PHP Reference Sheet

Andreas Plank (version April 15, 2010)

Basics (1)

Basics > Comments (1.1)

```
// Comment text
/* Multi-line comment text */
# Comment text
```

Basics > Data Types (1.2)

```
T/F bool boolean /* false, true */
0-9 int integer /* -1, 2, 0 */
1.209 float   /* 1.3445363 */
A-Z string /* 'text' */
array  /* $arr=array("key"=>"value", "something");*/
object  /* see class on page 4 */
resource /* external reference: pdf, aspell */
NULL  /* unset(), set NULL, not yet set */
```

Basics > Recommended Naming Rules/Documentation (1.3)

```
# Hungarian notation e.g.: $boolPost, $strContent ...
$intXxx or $iXxx # integer
$floatXxx or $fXxx # float
$doubleXxx or $dXxx # double==float
$strXxx or $sXxxx # string
$arrXxx or $aXxx # array
$boolXxx or $bXxx # boolean 'true' 'false'
$objXxx or $oXxx # object
/* classes, methods, objects */
class classText # use a noun with classes/objects
$objText = new classText();
# functions with verbs: get.. add... has... etc.
$objText->readFromDb(); # not $objText->readTextFromDb();
/** see phpDocumentor http://phpdoc.org/
 * showing a head line (short description)
 * bla bla bla (long description ...)
 * @author Max Muster <max@foo.bar>
 * @param string $strHead text for headings/sections
 * @param int $intSize=12 font size in pt
 * @return string in different sizes
 */
function showHeadline($strHead, $intSize=12){
  return "<h1 style=\"font-size:$intSize ↩
pt;\">$strHead</h1>";
}
```

Basics > Super Globals/Magic Constants (1.4)

```
$GLOBALS  #Access all global variables in script
$_SERVER  #Access web server variables
  ['PHP_SELF'] # /folder/testphp.php, file without root
  ['SCRIPT_NAME'] # /folder/testphp.php, file without root
  ['DOCUMENT_ROOT'] # /absolute/root/web/dir
  ['SCRIPT_FILENAME'] # /absolute/script/path/index.php
  ['REMOTE_ADDR'] # remote IP from user 91.64.225.29
  ['REMOTE_HOST'] # remote hostname
  ['REQUEST_URI'] # requested URL from current page
$_GET     #Values passed to script through URL
  ['nameOfFormElement'] = <value>
$_POST    #Values passed to script through HTTP Post
  ['nameOfFormElement'] = <value>
$_COOKIE  #Values passed by user cookie
$_FILES   #Values passed by HTTP Post File Uploads
  ['inputTagName']['name'] /* original file name of the↩
 file on the client machine. */
  ['inputTagName']['type'] /* The mime type NOT PHP ↩
checked! */
  ['inputTagName']['size'] /* The size: bytes */
  ['inputTagName']['tmp_name'] /* temporary filename on↩
 the server. */
  ['inputTagName']['error'] /* The error code ↩
associated with this file upload.*/
$_ENV     #Values passed to script via the environment
$_REQUEST #Values passed by URL, HTTP Post, Cookies
$_SESSION #Values passed through user's session
__LINE__  # current line number of the file
__FILE__  # full path and filename of the file.
```

If used inside an include, the name of the included file is returned. Since PHP 4.0.2, __FILE__ always contains an absolute path whereas in older versions it contained relative path under some circumstances.

```
__FUNCTION__  # The function name
__CLASS__   # class name
__METHOD__  # function name or class::method
```

Basics > External (Script) Files (1.5)

```
# Warnings + Fatal Errors given:
include '[url/]file.php';
include_once '[url/]file.php';
```

Variables as $_SERVER used in the file included return the current path not the path from included file itself.

```
include 'http://www.xxx.com/file.php?foo=1&bar=2';
# Warnings = Fatal Error!
require '[url/]file.php';
require_once '[url/]file.php';
0-9 readfile('[url/]file.php'); /* reads a file
and writes it immediately to the output buffer */
A-Z file_get_contents('[url/]file.php');
# get contents of a file into a string:
$filename = "/usr/local/something.txt";
$handle = fopen($filename, "r");
$contents = fread($handle, filesize($filename));
fclose($handle);
```

Basics > Outputs (1.6)

```
0-9 print( string $arg ); # Output a string
void echo( string $arg1 [, string $... ] );
mixed print_r( mixed $expression [, bool $return ]);
```

Basics > Declare Variables Constants (1.7)

```
/* type casiting + declaration printed with:
 echo("Hello ".print_r($test, true)."!"); */
$test  = (string) "world";   # Hello world!
$test  = (string) array("Hello", "world"); # Hello Array!
$test  = (bool)    "world";  # Hello 1!
$test  = (boolean) "world";  # Hello 1!
$test  = (int)     "world";  # Hello 0!
$test  = (integer) "world";  # Hello 0!
$test  = (float)   "world";  # Hello 0!
$test  = (double)  "world";  # Hello 0!
$test  = (real)    "world";  # Hello 0!
$test  = (array)   "world";
  # Hello Array ( [0] => world ) !
$test  = (object)  "world";
  # Hello stdClass Object ( [scalar] => world ) !
/* declare long strings */
$file=<<<EOD
hello world here
is mucht text ...
EOD;
# end here
/* declare constants: not changable */
  define("FOO", "something");
  define("FOO2", "something else");# 2FOO is invalid
  define("FOO_BAR", "something more");
```

Basics > Arithmetic Operators/Calculations (1.8)

```
+, -, *, /, % /*Modulus*/
5 % 3; # gives 2 (remainder 2 of 5 divided by 3)
pow(2,8); # 2^8
sqrt(10); # ∛10
exp(<float>) # e^<float>
log(<nfloat>[, <bfloat>]) # log_b n
pi() # 3.1415926... π
```

Basics > Relational/Logical Operators (1.9)

```
$a == $b  # equal
$a === $b # identical: equal + same type
$a != $b  # not equal
$a <> $b  # not equal
$a !== $b # not identical: not equal + not same type
$a < $b   # less than
$a > $b   # greater than
$a <= $b  # less than or equal to
$a >= $b  # greater than or equal to
$a and $b # And: if both $a and $b are TRUE.
$a && $b  # And: if both $a and $b are TRUE.
$a or $b  # Or: if either $a or $b is TRUE.
$a || $b  # Or: if either $a or $b is TRUE.
$a xor $b # Xor: if either $a or $b is TRUE, but not both
!$a       # Not: if $a is not TRUE.
```

Basics > Assignment Operators (from right) (1.10)

```
=, +=, -=, *=, /=, .=, %=, &=, |=, ^=, <<=, >>=
= /*assign*/ += /*addition*/ -= /*subtraction*/
$a = 3; $a += 5; # $a = 8 same as: $a = $a + 5;
*= /*multiplication*/ /= /*division*/
.= /*concatenation*/
$b = "Hello "; $b .= "There!"; # "Hello There!"
%= /*modulus*/ &= /* reference*/
```

References in PHP are a means to access the same variable content by different names. Note that in PHP, variable name and variable content are different, so the same content can have different names. The most close analogy is with Unix filenames and files - variable names are directory entries, while variable contents is the file itself.

```
|= /*or*/ ^= /*exclusive or*/
<<= /*left shift bitwise*/ >>= /*right shift bitwise*/
```

Bitwise operators allow you to turn specific bits within an integer on or off. If both the left- and right-hand parameters are strings, the bitwise operator will operate on the characters' ASCII values.

Manipulation Variable/Array (2)

```
$var = <value>;
$var = (string) <value>;
$anothervar =& $var; # assigned by reference
$arr = array();
```

```
$arr = array(# keys not explicit defined
  <val1>, # key starts at 0: $arr[0]
  <val2>, # $arr[1]
  <val3>  # $arr[2]
);# end array()
$arr = array(# keys defined
  <key1> => <value>,
  <key2> => <value>
);# end array()
$multiarr = array(# multi-dimensional
  <key> => array(<value1>,<value2>)
);# end multi array()
```

Manipulation > Array Functions (2.1)

```
T/F sort(<arr>); #  values + new keys 0 => A, 1 => B, 2 => C
T/F rsort(<arr>); # values reverse + new keys 0 => C, 1 => B, 2 => A
T/F asort(<arr>); #  values + old keys 0 => A, 1 => B, 2 => C
T/F arsort(<arr>);# values reverse + old keys 2 => C, 1 => B, 0 => A
T/F ksort(<arr>); # sort keys 0 => A, 1 => B, 2 => C
T/F krsort(<arr>); # sort keys reverse 2 => C, 1 => B, 0 => A
T/F natsort(<arr>); # natural sorting
0-9 count(<arr>); # count elements
0-9 count(<arr>,COUNT_RECURSIVE); # count multidimensional
array_push(<arr>,<val>); # push item onto the end
array_pop(<arr>); # pop item off from end
array_shift(<arr>); # shift item off from begin
array_slice(<arr>, off [, len]); # extract a slice
array_splice(<arr>, off [, len [, repl]]);
# remove a portion of an array (and do replace)
void list(mixed varname, mixed ... )
$info = array('coffee', 'brown', 'caffeine');
// Listing all the variables
list($drink, $color, $power) = $info;
echo "$drink is $color with $power.\n";
array_values(<arr>) # values + new index
array_keys(<arr>) # return keys
array_unique(<arr>) # without duplicates
array_merge(<arr1>,<arr2>) # number integer keys anew
T/F in_array('string/float', <arr> [, bool strict])
  # check for values
mixed array_search('string/float', <arr> [, bool strict])
  # returns the corresponding key if successful or T/F
```

Manipulation > String Manipulation (2.2)

Use multibyte string functions `mb_...` for unicode handling.

```
A-Z substr(<str>,<start>,[<len>]); # subsring
0-9 strlen(<str>); # length of string
trim(<str> [, charlist] );
ltrim(<str> [, charlist] ); # trim left
rtrim(<str> [, charlist] ); # trim right
```

strip whitespace/other characters from the beginning and/or end of a string

```
$text = "\t\tThese are a few words :) ...  ";
echo trim($text); # "These are a few words :) ..."
echo trim($text, " \t.");#"These are a few words :)"
$trimmed = ltrim($text, " \t.");
# left trimmed: "These are a few words :) ...  "
$clean = ltrim($binary, "\x00..\x1F");
```

trim ASCII control characters at the beginning of $binary (from 0 to 31 inclusive)

```
strtolower(<str>); # STRING -> string
strtoupper(<str>); # string -> STRING
str_replace(<search>, <repl.>, <str>,[<count>]);
preg_replace(<search>, <repl.>, <str>,[<limit>[,&count]);
```

every parameter in `str_replace()` can be an array

```
# Provides: <body text='black'>
$bodytag = str_replace("%body%", "black", "<body ↩
text='%body%'>");
0-9 strpos(<str>, <search>); # first string occurrence
$newstring = 'abcdef abcdef';
  $pos = strpos($newstring, 'a', 1);# $pos = 7, not 0
strcmp(<string1>,<string2>); # binary safe string compare
strncmp(<string1>,<string2>,<len>);
# binary safe string comparison of the first n characters
strcasecmp(<string1>,<string2>);
# binary safe case-insensitive comparison
```

all ...cmp() return < 0 if str1 is less than str2; > 0 if str1 is greater than str2, and 0 if they are equal.

```
strip_tags(<str>,[<allowable tags>]);# remove PHP/HTML
$text = '<p>Test paragraph.</p><!-- Comment --> ↩
Other text';
  echo strip_tags($text);# 'Test paragraph. Other text'
eval("return $value;"); # evaluates the string
explode(<delim>,<str>,[<limit>]);
  # Break string into array
A-Z implode(<delim>,<array>);
  # Join array into string separated by delim
A-Z rawurlencode("strings /?'ü'ß"); # 'historical'
  # strings%20%2F%3F%27%C3%BC%27%C3%9F
A-Z urlencode("strings /?'ü'ß"); # should be used
  # strings+%2F%3F%27%C3%BC%27%C3%9F
A-Z htmlentities("strings /?'ü'ß",ENT_QUOTES, 'UTF-8');
  # strings /?&#039;&uuml;&#039;&szlig;
A-Z utf8_encode("iso-8859-1"); # utf8 <- iso-8859-1
A-Z utf8_decode("utf8"); # iso-8859-1 <- utf8
```

Manipulation > Strings > Formatted Strings (2.2.1)

```
A-Z s|printf( A-Z <format> [, mixed args [, mixed ...]]);
echo sprintf("%04d-%02d-%02d", $year, $month, $day);
$n =  439; $u = -439; $c = 65; // ASCII 65 is 'A'
# notice double %%, this prints a literal '%' character
printf("%%b = '%b'\n", $n);# %b = '110110111' binary011₂
printf("%%c = '%c'\n", $c);# %c = 'A', same as chr()
printf("%%d = '%d'\n", $n);# %d = '439' 'decimal'123₁₀
printf("%%e = '%e'\n", $n);# %e = '4.39000e+2' scientific
printf("%%u = '%u'\n", $n);# %u = '439' unsigned123₁₀
printf("%%u = '%u'\n", $u);# %u = '4294966857' from neg.
printf("%%f = '%f'\n", $n);# %f = '439.000000' floating
printf("%%o = '%o'\n", $n);# %o = '667' octal123₈
printf("%%s = '%s'\n", $n);# %s = '439' as 'sting'
printf("%%x = '%x'\n", $n);# %x = '1b7' hexadecimal lower
printf("%%X = '%X'\n", $n);# %X = '1B7' hexadecimal UPPER
printf("%%+d= '%+d'\n",$n);# %+d = '+439' +/-123₁₀
printf("%%+d= '%+d'\n",$u);# %+d = '-439' +/-123₁₀
/* padding or cutoff */
$s = 'monkey'; $t = 'many monkeys';
printf("[%s]\n",      $s);# [monkey]     standard output
printf("[%10s]\n",    $s);# [    monkey] right adjust
printf("[%-10s]\n",   $s);# [monkey    ] left adjust
printf("[%010s]\n",   $s);# [0000monkey] zero padding
printf("[%'#10s]\n",  $s);# [####monkey] #-padding
printf("[%10.10s]\n", $t);# [many monke] cutoff
/* referring to positions */
$format = 'That\'s a nice %2$s full of %1$d monkeys.';
printf($format, 3, 'place');
# That\'s a nice place full of 3 monkeys.
```

Manipulation > Strings > Formatted Date (2.2.2)

```
date("r"); # Sun, 14 Dec 2008 14:01:24 +0100
date("F j, Y, g:i a<b\\r>");# December 13, 2008, 3:17 pm
date("m.d.y<b\\r>");  # 12.13.08
date("j, n, Y<b\\r>");# 13, 12, 2008
date("Ymd<b\\r>");    # 20081213
date('h-i-s, j-m-y, it is w Day z <b\r>');
  #03-17-04, 13-12-08, 1731 1704 6 Satpm08 347
/* escaping */
date('\i\t \i\s \t\h\e jS \d\a\y.<b\r>');
  # it is the 13th day.
date("D M j G:i:s T Y<b\\r>");
  # Sat Dec 13 15:17:04 CET 2008
date('H:m:s \m \i\s\ \m\o\n\t\h<b\r>');
  # 15:12:04 m is month
date("H:i:s<b\\r>"); # 15:17:04
```

Searching (3)

Searching > Regular Expressions (3.1)

```
^ # at beginning of a string: "/^cat/" any string 'cat..'
$ # at the end of strings "/cat$/" any string '...cat'
. # any single character "/cat./" -> catT, cat2 NOT catty
() /*a(bee)?t -> at, abeet NOT abet or ref to found \\1*/
[] # character class: gr[ae]y matches gray or grey
[^] # without-expression: 1[^02] -> 13 but not 10 or 12
[-] # ranges, [1-9] matches 1,2..,9 EXCEPT 0; [a-züßæ]
### Quantifiers
? # 0, 1 times: colou?r -> color, colour NOT colouur
+ # 1, multiple times: be+ -> be, bee NOT b
* # 0, multiple times: be* -> b, be, beeeeeeeeee
{n} # pattern n-times: /'[0-9]{3}'/ -> 264 NOT 12 or 8372
{n,} # n-times or more: '/[0-9]{3,}/' 347, 92038, 9074
{n,m} # n to m-times: '/[0-9]{3,5}/'  any 3, 4, 5 digits
| # alternatives: July (first|1st|1) -> 'July 1st' NOT ↩
'July 2'
### Character Definition  Example
[:alnum:] /* alpha-numeric character: [[:alnum:]]{3} ->↩
 letters/numbers [a-z0-9] like '7Ds' */
[:alpha:] /* alphabetic character: [[:alpha:]]{5} -> 5 ↩
alphabetic characters, any case, like aBcDe */
[:blank:] /* space and tab [[:blank:]]{3,5} matches any↩
 3, 4, or 5 spaces and tabs */
[:digit:] /* digits  [[:digit:]]{3,5} matches any 3, 4,↩
 or 5 digits, like 3, 05, 489*/
[:lower:] /* lowercase alphabetics [[:lower:]] */
[:upper:] /* UPPERCASE alphabetics [[:upper:]] */
[:punct:] /* punctuation characters [[:punct:]] -> ! or↩
 . or , NOT 'a' or '3' */
[:space:] /* all whitespace characters: [[:space:]] ↩
space, tab, newline, or carriage return */
### Perl-Style Metacharacters
```

```
// # Default delimiters: '/colou?r/' finds color or colour
i /* (PCRE_CASELESS) case insensitive: /colou?r/i ↩
matches COLOR or Colour */
m /* (PCRE_MULTILINE)  multi line search: '/^.+$/m' -> ↩
a whole line */
s /* (PCRE_DOTALL) '.' matches all characters, ↩
including newlines */
e # evaluate PHP
U # ungreedy: '/<a.+>/U' finds single link tags
u # pattern strings are treated as UTF-8
x # space is ignored
### escaping strings
\b /* word boundary: spot between word (\w) and ↩
non-word (\W) characters  /\bfred\b/i matches Fred but ↩
not Alfred or Frederick */
\B /* non-word boundary /fred\B/i -> Frederick but not ↩
Fred */
\d # a single digit: /a\db/i matches a2b NOT acb
\D # a single non-digit: /a\Db/i matches aCb but not a2b
\n # The newline character. (ASCII 10) /\n/ -> newline
\r # the carriage return character (ASCII 13)
\s # a single whitespace: /a\sb/ matches 'a b' NOT 'ab'
\S # non-whitespace character /a\Sb/ matches 'a2b'  'a b'
\t # the tab character. (ASCII 9)
\w /* single word: alphanumeric + underscore /\w/ -> ↩
'1' or '_' NOT '?' */
\W /* single non-word: /a\Wb/i matches a!b NOT a2b
### e = evaluates code
'/(.+)/e' -> 'strip_tags("\\1", "<img><a>")'
'/(.+)/e' -> '\'\\1\'==\'text\'?\',\':\'\''
#'text' == 'text'?','':''; → if identical ',' else ''
### greedy/ungreedy
"/\<style.*[^\/]>.+<\/style>/U" # remove CSS
```

Searching > Common Functions (3.2)

```
A-Z str_replace(<search>,<replace>,<str>,[<count>]);
```

i Every parameter in str_replace() can be an array, If no regular expressions are needed take this function

```
### replace string/array by regular Expressions
A-Z preg_replace(<search>, <repl.>, <str>, ↩
[<limit>[,&count]);
  # <search>,<replace>,<str> can be an array()
0-9 preg_match_all(<search>, <str>, [array] &matches [,0-9 ↩
flags [,0-9  offset]])
### extract array by regular Expressions
[array] preg_grep(<search>, <array>);
$arrayOut = preg_grep("/^(\d+)?\.\d+$/", $array);
  # only float numbers
```

Cookies (4)

```
T/F setcookie(<cookiename>, [<value>], ↩
[<expire_time_in_secs_since_epoch>], [<path>], ↩
[<domain>], [T/F <secure>], [T/F <httponly>]);
```

i Cookies are part of the HTTP header, so setcookie() must be called before any output is sent to the browser. This is the same limitation that header() has.

```
$value = 'something from somewhere';
setcookie("TestCookie",$value,time()+3600);
  # expire in 1h
$_COOKIE['cookiename']; # Returns value of cookie
```

Sessions (5)

```
T/F session_start(); # Create session
$_SESSION['key_name'] = value; #set variable
  $variablename = $_SESSION['key_name']; # return value
T/F session_destroy(); # Destroy session
0-9 session_cache_expire(<0-9>);
# return current cache expire given in minutes
A-Z session_cache_limiter(<str>);
```

i get and/or set the current cache limiter: 'nocache' disallows any client/proxy caching, 'public' permits caching by proxies and the client, 'private' disallows caching by proxies and permits the client to cache the contents.

```
T/F session_decode(<string data>);# decodes data
A-Z session_encode();#encodes the current session data
[array] session_get_cookie_params()#get session cookie params
A-Z session_id([<str>]); # get and/or set the current ↩
session id
T/F session_is_registered(<str>); # find out whether a ↩
global variable is registered in a session
A-Z session_name([str])
# get and/or set the current session name
T/F session_regenerate_id()
# update the current session id with a newly generated one
T/F session_register(<str>[, <str>]) # register one or ↩
more global variables with the current session
A-Z session_save_path([<str>])
# get and/or set the current session save path
void session_set_cookie_params(0-9 lifetime [, <path> [, ↩
<domain> [, T/F <secure> [, T/F httponly]]]])
# set the session cookie parameters
T/F session_set_save_handler("open", "close", "read", ↩
"write", "destroy", "gc") # sets user-level session ↩
storage by same-named functions
T/F session_unregister(<name>)
# unregister a global variable from the current session
void session_unset() # free all session variables
void session_write_close()
# write session data and end session
```

Error Handling (6)

```
try {# PHP 5
  <statements that may cause error>;
} catch(<Exception Class> $exception_name){
  <statements to execute when error is caught>;
}# end catch
T/F trigger_error( 'error_msg' [, int type ] )
trigger_error("Error in ".__METHOD__." ...", ↩
E_USER_ERROR);
# 'Error in function name ...' -> FATAL ERROR
```

Control Structure (7)

Control Structure > If Else (7.1)

```
if (<condition 1>)
  { <statement 1>; }
elseif (<condition 2>)
  { <statement 2>; }
else
  { <statement 3>; }
<condition>? true : false; # "?:" ternary operator
echo ((true ? 'true' : 'false') ? 't' : 'f'); # 't'
```

Control Structure > For Loop/Increments (7.2)

```
for (<initialize>;<condition>;<update>)
{<statements>;}
# numbers from 1 ... 10
for($i = 1; $i <= 10; $i++) echo $i;
for($i = 1; ; $i++) {
  if ($i > 10) break;
  echo $i;
}# end for()
for($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
for($i=0; $i<count($array); $i++)
  echo "\n  <option value='".$i."' ".# compare with $i
  ($someInteger==$i?"selected":"").">".$array[$i];
++$a  # Pre-increment by one: $a = $a + 1
--$a  # Pre-decrement by one: $a = $a + 1
$a++  # Post-increment: Returns $a, THEN $a = $a + 1
$a--  # Post-decrement: Returns $a, THEN $a = $a - 1
```

Control Structure > For Each Loop (7.3)

```
foreach (<array> as [<value> | <key> => <value>])
{<statements>;
  [break;]
  [continue;]
}# end foreach()
$arr = array(1, 2, 3, 4); # PHP 5
foreach ($arr as &$value)
  $value = $value * 2; # $arr is now array(2, 4, 6, 8)
```

Control Structure > While/Do-While Loop (7.4)

```
while (<condition>) # do if 'condition' is TRUE
  {<statements>;}
$i = 1;
while($i <= 10) {
 echo $i++.(($i<=10)?",":"");#if $i≦10 then "," else ""
} # 1,2,3,4,5,6,7,8,9,10
do{<statements>;}# do if 'condition' is TRUE, check at end
while (<condition>);
```

Control Structure > Switch (7.5)

```
switch ($i) { # value or expression
case 0: /* literal or type*/
  echo "i is equal to 0"; break;
case 1:
  echo "i is equal to 1"; break;
default:
  echo "i is neither 0 or 1"; /* a default value */
}# end switch
```

Programming (8)

Programming > Function Structure (8.1)

```
function <functionName>([<parameters>='default'])
{<statements>;
  [return <value>;] // returns + stop function here!!!
  <further statements>
}# end functionName()
@doSomething(<parameter>); # @-supresses error messages
```

Programming > Class Structure PHP5 (8.2)

```
class <className> [<extends baseClass>]
{ [<modifiers*>] [<class member variables>];
  [<modifiers*>] function <functionName>([<parameters>]){
    <statements>;
    parent::function() # call from parent class
    self::function() # call within class itself
  }# end function
}# end class
### *modifiers used in PHP5:
public # can be 'seen' everywhere
protected # limits access to inherited and parent classes
private # limits visibility only to the class
static # is static, i.e. not changable
```

Because static methods are callable without an instance of the object created, the pseudo variable $this is not available inside the method declared as static. Static properties cannot be accessed through the object using the arrow operator ->.

```
final # prevents child classes from overriding a method
interface # a class with no data members, contains only↩
 member functions
interface employee{
  function setData($empName,$empAge);
  function outputData();}
class payment implements employee{
  function setData($empName,$empAge) {/* Functionality */}
  function outputData()  { echo "Inside payment Class";}
}# end class payment
abstract /* defines the basic skeleton for the class, ↩
contains attributes and methods changed/used by an ↩
extended class */
```

Programming > Classes PHP5 > Declare/Use (8.2.1)

```
$objName = new className([<construnctor>]);
$objNameSibling = clone $objName; # a clone
mixed $objName->funcName(); # call to class function
$objAssigned   =  $objName;
$objReference  =& $objName; # reference = same content
mixed className::funcName(); # Static call without instance
/* changing variables */
  $objName->var = '$objAssigned will have this value';
  $objName = null; # $instance and $reference become null
class SimpleClass{# declarations
  public $var = 'a default value';
  // private $var4 = self::myStaticMethod();
  public function displayVar() {
    echo $this->var; # show the within-class value
  }// end method displayVar
}// end SimpleClass
$objName = new SimpleClass();
$objName->displayVar();# 'a default value'
$objName->var='another value';
$objName->displayVar();# 'another value'
var_dump($objName); # ask for infos
# object(SimpleClass)#1 (1) {
#   ["var"]=>
#   string(13) "another value"
# }
### call another class statically with ::
class A{# from http://www.php.net/manual/en/
  public function TestFunc(){
    return $this->test." to call ".__CLASS__;}
}# end class A()
class B{
  public $test;
  public function __construct(){
    $this->test = "Nice trick from ".__CLASS__;}
  public function GetTest(){return A::TestFunc();}
}# end class B()
$objB = new B;
echo $objB->GetTest(); # Nice trick from B to call A
```

Programming > Classes PHP5 > Store/Recover Obj. (8.2.2)

```
### serialize objects -> storable strings
class test{public $var = 'Variable';}
  echo serialize(new test());
// O:4:"test":1:{s:6:"strVar";s:8:"Variable";}
### unserialize (restore) objects with its properties
$strObj = 'O:4:"test":1:{s:6:"strVar";s:8:"Variable";}';
$obj = unserialize($strObj);
echo $obj->strVar;# Variable
### serialize but clean a objects __sleep()
class SerializeTest {
  public $var  = 'Variable';
  public $var1 = 'Variable 1';
  private $resource = 'Resource id #1'; //a ressource
  function __sleep() {#saves $var & $var1, not ressource
    return array('var', 'var1'); # must be array!!
    /* return get_object_vars($this); */
    /* for all variables ^^^ instead */}
}# end class SerializeTest()
$objSerialize = new SerializeTest();
echo serialize($objSerialize); #call to  __sleep()
#O:13:"SerializeTest":2:{s:3:"var";s:8:"Variable";s:4: ↩
"var1";s:10:"Variable 1";}
```

In __sleep() was specified, to save only $var and $var1 to be saved for a subsequent access.

```
### unserialize and restore things  __wakeup()
class SerializeTest{
  public $var  = 'Variable'; public $var1 = 'Variable 1';
  private $resource = 'Resource id #1';
  function __sleep()
    {return array('resource');/* save only $resource */}
  function __wakeup()
    {echo 'Connect for instance to DB '.$this->resource;}
}# end class SerializeTest()
```

Programming > Classes PHP5 > Magic Functions (8.2.3)

```
### __autoload() automatically done or loaded
# anywhere before the classes:
function __autoload($className)
  {require_once '/pathToClass/'.$className.'.inc';}
$obj =new MyClass1();# loads '/pathToClass/MyClass1.inc'
$obj2=new MyClass2();# loads '/pathToClass/MyClass2.inc'
### __construct: automatically at initiation
class greetings{ # automatically at initiation:
  function __construct($name) {echo "Hello $name!";}
}# end class greetings()
$objGreet = new greetings("Robert"); # Hello Robert!
### __destruct: automatically at termination
class MyDestructableClass{
  public $name;
  function __construct() {
    print "In constructor done\n";
    $this->name = "MyDestructableClass";}
  function __destruct() {
    print "While destroying " . $this->name . "\n";}
}# end class MyDestructableClass()
$obj = new MyDestructableClass(); # 'In constructor done'
unset($obj); # While destroying MyDestructableClass
### __clone() identical copy of objects
/* clone objects with 'clone': */
  $copy_of_object = clone $object;
/* or explicit in called method __clone() detailed */
class Animal{# from http://www.hiteshagrawal.com/
  public $name; public $legs;
  function setName($name){ $this->name = $name; }
  function setLegs($legs){ $this->legs = $legs; }
  function __clone()
    { echo "<br>Object Cloning in Progress"; }
}# end class Animal()
$tiger = new Animal(); $tiger->name = "Tiger";
$tiger->legs = 4; $kangaroo = clone $tiger;
echo "<br>".$kangaroo->name."---".$kangaroo->legs;
  // Tiger---4 (here identical)
$kangaroo->name = "Kangaroo"; $kangaroo->legs = 2;
echo "<br>".$tiger->name."---".$tiger->legs;
echo "<br>".$kangaroo->name."---".$kangaroo->legs;
  // Object Cloning in Progress
  // Tiger---4
  // Kangaroo---2 (here now modified)
### __toString() returns a string from an object
function __toString()
  { /*... statements */}
```

In PHP5 get called while trying to print or echo the class objects. This method can be used print all the methods and attributes defined for the class at runtime for debugging. Also this method can be used to give error message while somebody tries to print the class details (http://www.hiteshagrawal.com/).

```
### __unset() check for clear an undeclared data member
function __unset($var){
   /* $var is a Variable that unset is supposed to ↩
check. Error message or error-handling can be putted ↩
here. */}
### __isset() check for undeclared data member
function __isset($var){
   /* $var is a Variable that isset is supposed to ↩
check. Error message or error-handling can be putted ↩
here. */}
### __get() check for undeclared or undefined attributes
function __get($var){
   /* $var is a Variable that checked, if able to get. ↩
Error message or error-handling can be putted here. */}
### __set() check for undeclared or undefined attributes
function __set($data,$value)
  {/* $data - holds the name of the undefined ↩
attributes. $value - holds the value assigned to the ↩
undefined attributes. */}
### __call() check for undefined methods ###
function __call($data,$argument)
  { /* $data holds the name of the undefined method ↩
getting called. $argument holds the argument passed to ↩
the method. */ }
```