to EOF or $len bytes from file opened for reading. **fopen** the file with 'b' in the mode flag.
funcate($fhandle, $size**)**	Truncates file open for writing. Adds null bytes if $size > filesize.
fwrite($fhandle, $str,[$len]**)**	Writes $str to file opened for writing. Stops at $len if $str length is greater.
file_put_contents($fname,$data, **[**$flags**])**	Combined **fopen**, **fwrite** & **fclose**. $fname can be a string or an array. $fname is created or overwritten. **OR** the following for $flags **Value** / **Meaning** FILE_APPEND — Append to file, not overwrite. LOCK_EX — Lock prior to write
fseek($fhandle, $offset,[$whence]**)**	Sets file pointer to $offset bytes from $whence which is one of **Value** / **Meaning** SEEK_SET[1] — BOF SEEK_CUR — Current pos SEEK_END — EOF 1Default
ftell($fhandle**)**	File pointer position
rewind($fhandle**)**	File pointer to BOF. Useless in a/a+ modes
fflush($fhandle**)**	Commits buffered writes

1 $fname can be a URL.

Math Functions

Function	Description
abs($num**)**	Absolute value of $num
a#($arg**)**[1]	Inverse trig functions. $arg in rad.
a#h($arg**)**[2]	Inverse hyperbolic functions
base_convert($num, $from,$to**)**	Convert bases. $to & $from in the range 2..36. Letters a..z used for $from/$to > 10.
ceil($num**)**	Rounds up $num to integer.
dechex($num**)**	$num in hexadecimal
deg2rad($deg**)**	Degrees to radians
exp($num**)**	**e**$num
floor($num**)**	$num rounded down to integer
fmod($x,$y**)**	Floating point remainder of $x/$y
hexdec($str**)**	Decimal equivalent of hexadecimal $str. Invalid chars in $str are ignored.
log10($num**)**	Guess!
log($num,[$base]**)**	$num to **e** or $base
pi()	Approx value for π
pow($num,$base**)**	numbase
rad2deg($rad**)**	Radians to degrees
rand([$min],$max**)**	Random value 0/$min.. $max.
round($num,[$prec]**)**	**round(**3.142**) = 3** **round(**3.142,0**) = 3** **round(**3.14159,1**) = 3.2** **round(**12811,-2**) = 12800**
sqrt($num**)**	Squareroot of $num or NaN

1# is cos, sin or tan. 2cos, sin or tan. Not on Windows

Output & Formatting

Function	Note
echo $arg1[,**$arg2**...]	Echo to standard output
print $arg	Output a string
print_r($arg**)**	$arg in human readable format. Handles objects too. Very useful with arrays.
printf($fmt,$arg1[, $arg2...]**)**[1]	Prints args using format information in $fmt **Format** / **Output** %b — Integer as binary %c — ASCII char %d — Integer %e — "**E**" notation with **p** (see below) digits %f — Floating point %s — String %x — Hexadecimal l.c. %X — Hexadecimal u.c. %% — Literal %

Each % argument can have a number of optional specifiers. In order they are

- **%+** : Sign specifier. Outputs + or -. Default is no + sign
- **%0, %** or **%'c:** padding specifier. Uses 0, space or the character c for padding
- **%-**:alignment specifier. Causes left justification. Default is right justification.
- **%w**:Width specifier. Output has a minimum of **w** characters.
- **%.p**:Precision specifier. Decimal digits for floats and number of characters for strings.

Everything else in $fmt gets treated as literal text.

Examples

Format	Output
printf('%d',23**)**	23
printf('%03d',23**)**	023
printf('%.3f',3.141596**)**	3.142
printf('%.3s','PHP Script'**)**	PHP
printf('%s%3d','Route',66**)**	Route 66

1sprintf is similar but returns a string

Strings

PHP strings are sequences of single byte characters. They can be defined in four different ways

- **Single Quoted**: e.g. 'ExplainThat'. Variables and escape sequences other than \' and \\ are not expanded.
- **Double Quoted**: e.g. "One\nTwo". Variable references and escape sequences are expanded.
- **Heredoc**: To define complex strings like double quoted strings but without using double quotes. e.g.

```
$x = <<<PHP
<pre>
For more information see
http://www.php.net
</pre>
PHP;
```

1.<<<IDENT must be followed by a newline character
2.The actual string contents follow
3.The closing identifier must **not** be indented and cannot have any following characters except **;**

- **Nowdoc**: The sinqle quoted string equivalent of Heredoc. Similar syntax but with <<<'IDENT' (quotes!)

Strings can be treated as zero based arrays to access individual characters, e.g. $Name[0].

String Manipulation

Function	Description
. (not +!)	String Concatenation
strlen($str**)**	String length
strpos($str,$find, [$off]**)**	First $find in $str optionally starting at $off
strrpos($str,$find, [$off]**)**	Ditto but reports last $find
stripos & strripos	Case insensitive versions
strtolower & **strtoupper**	Guess!
chr($ascii**)**	Char at $ascii
ord($str[index]**)**	Ordinal value
explode($delim,$str, [$lim] **)**	Returns array of substrings of $str delimited by $delim, optionally with $lim elements.
implode($glue, $pieces**)** Alias **join**	Concats $pieces array using $glue.
ltrim($str,[$clist]**)** Similarly **rtrim(**$str, [$clist]**)** and **trim.**	Strip chars in $clist from left of $str. Strips spaces by default.
strip_tags($str, [$retain]**)**	Discard HTML & PHP tags. Retain tags in $retain.
substr($str,$start, [$len]**)**	Returns substring, optionally $len long starting at $start. -ve $start for substring from end of $str. -ve $len to omit chars from end of substring.
substr_count($str, $sub,[$start,$len]**)**	Occurrences of $sub in $str. Optionally starting at $start and within $len of $start.
str_replace($search, $rep,$str,[$count]**)**	Replaces $search in $str with $rep. Reports replacements in $count.
ucwords($str**)**	All words in $str to uppercase.

Conditional Execution

if (ConditionA) ifStmt;[elseif(ConditionB) elseifStmt;][else elseStmt;]

Multiline **#Stmt** code must be placed in braces, **{}**. The **;** terminating each statement is obligatory – even if the next character is a closing brace, **}**.

```
switch ($var){
  case Value1:Code;
                  break;
  [case Value2:Code;
                  break;
  ...]
  [default:Code;]
}
```

- $var can be a boolean, an integer or a string.
- Note the **break** after each **case** statement.
- If **default** is not the last option provide a **break**.
- To execute the same action(s) for a range of cases

```
switch ($var){
  case Value1:
  case Value2:
  case Value3:Code;
                  break;
 ...}
```

case comparisons are **loose**. Beware of **switch** blocks that use mixed values in individual **case**. The block may terminate prematurely because of a partial **case** match.

```
(condition)?trueCode:falseCode;
```

#Code can be a function call. This is the PHP ternary conditional operator. It can be assigned to **return**, **print** or **echo**, passed as a parameter in a function call etc. Parentheses are not necessary but recommended.

Exception Handling

```
<?php
function inverse($a){
 if ($x == 0) throw new Exception('Zero divide');
 return 1/$a;
 //not executed if exception is thrown}

function whenExcept($e){
echo $e->getMessage().'<br>';}

set_exception_handler('whenExcept');
//default exception handler
try{
  echo inverse(5);
  echo inverse(0);//triggers exception}
catch (Exception $e) {
echo 'Error '.$e->getMessage().'<br>';}
echo 'Hello world!';
//executed since exception was caught
throw new Exception('Oops');
echo 'Moien!';//not executed
?>
```

Looping

```
function whileLoop($num){
 while ($num > 0)
 {echo($num).'<br>';
  $num--;}
}

function doLoop($num){
 do{
     echo($num).'<br>';
     $num--;
 }while ($num > 0);
}

 function forLoop($num){
 for ($i=0;$i<$num;$i++){
      echo $i.'<br>';
 }
}
```

break causes immediate termination of the loop. Loop statements after **continue** are skipped and the next execution of the loop is performed.

```
 function foreachLoopA(){
 foreach($GLOBALS as $key => $value){
      echo $key.'='.$value.'<br>';
 }
}

 function foreachLoopB(){
 foreach($_SERVER as $value) echo $value.'<br>';
}
```

foreach offers a neat way of iterating over an array.

User Functions

```
function calcArea($x,$y,$isRect = true){
return ($isRect)?$x*$y:0.5*$x*$y;
//assume triangle if $isRect is false
}
```

Scalar function arguments can be given a default value – e.g. $isRect = **true** as above. Parameters are passed by value. To pass them by reference precede the parameter name in the function declaration with an ampersand, e.g. &$y.

return causes immediate termination of the PHP function. If no value is returned, or if **return** is missing the function return type is **NULL**

exit($status) - **die –** causes immediate termination of the current script. If $status is a string it will be printed. If it is an integer it will be the exit status.

Superglobals

Superglobals are arrays containing variables from the web server (when applicable), the environment and user input. They are always visible.

Variable	Contents
$GLOBALS	All below in a one array
$_SERVER	Server information
$_GET	HTTP GET variables
$_POST	HTTP POST variables
$_FILES	HTTP file upload variables
$_SESSION	Session variables
$_ENV	Environment variables
$_COOKIE	HTTP cookies

There can be minor, server-dependent, variations in the information returned in variables such as $GLOBALS, $_SERVER etc. To check just what is available use the script below to dump these variables to your browser.

```
<?php
function dumpThis($sg){
 foreach($sg as $key => $value){
     echo $key.'='.$value.'<br>';
 }
}

dumpThis($_SERVER);
?>
```

Miscellanea

Warning – thoughtless use of the features described here could seriously damage your server installation.

The prepend operator, **@**, can be used with constants, variables, function calls, **eval** and **include** to suppress error messages.

The backticks operator `returns the results of running a shell command. For instance, `**ls**` - **dir** on Windows – would return a directory listing. This can be assigned to a variable or echoed to standard output. Typing 96 while holding down the **ALT** key is a keyboard layout independent way of entering the ` operator.

eval($expr) evaluates the PHP code provided in the string $expr. The string must be valid PHP code – inclusive of terminating semicolons. Errors in $expr may cause the parser to die. Code in $expr forms part of the parent script so variable assignments in $expr are retained.

PHP in HTML

The safest way to embed PHP code in HTML is to delimit it using the <?php...?> tag pair. Other syntax exist but are not accepted by all web servers. The resulting file should be saved with the extension **.PHP**.

To call a PHP script via SSI use

```
<!--#include virtual="/path/scriptname.php"-->
```

If the same script is called from includes in different HTML files you can access the identity of the parent HTML file using $_SERVER['REQUEST_URI'].

Notes

Color Key

while – PHP keyword
funcName – user function
echo – language construct
$var – variable 'string'
3.142 – number
true – case insensitive
«x» - similarly **x**
[option**]**
//comment
constant

Using value == $var rather than $var == value. when doing comparisons against a value avoids bugs arising from typing = in place of ==.