PHP Quick Reference

Version

PHP 5.1.2
PHP 7.3.8

Escape from HTML

Use `<?php ... ?>`

Any text outside of `<?php ... ?>` will get served as is.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dt
d">
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="en">
<head>
  <title>helloWorld PHP</title>
</head>
<body>
  <?php echo '<p>Hello World PHP</p>'; ?>
</body>
</html>
```

Use XHTML `<pre>` for debuging exact output.

```
<pre>
<?php
echo '<p>Hello World PHP</p>';
echo $_SERVER['SERVER_NAME']."\n";
?>
</pre>
```

Functions can wrap HTML output that exists outside of php escaping, `<?php … ?>`

```
<?php function outputHtml() { ?>

<p>Hello world.</p>

<?php
};
outputHtml();
?>
```

' Hello world. ' lives outside the PHP escaping.

End of Statement

Semicolons terminate ";"

A statement can be spread over multiple physical lines.

```
file_put_contents($outputFileName,
                  $dataToFile);
```

Last line before closing ?> doesn't need ";"

```
<?php echo 'This is a test' ?>
```

The closing tag of a PHP block at the end of a file is optional.

```
<?php echo 'We omitted the last closing
tag';
```

This may be desirable in included files.

PHP Manual

Case

Variable & Constant names are case-sensitive.

http://www.php.net/manual/en/language.variables.php
http://www.php.net/manual/en/language.constants.php

Boolean values are case-insensitive.

http://www.php.net/manual/en/language.types.boolean.php

NULL is case-insensitive.

http://au3.php.net/manual/en/language.types.null.php

Array string literal indexes are case sensitve.

```
$beliefs["You"] = "cynic";

// Access Right
$beliefs["You"] = "cynic";

// Access wrong
$beliefs["you"] = "cynic";
```

Comments

One line: `// or #`

Multiline: `/* ... */`

Class and functions: `/** ... */`

Reflection only picks up "bookend" comments starting with two asterisks.

Sever Configuration

The server is configured via php.ini. You may like to use the runtime variable `ini_set` instead.

```
ini_set("sendmail_from",
"jlrbentley@yahoo.com.au");
```

This allow portability. You may not have access to the server on which your PHP files run.

On your development server install and configure a debugger. Use Xdebug.

```
To install and configure XDebug

* From http://www.xdebug.org/ download the
windows module,
eg
http://www.xdebug.org/link.php?url=xdebug200
rc3-521-win

* Copy the xdebug windows module to your PHP
extension directory.
eg. php_xdebug-2.0.0rc3-5.2.1.dll to
C:\Php\ext\php_xdebug-2.0.0rc3-5.2.1.dll

* In php.ini add:
;****** Added by John B. for xdebug.
; Basics
zend_extension_ts="C:/Php/ext/php_xdebug-
2.0.0rc3-5.2.1.dll"
xdebug.show_local_vars=1

; For use with a client debugger such
; as tsWebEditor
; xdebug.remote_enable=On

; Profiling
xdebug.profiler_output_dir =
"C:/Data/ZTemp/PhpXdebugProfiles"
xdebug.profiler_output_name = "timestamp"
; xdebug.profiler_enable = 1

* Restart Apache
```

Compression and Output Buffering

By default the best way to speed up your page delivery (and execution speed) is to use Apache Compression (mod_deflate) with PHP zlip.output_compression (from .htaccess).

- Using Apache compression (mod_deflate) should be on for .html, .css, and .js files anyway. It doesn't harm .php files that also have some kind of compression (eg zlip.output_compression).

- For compression via PHP using zlip.output_compression is better than `ob_start()` or `<?php ob_start("ob_gzhandler"); ?>` as it sends parts of a page, compressed, once the buffer is reached (4KB by default). This slows down the total load time compared to the other types of PHP compression but reduces the user's perception of latency.

- Setting zlip.output_compression from the Apache config file .htaccess has the advantage of not having to code each individual .php file.

Turn Apache Compression on (mod_deflate)

```
-- In Apache 2.x httpd.cong:
LoadModule deflate_module
modules/mod_deflate.so

# Debug ...
DeflateFilterNote ratio
LogFormat "%v %h %l %u %t \"%r\" %>s %b
mod_deflate: %{ratio}n pct." deflate
CustomLog logs/deflate.log deflate


-- In .htacess (or server config, virtual
host, directory)
AddOutputFilterByType DEFLATE text/html
text/plain text/xml text/css application/x-
javascript text/javascript


-- Restart Apache.
```

To turn zlib.output_compression on in an Apache (as for V1.3) httpd.conf or .htaccess file:

```
<FilesMatch "\.(php|html?)$">
# turn it on with the buffer set to 2K
# php_value zlib.output_compression 2048
# or just turn it on using php_flag
php_flag zlib.output_compression On
</FilesMatch>
```

PHP Manual > *ob_gzhandler* > *User Notes* > *nospam at 1111-internet dot com (08-Mar-2003 11:23)*

Turn output buffering on to speed up page delivery to the browser. (But compression preferred)

```
// At the top of your page.
// This is best for portability
ob_start();

// On your own production server, in Php.ini
// Turn output buffering on for every page.
output_buffering = 4096
```

Compression of PHP pages regardless of whether Apache has compression (eg via mod_deflate).

```
// At the very top of your XTHML
// Above the DOCTYPE
<?php ob_start("ob_gzhandler"); ?>
```

This achieves the same server speed as output buffering but sends a compressed page. Therefore, overall speed saved.

Uses the zlip module.

Compression and output buffering comparision:

```
// From Local Server.
```

	Apache Compression Off	Apache Compression On (mod_deflate)
Normal PHP Page	93KB 18ms PHP 980ms Total	2KB 18ms PHP 750ms Total.
ob_start()	93KB 10ms PHP 906ms Total	2KB 10ms PHP 734ms Total
<?php ob_start("ob_gzhandler"); ?>	2KB 10ms PHP 750ms Total	2KB 10ms PHP 734ms Total
zlip.output_compression from php.ini.	3KB 21ms 765ms Total	3KB 20ms PHP 797ms Total
zlip.output_compression from .htaccess.	3KB 19ms PHP 766ms Total.	3KB 20ms 765ms Total

```
// From Remote Server (using different
timing functions so don't compare with
previous table).
```

	Apache Compression Off	Apache Compression On (mod_deflate)
Normal Page	93KB 1606 ms PHP 2578 ms Total	2 KB 3 ms PHP 953 ms Total
ob_start()	93 KB 1 ms PHP ??? 2391 ms Total	2 KB 1 ms 813 ms Total
<?php ob_start("ob_gzhandler"); ?>	2 KB 1 ms PHP ??? 831 ms Total	2 KB 1 ms PHP 875 ms Total
zlip.output_compression from php.ini.		
zlip.output_compression from .htaccess.	2 KB 3 ms PHP 828 ms Total	2 KB 3 ms PHP 984 ms Total

Differences between zlib.output_compression and ob_gzhandler:

zlib.output_compression runs in parallel with script execution - it begins compressing as soon as it receives any output from your script, and sends data to the client each time its buffer (4K by default) gets full. ob_gzhandler (actually 'ob_start("ob_gzhandler");') will not start compressing until the script flushes (or, usually, exits), and will in turn send the entire compressed document at once - which makes it more susceptible to causing a perception of latency.

Advantage: zlib.output_compression

On the other hand, ob_gzhandler gives you script-level control allowing you to use it selectively or to unset it after setting it in certain cases. Despite some documentation to the contrary, zlib.output_compression does not appear to be able to be set or unset on a script level; you must instead set it globally (in php.ini) or in your webserver configs or .htaccess files, possibly using FilesMatch-type mechanisms to control which scripts it will or won't apply to - which could get unwieldy for large projects - particularly those which employ PHP to produce images and other non-text output in addition to normal text output.

Advantage: ob_gzhandler

Net advantage: depends on your particular needs. I'm trying zlib.output_compression for now, but I miss the flexibility that ob_gzhandler would provide.

PHP Manual > ob_gzhandler > User Notes > nospam at 1111-internet dot com (08-Mar-2003 11:23)

Internal Libraries

In general use `require_once` rather than `require` or `include_once`

```
require_once 'Calendar/Month.php';
```

To make paths relative to the called file, not the calling file use:

```
// References relative to this called file,
not the calling file

// Subfolder of this, called, file
require_once
dirname(__FILE__).'/./fileE.php';
require_once
dirname(__FILE__).'/foo/fileD.php';

// Subfolder of this, called, file going up.
require_once
dirname(__FILE__).'/../subfolder02/fileC.php
';
```

Deprecated: To make paths relative to the called file, not the calling file use `realpath`:

```
require_once realpath(dirname(__FILE__) .
'/' . './foo/fileC.php');
```

External Libraries

PEAR, PHP Extension and Application Repository, is THE external library for PHP.

To install PEAR and run on windows:
1. Your PHP installation properly already has PEAR setup files.
2. Run C:\Php\go-pear.bat
3. Ensure that PHP.ini is updated with include_path=".;C:\Php\pear".

To install a PEAR Package:
1. C:\Php\peardev.bat for a list of commands
2. C:\Php>peardev.bat install -a Calendar

Include package and use:

```
        require_once 'Calendar/Month.php';

        $Month = new Calendar_Month(2003,
10); // October 2003

        $Month->build(); // Build the days
in the month

        // Loop through the days...
        while ($Day = $Month->fetch()) {
            echo $Day->thisDay().'<br />';
        }
```

http://pear.php.net/

Variables

Basic

Declare with dollar sign($).

```
$person = 'trent';
```

Declare by reference with ampersand (&)

```
$foo = 'Bob';  // Assign the value 'Bob' to
$foo
$bar = &$foo;  // Reference $foo via $bar.
```

Variable scope: global (outside a function) V local (inside a function).

http://www.php.net/manual/en/language.variables.scope.php

The `global` keyword makes global variables accesible within a function.

```
$a = 5;

function getNum(){
  global $a;
  return $a;
}

echo getNum(); // Returns 5
```

The `global` keyword preface variables on one line, makes them all those variables accessible as globals, to the containing function.

```
function getNum(){
  global $a, b$;
  return $a.' '.$b;
}
```

A static variable, only useful for local variables, does not lose its value when program execution leaves this scope.

```
function Test () {
   static $a = 2;
   echo $a;
   $a++;
}
echo Test();   // 2
echo Test();   // 3
```

It is not necessary to initialize variables in PHP however it is a very good practice.

Uninitialized variables have a default value of their type - FALSE, zero, empty string or an empty array.

http://www.php.net/manual/en/language.variables.php

Variable variables

```
$a = "hello";
$$a = "world";
```

```
echo "$a ${$a}\n"; // hello world
echo "$hello\n";  // world
```

Predefinded (Autoglobals/Superglobals)

'Autoglobals' or 'superglobals' are predefined arrays containing variable values. They are available everywhere in the script.

```
echo $_SERVER['SERVER_NAME'];
```

$GLOBALS	Contains a reference to every variable which is currently available within the global scope of the script.
$_SERVER	Variables set by the web server or otherwise directly related to the execution environment of the current script.
$_GET	Variables provided to the script via URL query string.
$_POST	Variables provided to the script via HTTP POST.
$_COOKIE	Variables provided to the script via HTTP cookies.
$_FILES	Variables provided to the script via HTTP post file uploads.
$_ENV	Variables provided to the script via the environment.
$_REQUEST	Variables provided to the script via the GET, POST, and COOKIE input mechanisms
$_SESSION	Variables which are currently registered to a script's session.

Variables from outside PHP

Use either $_GET or $_POST, depending on form method. Use $_REQUEST, regardless of form method.

```
$area    = $_POST['cboAreaID'];
$launchName = $_POST['txtLaunchName'];
$altitude   = $_REQUEST['txtAltitude'];
$windDirection = $_POST['txtWindDirection'];
$rating = $_POST['txtRating'];

echo <<<EOT
<pre>
Area: $area
LaunchName: $launchName
Altitude: $altitude
Wind Direction: $windDirection
Rating:  $rating
</pre>
```

import_request_variables() has been removed as of 5.4.0

Set Cookies easily:

```
// Expires in one hour;
setcookie("MyCookie", 'tasty', time()+3600);
```

http://www.php.net/manual/en/language.variables.external.php

Constants

Basic

A constant: always has global scope; set with `define();` Reference without `$;` is Case-Sensitive.

```
define("myPhrase", "Hello world.");
echo myPhrase; // outputs "Hello world."
echo MYPHRASE; // illegal
```

http://www.php.net/manual/en/language.constants.php

You can define a constant by using the define()-function or by using the const keyword outside a class definition as of PHP 5.3.0.

```
const CONSTANT = 'Hello World';
```

https://www.php.net/manual/en/language.constants.syntax.php

Predefined

"Predefined Constants" exist as "core", "standard" or from optionally loaded extensions.

```
// Core
echo PHP_VERSION."\n";

// Standard
echo LC_TIME."\n";

// Extension
echo MYSQL_ASSOC."\n";
```

To get a list of predefined constants.

```
print_r(get_defined_constants());
```

Magic

Magic Constants have values that change depending on context. Case-insensitive.

```
__LINE__, __FILE__, __DIR__ __FUNCTION__,
__CLASS__, __METHOD__,
__COMPILER_HALT_OFFSET__
```

__DIR__ for relative directory handling

For robust relative directory handling use something like

```
require_once __DIR__.
'./../standardLibrary/php/string.php';
```

C:\Users\John\Documents\Sda\Code\web\Libraries\Html5Library\web-prototyper\100-tech-base\designOptions\design-options.phps

Operators

Arithmetic		Bitwise	
-$a	Negation	$a & $b	And
$a + $b	Addition	$a | $b	Or
$a - $b	Subtraction	$a ^ $b	Xor
$a * $b	Multiplication	~ $a	Not
$a / $b	Division	$a << $b	Shift left
$a % $b	Modulus	$a >> $b	Shift right

Comparison

$a == $b	Equal	
$a === $b	Identical	(Type and value match)
$a != $b	Not equal	
$a <> $b	Not equal	
$a !== $b	Not identical	(value or type mismatch)
$a < $b	Less than	
$a > $b	Greater than	
$a <= $b	Less than or equal to	
$a >= $b	Greater than or equal to	

The expression (expr1) ? (expr2) : (expr3) evaluates to expr2 if expr1 evaluates to TRUE, and expr3 if expr1 evaluates to FALSE.

```
echo $a > 10 ? "big": "small";
```

Logical

$a and $b	And
$a or $b	Or
$a xor $b	Xor
! $a	Not
$a && $b	And
$a || $b	Or

Increment/Decrement

++$a	Pre-increment
$a++	Post-increment
--$a	Pre-decrement
$a--	Post-decrement

The operators `and, or, &&, ||` use shortcut evaluation.

Assignment

=	$a = 5
$a += 5	$a = $a + 5
Other Arithmentic: -= *= /= .= %=	
Other Bitwise: &= |= ^= <<= >>=	
$a .= "There"	$a = $a."There"
Array Union	

String Concatenation use dot, .

```
$a = "Hi "."There";
```

Array

$a + $b	Union	Union of $a and $b.
$a == $b	Equality	TRUE if $a and $b have the same key/value pairs.
$a === $b	Identity	TRUE if $a and $b have the same key/value pairs in the same order and of the same types.
$a != $b	Inequality	TRUE if $a is not equal to $b.
$a <> $b	Inequality	TRUE if $a is not equal to $b.
$a !== $b	Non-identity	TRUE if $a is not identical to $b.

The array union operator, +, appends the right handed array to the left handed. For any duplicate keys the left handed array takes precedence.

Object instance operator, `instanceof`.

```
class A { }
$thing = new A;
if ($thing instanceof A) {
   echo 'A';
}
```

Error Operator, @, When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.

```
/* Intentional file error */
$my_file = @file ('non_existent_file') or
   die ("Failed opening file: error was
'$php_errormsg'");
```

If the track_errors feature is enabled, any error message generated by the expression will be saved in the variable $php_errormsg.

The Execution operator, the backtick, `, executes a shell command.

```
$output = `dir`;
echo "$output";
```

Disabled when safe mode is enabled or shell_exec() is disabled. Identical to shell_exec().

Datatypes

Datatype	Examples, Value, or Notes	Test
Integer	42	is_int()
Float	3.1415, -3.1E12, .1e12, 2E-12	is_float()
String	"Nice"	is_string()
Boolean	TRUE, FALSE, true, false	is_bool()
Null	null	is_null()
Array		is_array()
Object		is_object()
Resource		is_resource()

There is no Date datatype.

Determine variable types with the specific test functions above and with gettype().

Values

Null

A variable is NULL when: it has been assigned the constant NULL.; it has not been set to any value yet; or it has been unset().

NULL is case-insensitive.

http://au3.php.net/manual/en/language.types.null.php

Empty

These are emtpy:

```
"" (an empty string)
0 (0 as an integer)
"0" (0 as a string)
NULL
FALSE
array() (an empty array)
var $var; (a variable declared, but without
a value in a class)

$var = 0;

// Evaluates to true because $var is empty
if (empty($var)) {
   echo '$var is either 0, empty, or not set
at all';
}
// Evaluates as true because $var is set
if (isset($var)) {
   echo '$var is set even though it is
empty';
}
```

Booleans

True V False

```
false: FALSE, false, 0  (zero), 0.0 (zero) ,
zero-length string (""), "0", an array with
zero elements, an object with zero member
variables (PHP 4 only), NULL (including
unset variables), SimpleXML objects created
from empty tags.

true: non-zero, -1, 1, any resource.
```

Expression	empty()	is_null()	isset()	boolean : if($x)
$x = "";	TRUE	FALSE	TRUE	FALSE
$x = NULL	TRUE	TRUE	FALSE	FALSE
var $x;	TRUE	TRUE	FALSE	FALSE
$x is undefined	TRUE	TRUE	FALSE	FALSE
$x = array();	TRUE	FALSE	TRUE	FALSE
$x = false;	TRUE	FALSE	TRUE	FALSE
$x = true;	FALSE	FALSE	TRUE	TRUE
$x = 1;	FALSE	FALSE	TRUE	TRUE
$x = 42;	FALSE	FALSE	TRUE	TRUE
$x = 0;	TRUE	FALSE	TRUE	FALSE
$x = -1;	FALSE	FALSE	TRUE	TRUE
$x = "1";	FALSE	FALSE	TRUE	TRUE
$x = "0";	TRUE	FALSE	TRUE	FALSE
$x = "-1";	FALSE	FALSE	TRUE	TRUE
$x = "php";	FALSE	FALSE	TRUE	TRUE
$x = "true";	FALSE	FALSE	TRUE	TRUE
$x = "false";	FALSE	FALSE	TRUE	TRUE

http://www.php.net/manual/en/types.comparisons.php

isset() is the logical contrary of is_null().

DataType Conversion

(int), (integer) - cast to integer
(bool), (boolean) - cast to boolean
(float), (double), (real) - cast to float
(string)
(array)
(object)

```
$a = "4.8";
$b = "5.7";
echo (int) $a+ (int) $b; // 9
```

Strings Literals

Single Quotes

Variables will not be expanded when they occur in single quoted strings.

Single quote escape characters.

\'	Single quotes
\\	backslash

Double Quotes

Double Quote escape characters

\n	linefeed (LF or 0x0A (10) in ASCII)
\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\\	backslash
\$	dollar sign
\"	double-quote
\[0-7]{1,3}	the sequence of characters matching the regular expression is a character in octal notation
\x[0-9A-Fa-f]{1,2}	the sequence of characters matching the regular expression is a character in hexadecimal notation

When outputting to a windows file use: \r\n

Otherwise you won't get a new line but a square box where the newline character ought be.

Outputting in an html pre tag.

```
<pre><?phps
$format = "%s\r";
printf($format, 'Cool');
printf($format, 'Thing');
?>
</pre>
```

Heredoc

Heredoc text behaves just like a double-quoted string.

```
echo <<<EOD
Spanning multiple lines
using heredoc syntax:
* Works as for double quoted string.
* No need for '\\n' character at end of
line, except for the last.\n
EOD;
```

Be sure there is no tabs or spaces after the last EOD

```
// Otherwise variables will not work with
angled brackets.
```

Nowdoc

Nowdoc text behaves like heredoc, except no parsing is done. To designate nowdoc single quote the identifier.

```
echo <<<'EOD'
Example of string spanning multiple lines
using nowdoc syntax. Backslashes are always
treated literally,
e.g. \\ and \'.
EOD;
```

https://www.php.net/manual/en/language.types.string.php#language.types.string.syntax.heredoc

String Parsing

Strings will be parsed for extra escapes characters; variables; array values; or object properties - in **double quotes or heredoc format** but not single quotes.

```
$expand = "My string";
echo 'Variables and some escapes do not \n
$expand';
echo "\nVariables and more escapes do \n
$expand\n";
echo <<<EOD
Variables and more escapes do \n $expand
EOD;
```

Strings will be parsed for **extra escapes characters; variables; array values; or object properties** - in double quotes or heredoc format but not single quotes.

```
$beer = 'Heineken';
$fruits = array('strawberry' => 'red',
'banana' => 'yellow');
class Aircraft {
  public $type = 'boeing';
}
$b747 = new Aircraft();

// Simple Syntax
echo "A variable $beer's\n";
echo "An array $fruits[strawberry]\n";
echo "An ojbect $b747->type\n";
```

Simple Syntax is without curly braces {}

Simple Syntax arrays don't use quotes around array keys.

```
echo "An array $fruits[strawberry]\n";
echo <<<EOD
$fruits[strawberry]\n
EOD;
echo "An ojbect $b747->type\n\n";
```

Complex syntax: employ curly braces, {}, to write an expression the same way as you would outside the string

```
echo "Outside String\n";
echo "A variable " . $beer . "'s\n";
echo "An array " . $fruits['strawberry']
."\n";
echo "An ojbect " . $b747->type . "\n\n";

echo "Complex Syntax\n";
echo "A variable {$beer}'s\n";
echo "An array {$fruits['strawberry']}\n";
echo <<<EOD
{$fruits['strawberry']}\n
EOD;
echo "An ojbect {$b747->type}\n";
```

Numbers

When rounding a number avoid using the native function: `round($x, precision)`. Use a custom banker's rounding instead.

The native rounding function rounds away from zero. Banker's rounding reduces errors over many calculations.

Value	Native round()	Custom bankersRound()
4.55	4.6	4.6
4.65	**4.7**	**4.6**
-4.55	-4.6	-4.6
-4.65	**-4.7**	**-4.6**
Rounding Goal	Accuracy	Accuracy
Rounding Direction	Away From Zero	Toward Even (Bankers Round)

Bentley > EcmaScriptLibrary > roundingFunctions.html & Math.js

Date Time

DateTimes are stored in seconds since January 1 1970 00:00:00 GMT. This is known as a "Unix timestamp"

PHP Manual > strtotime

Valid range of Unix Timestamp
Fri, 13 Dec 1901 20:45:54 GMT to
Tue, 19 Jan 2038 03:14:07 GMT

PHP Manual > date()

Date Time values are based on on server settings.

Creation

Input

Input as Local

```
date_default_timezone_set('Australia/Sydney'
);
$takeOffLocalUnix =
mktime(20,30,00,12,25,2006);
// 1167039000

date(DATE_RFC2822, $takeOffLocalUnix);
// Mon, 25 Dec 2006 20:30:00 +1100

gmdate(DATE_RFC2822, $takeOffLocalUnix);
// Mon, 25 Dec 2006 09:30:00 +0000
```

Input as UTC

```
$refuelUtcUnix =
gmmktime(20,00,00,12,25,2006);
// 1167076800

date(DATE_RFC2822, $refuelUtcUnix);
// Tue, 26 Dec 2006 03:00:00 +0700

gmdate(DATE_RFC2822, $refuelUtcUnix);
// Mon, 25 Dec 2006 20:00:00 +0000
```

Parse

Parse as local

```
date_default_timezone_set('Australia/Sydney'
);

$landingLocalUnix = strtotime("2:30 26 Dec
2006");
// 1167075000

date(DATE_RFC2822, $landingLocalUnix);
// Tue, 26 Dec 2006 02:30:00 +0700

gmdate(DATE_RFC2822, $landingLocalUnix);
// Mon, 25 Dec 2006 19:30:00 +0000
```

Parse as UTC

```
date_default_timezone_set('Asia/Bangkok');
$refuelUtcUnix = strtotime("20:00 25 Dec
2006 UTC");
// 1167076800

date(DATE_RFC2822, $refuelUtcUnix);
// Tue, 26 Dec 2006 03:00:00 +0700

gmdate(DATE_RFC2822, $refuelUtcUnix);
// Mon, 25 Dec 2006 20:00:00 +0000
```

Now

```
date_default_timezone_set('Australia/Sydney'
);
date('Y-m-d H:i:s (O)', time());
// 2009-05-24 19:34:14 (+1000)

$nowLocalUnix = time();
// 1148451128

$nowLocalString = date(DATE_RFC2822);
// Wed, 24 May 2006 16:12:08 +1000

$nowUtcUnix = gmmktime();
// 1148451128

$nowUtcString = gmdate(DATE_RFC2822);
// Wed, 24 May 2006 06:12:08 +0000
```

Formats

Full constants with `date()`.

```
date(DATE_RFC2822, time());
// Wed, 24 May 2006 19:42:03 +1000
```

Constant	Value
DATE_COOKIE	Wed, 24 May 2006 16:53:50 EST
DATE_ISO8601	2006-05-24T16:53:50+1000
DATE_RFC850	Wednesday, 24-May-06 16:53:50 EST
DATE_RFC2822	Wed, 24 May 2006 16:53:50 +1000
DATE_W3C	2006-05-24T16:53:50+1000

https://www.php.net/manual/en/class.datetimeinterface.php#datetime.constants.types

Custom constants with date().

```
date('Y-m-d H:i:s', time());
// 2009-05-24 19:34:14
```

Constant	Value
D, d M Y T (O)	Wed, 24 May 2006 EST (+1000)
D, d M Y e (O)	Wed, 24 May 2006 Australia/Sydney (+1000)
D, d M Y e (O) H:i:s	Wed, 24 May 2006 Australia/Sydney (+1000) 19:34:14
H:i	19:34
d M Y H:i	24 May 2006 19:34
Y-m-d	2009-05-24
Y-m-d H:i:s	2009-05-24 19:34:14
Y-m-d H:i:s (O)	2009-05-24 19:34:14 (+1000)

https://www.php.net/manual/en/function.date.php

Arithmetic

Use functions to add and subtract dates don't work on integers directly.

```
// These
$date = strtotime('+7 days', $date);
```

```
$tomorrow  = mktime(0, 0, 0, date("m"),
date("d")+1, date("Y"));

// Not this
$date += 7*24*60*60;
```

That way leap years, daylight savings, and date validation occurs.

PHP Manual > *Date and Time Functions*

PHP Manual > *date*

Using the DateTime Object

C:\Users\John\Documents\Sda\Code\Php\Examples\dates.php
https://www.php.net/manual/en/class.datetime.php

Creation

Input as Local

```
date_default_timezone_set('Australia/Sydney'
);
$datetime = new DateTime('2009-11-13
01:00:00');

echo $datetime->getTimezone()->getName() .
"\n";
// Australia/Sydney

echo $datetime->format('D, d M Y H:i:s e (O)
') ."\n";
// Fri, 13 Nov 2009 01:00:00
Australia/Sydney (+1100)
```

Input as UTC

```
$utcTimeZone = new DateTimeZone('UTC');
$datetime->setTimezone($utcTimeZone);
// or
$datetime->setTimezone(new
DateTimeZone('UTC'));
echo $datetime->getTimezone()->getName() .
"\n";
// UTC

echo $datetime->format('D, d M Y H:i:s e (O)
') ."\n";
// Thu, 12 Nov 2009 14:00:00 UTC (+0000)
```

CreateFromFormat

```
$myDateTime =
DateTime::createFromFormat('!Y', '2016');
echo '$myDateTime: '. $myDateTime-
>format(DateTimeInterface::ISO8601)."\n";
// $myDateTime: 2016-01-01T00:00:00+1100
```

Now

```
$datetime = new DateTime();
```

DateTimeWithPrecision

Use DateTimeWithPrecision, a class by John Bentley, which outputs in ISO8601 format and handles year only and year-month date.

```
echo new
Softmake\DateAndTime\DateTimeWithPrecision('
2004-06');
// 2004-06
```

C:\Users\John\Documents\Sda\Code\Php\Libraries\StandardLibrary\dateAndTime.php

Control Structures

Conditionals

```
if (condition) {
} elseif (condition) {
} else {
}

switch ($i) {
  case 0:
    ...
    break;
  case 1:
    ...
    break;
  default:
     ...;
}
```

Loops

```
while (condition) {
}

do {
} while (condition);

for ($i = 1; $i <= 10; $i++) {
}
```

Array iteration basic

```
for ($i = 0; $i < count($beliefs); $i++) {
  echo $beliefs[$i];
}

for ($i = 0; $i < count($array); $i++) {
    echo $array[$i]['filename'];
    echo $array[$i]['filepath'];
}
```

Break and continue

break ends execution of the current for, foreach, while, do-while or switch structure.

```
break 2;  // Exit the switch and the while.
```

Accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of. **continue** is used within looping structures, including **switch**, to skip the rest of the current loop iteration and continue execution at the condition evaluation.

Continue

```
continue 2;  // Exit the switch and the
while.
```

Accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of **return** immediately ends execution of the current function, and returns its argument as the value of the function call. return() will also end the execution of an eval() statement or script file.

For each

Iterate over arrays with amp, &, to change.

```
$arr = array("first" => 1, "second" => 2,
"third" => 3);
foreach ($arr as &$value) {
   $value *= 2;
}
```

Iterate over arrays without amp, &, just goes through a copy.

```
foreach ($arr as $value) {
   echo "$value\n";
}
```

Iterate over the key as well as value.

```
foreach ($arr as $key => $value) {
   echo "$key: $value\n";
}
```

Objects used for iteration will go through visible properties.

```
$class = new MyClass();

foreach($class as $keyName => $value) {
   echo "$keyName => $value\n";
   // Refer to property
   echo var_dump($class->$keyName);
}
```

Alternative Syntax

if, while, for, foreach, and switch can have alternative syntax. Change opening brackets, {, to colon, :, and prefix "end" to a closing control structure term.

```
$cool = false;
$nice = false;

if ($cool):
   echo "right on.";
elseif ($nice):
   echo "nicely";
else:
   echo "ordinary";
endif;
```

Inline if

Two methods of inline if. No braces and operator.

```
// Method 1: no braces
if ($temp < 20) echo 'cool';

// Method 2: ()?: operator
echo ($a > 10) ? "big": "small";
```

Unusual

require() and **include()** are identical except **include()** produces a Warning while **require()** results in a fatal error.
require_once() and include_once() only load the file once if already loaded.

```
require_once 'purePhp.php';
```

Ticks

```
// A function that records the time when it
is called
function profile()
{
   static $i = 0;
   echo "tick". $i++;
}

// Set up a tick handler
register_tick_function("profile");

// Run a block of code, throw a tick every
2nd statement
declare(ticks=2) {
  1;
}
```

Using ticks on Windows, running Apache as a php-module, crashes Apache and will not work. You have to use the CGI-version to use ticks on Windows.

http://www.php.net/manual/en/control-structures.declare.php

http://bugs.php.net/bug.php?id=26771

Switch

In a switch statement match multiple cases like this:

```
switch ($i) {
  case 0:
  case 1:
  case 2:
     echo "i is less than 3 but not
negative";
     break;
  case 3:
     echo "i is 3";
}
```

Switch with comparison operators.

```
switch (true) {
  case $count == 1:
    break;
  case $count == 2:
    break;
  case $count > 2:
    break;
  default:
    throw new Exception("\$creatorsArray
count unexpected: " .
count($creatorsArray));
}
```

https://stackoverflow.com/a/24813225/872154 , " Using comparison operators in PHP switch", Answer by Konr Ness

Enumerated Constants (Simulated Enum)

Use a class with constants.

```
class ThemeEnum {
  public const Base = 0;
  public const Example = 1;
};

$activeTheme = ThemeEnum::Base;

if ($activeTheme === ThemeEnum::Base)
```

Arrays

Zero based; can contain different datatypes; can grow in size at any time.

```
var_dump(array("me",true,12));
```

Simple

Declare empty

```
$futureCauses = array();
```

Declar and assign values

```
// First form.
$beliefs = array("liberal",12,true);

// Second form.
$beliefs[] = "nihilism";
// Takes the next highest numeric index, 0
if none available.
```

Declare and assigin, keys and values.

```
// First form.
$beliefs = array("Me" =>"liberal", 'you'
=>12, 5 => true);

// Second form.
$beliefs[12] = "nihilism";
$beliefs["jane"] = "nihilism";
```

A key may be either an integer or a string. A value can be of any PHP type.

Arrays will always have a numeric index available.

```
// $beliefs[1] == 12.
```

You should always use quotes around a string literal array index.

```
// This
$beliefs["jane"] = "nihilism";

// Not this
$beliefs[jane] = "nihilism";
```

PHP Manaul > Arrays > Array Do's and Don'ts

Delete Entry

```
unset($beliefs["you"]);
// Arrary not reindexed.

$beliefs = array_values($beliefs);
// Keys removed and array reindexed
numerically.
```

Delete all entries

```
$beliefs = null; // Completely destroy
$beliefs = array(); // reset to empty array.
```

Access

```
$beliefs[2];
$beliefs["you"];
$beliefs['you'];

// Array string literal indexes are case
sensitve.
// Wrong:
$beliefs['You'];
```

Array of Arrays

Create (Optional)

```
$fleetArrayOfArrays = array();
```

Assign

```
$fleetArrayOfArrays[0] = array("B734",
"Boeing", "737-400", 168);
$fleetArrayOfArrays[1] = array("C182RG",
"Cessna", "182RG", 4);
$fleetArrayOfArrays[2][] = "BE58";
$fleetArrayOfArrays[2][] = "Beechcraft";
$fleetArrayOfArrays[2][] = "Baron 58";
$fleetArrayOfArrays[2][] = 6;

// or
$fleetArrayOfArrays =
array( array("B737", "Boeing", "737-400",
168),
       array("C182RG", "Cessna", "182RG",
4),
       array("BE350", "Beechcraft", "King
Air 58", 6) );
```

Access

```
$counti = count($fleetArrayOfArrays);
for ($i = 0; $i < $counti; $i++){
  echo "\n";
  $countj = count($fleetArrayOfArrays[$i]);
  for($j = 0; $j < $countj; $j++) {
    echo $fleetArrayOfArrays[$i][$j]." ";
  }
}

// or
foreach ($fleetArrayOfArrays as $aircraft) {
  echo "\n";
  foreach ($aircraft as $value) {
  echo $value." ";
  }
}

// or
foreach ($this->exifInfo as $header =>
$section) {
  foreach ($section as $name => $val) {
    $output .= "$header.$name: $val\r";
  }
}
```

Array of Objects

Object definition

```
class Aircraft {
  public $typeCode = "B734";
  public $ceiling = 34000;
  public $model = "";
  public $seats = 0;

  public function __construct($typeCode =
"B734",
                              $ceiling =
34000,
                              $model,
                              $seats) {
    $this->typeCode = $typeCode;
    $this->ceiling = $ceiling;
    $this->model = $model;
    $this->seats = $seats;
  }

  public function __toString() {
    return "Aircraft: $this->typeCode,
$this->model, $this->ceiling, $this->seats
\n";
  }
}
```

Create

```
$fleet = array();
```

Assign

```
$fleet[] = new Aircraft("B734", "Boeing",
"737-400", 168);
$fleet[] = new Aircraft("C182R", "Cessna",
"182RG", 4);
$fleet[] = new Aircraft("BE58",
"Beechcraft","Baron 58", 6);
```

Access

```
foreach ($fleet as $aeroplane) {
  echo "Type Code: $aeroplane->typeCode ";
  echo $aeroplane; // calls __toString.
}
```

See > Bentley's > Code\Php\Examples\arrays.php

Useful Array Functions

Array to string: implode.

```
$array = array('lastname', 'email',
'phone');
$arrayOutput = implode("\n", $array);
echo $arrayOutput."\n";

// Outputs
lastname
email
phone
```

String to Array: explode.

```
$str = 'one|two|three|four';

// Simple
$numberArray = explode('|', $str);
print_r($numberArray);
```

```
// Outputs
Array
(
    [0] => one
    [1] => two
    [2] => three
    [3] => four
)
```

Functions

Creating

Normal Function

```
function functionName([arg1] ... [,argN]) {
  statements(s)
}
```

Function variables

```
function foofunc() {
  echo "foo";
}
function bar() {
  echo "bar";
}

$myFunctionVar = 'foofunc';
$myFunctionVar();
$myFunctionVar = 'barfunc';
$myFunctionVar();
echo "\n";
// Output: foobar
```

Anonymous Function

```
$add = create_function('$a,$b', 'return $a +
$b;');
echo "\$add(): ".$add(2, 3)."\n";
```

PHP Manual > *create_function*

Conditional functions.

```
$debug = true;
if ($debug) {
 function debugOutput()
 {
   echo "Program is groovy\n";
 }
}
if ($debug) debugOutput();
```

Calling Language Constructs

Language constructs are not functions and therefore the parentheses, (), are optional.

```
// Both OK
echo ("hi ");
echo "there";
```

PHP Manual > *echo*

Returning

The function is terminated on the first return encountered.

```
function returnValue () {
  return 5;
  return 6;
}
echo returnValue();
// 5 is returned.
```

return() will also end the execution of an eval() statement or script file.

PHP Manual > *return*

A function is not required to return a value.

```
function doStuff () {
  echo "stuff done";
}
doStuff();
```

Function Arguments

General

To pass an argument by reference prepend an ampersand (&) to the argument name.

```
function add_some_extra(&$string)
{
   $string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra($str);
echo $str;
// outputs 'This is a string, and something
extra.'

add_some_extra(&$str); // Deprecated.
```

PHP Manual > *Function Arguments*

Default argument values.

```
function output($msg = "good", $height = 10,
$peace = 'nice') {
  echo "$msg $height $peace\n";
}
output();  // good 10 nice
output("bad");  // bad 10 nice
```

Make Optional function arguments by setting to null

```
function fun ($arg1, $argOptional = null) {
  echo "$arg1\n";
  if (!is_null($argOptional)) {
    echo "$argOptional\n";
  }
}
fun ("Now");
```

You must supply prior arguments in order to avail yourself of default argument value

```
function output($msg = "good", $height = 10,
$peace = 'nice') {
  echo "$msg $height $peace\n";
}
output($msg = 'bad', $height = 11 );
// bad 11 nice

output($msg = 'bad', 11 );
// bad 11 nice
```

Named arguments

```
$db->selectIntoHtmlCombobox($sql,
            $displayIDField = true);
```

The arguments must be in the order of the parameter.

Parameter Arrays

Parameter Arrays (variable length argument lists). Use the array returned by **func_get_args()** to access all arguments passed to a function regardless of whether there is explicit parameters or not. Use also **func_num_args()**; and **func_get_arg(x)**.

```
function foo($first, $second) {
   $numargs = func_num_args();
   echo "Number of arguments: $numargs\n";

   echo "Second Argument: $second \n";
   echo "Second Agrument:
".func_get_arg(1)."\n";
   if ($numargs >= 3) {
       echo "Third Argument:
".func_get_arg(2)."\n";
   }

   $arguments = func_get_args();
   foreach ($arguments as $argument) {
      echo $argument."\n";
   }
}
foo(1, "bill", 3, "jane");
```

Php Manaul > *func_get_args*

If you want an array to be interpreted as an argument list use `call_user_func_array`:

```
$places = array("New Zealand" ,"Thailand",
"France");
call_user_func_array('foo', $places);
```

PHP Manual > *call_user_func_array*

Variable length argument list, with fixed arguments. Just pass an array on the end.

```
function showArgList($arg01, array
$argArray) {
  $i = 0;
  echo "$i: $arg01\n";
  $i++;
  foreach ($argArray as $value) {
    echo "$i: $value\n";
    $i++;
  }
}

$places = array("New Zealand" ,"Thailand",
"France");
showArgList('Australia', $places);

// Outputs
0: Australia
1: New Zealand
2: Thailand
3: France
```

Parameter type hinting

You can force the datatypes of function parameters to be a specific object or an array.

```
class Aircraft {
  public $model = "";

  public function __construct($model) {
    $this->model = $model;
  }
}

$cessna = new Aircraft("cessna 172");
$persons = array("bill","jack","okcana");

function output(Aircraft $theAircraft,
               array $friends) {
  echo "Aircraft: ".$theAircraft-
>model."\n";
  echo "Friends:
".implode(",",$friends)."\n";
}

output($cessna, $persons);
```

PHP Manual > *Type Hinting*

Classes and Objects (PHP 5)

Define and Create

Define Custom Object.

```
class Aircraft {
  public $typeCode = "";
  public $ceiling = 34000;
  public $model = "";
  private $seats = 50;
  const airline = 'qantas';
```

```
  public function __construct($typeCode =
"B734",
                              $ceiling,
                              $model) {
    $this->typeCode = $typeCode;
    $this->ceiling = $ceiling;
    $this->model = $model;
  }

  public function __destruct() {
    // $this alone calls __toString()
    echo "$this... has been Destroyed.";
    echo "<br />";
    echo $this->output();
    echo " Goodbye ";
  }

  public function __toString() {
    // $this can refer to public and private
properties
    return "Aircraft: $this->typeCode,
$this->model, $this->ceiling, $this->seats
\n";
  }

  public function output() {
    echo "$this->typeCode $this->ceiling
".self::airline."\n";
  }

  public function authority() {
    echo "CAA";
  }
}
```

The `$this` psuedo-variable is available inside a class' method. Generally, `$this` is a reference to the calling object (usually the object to which the method belongs).

$this can refers to properties or methods of the calling object. These properties or methods can be public or private.

Create

```
$boeing747 = new Aircraft("B744", 36000);
```

Destroy

```
$boeing747 = NULL;
```

Calling

```
echo $boeing747; // Call __toString()

// Property
echo "$boeing747->typeCode\n";

// Constant (from outside class)
Aircraft::airline;

// Method
$boeing747->output();

// A static call.
Aircraft::authority()
```

Define default values for an object. Constructorless technique.

```
class Aircraft {
  public $type = "B734";
  public $ceiling = 34000;

  public function output() {
    echo "$this->type $this->ceiling\n";
  }
}

$boeing747 = new Aircraft();
$boeing747->output();
```

Define default values for an object. With Constructor.

```
class Aircraft {
  public $type = "";
  public $ceiling = 0;

  public function __construct($type =
"B744", $ceiling = 37000) {
    $this->type = $type;
    $this->ceiling = $ceiling;
    echo "Constructed: ".$this->output();
  }
}

$boeing733 = new Aircraft("B733");
```

Declare class constants with `const`. Use `self::` to internally reference. You can only externally reference a class constant as a static property, using the scope resolution operator `::`.

```
class Aircraft {
  const airline = 'qantas';

  public function output() {
    echo self::airline."\n";
  }
}

$boeing747 = new Aircraft();
$boeing747->output();
echo Aircraft::airline;
```

Object Member Scope/Visibility

The scope/Visiblity, private, protected, or public, of an object's members, properties and methods, can be achieved with the following ...

```
// declaration
class myClass {
  // variables
  private $privateVar = "I am private";
  protected $protectedVar = "I am
protected";
  public $publicVar = "I am public";

  // functions
  private function privateFunction(){
    echo "I am private\n";
  }

  protected function protectedFunction (){
    echo "I am protected\n";
  }

  public function publicFunction (){
    echo "I am public\n";
  }
}

// usage
$myInstance = new myClass();

$myInstance->publicVar = "new value\n";
echo $myInstance->publicVar;     // ok
echo $myInstance->privateVar;    // error.
echo $myInstance->protectedVar;     //
error.

$myInstance->publicFunction();    // ok
$myInstance->privateFunction();   // error
$myInstance->protectedFunction();   // error
```

`Public` declared items can be accessed everywhere. `Protected` limits access to inherited and parent classes (the class that defines the item). `Private` limits visibility only to the class that defines the item.

PHP Manual > Visibility

Constants within objects cannot have a scope qualifier.

```
// Do this
class Aircraft {
    const airline = 'qantas';

// Not this
class Aircraft {
    public const airline = 'qantas';

// Call
echo Aircraft::airline;
```

Class members (properties and methods) are defined as public, protected, or private.

- Public declared items can be accessed everywhere.
- Protected limits access to inherited and parent classes (and to the class that defines the item).
- Private limits visibility only to the class that defines the item.

Properties must have an explicit scope declartion.

Methods without any scope declaration are defined as public.

PHP Manual > Visibility

Private functions can access public members without any issue.

```
class myClass {
  public $publicVar = "I am public";

  // functions
  private function privateFunction(){
    echo 'I am a private function.
$publicVar: '. $this->publicVar . "\n";
    $this->doMore();
  }

  public function publicFunction (){
    $this->privateFunction();
  }

  public function doMore() {
    echo "More done\n";
  }
}

$myInstance = new myClass();
$myInstance->publicFunction();
```

Static Members

Static properties and methods must have a `'static'` keyword declaration.

Use `self` to reference static properties from within static methods. (You can't use `'this'`).

```
// Static definition
class Aircraft {
  static public $maxLegalSpeed = 250;
  const airline = 'qantas';

  public function output() {
    echo "Max Legal Speed:
".self::$maxLegalSpeed."\n";
    echo self::airline . "\n\n";
  }
  static public function authority() {
    echo "CAA";
  }
}

// Ordinary calls
$boeing747 = new Aircraft("B744", 36000,
"boeing");
```

```
$boeing747->output();

// Static calls
Aircraft::authority();
echo Aircraft::$maxLegalSpeed."\n";
echo Aircraft::airline."\n";
```

Don't add members on the fly

Don't Define and Create an object then add members on the fly.

```
// OK
class Aircraft{};
$boeing747 = new Aircraft();
$boeing747->type = "B744";
echo $boeing747->type;

// Error
function outputx() {
  echo "nice\n";
}
$boeing747->output= 'outputx';
$boeing747->output();
```

While you can assign new properties you can't (as far as I know) assign functions to methods on the fly. Therefore the technique has no merit.

Callback Function

A PHP function is simply passed by its name as a string.

```
class coolClass {

  private $theCallBackFunction = "";

  public function
__construct($theCallBackFunction) {
    $this->theCallBackFunction =
$theCallBackFunction;
  }

  public function coolFunction () {
    call_user_func($this-
>theCallBackFunction, 8);
  }

  public function
fooFunction($theReturningCalledBackFunction)
{
    echo "Transformed suave number:
".call_user_func($theReturningCalledBackFunc
tion, 16);
  }
} // coolClass

function calledBack($secretNumber) {
  echo "The Secret Number is: " .
$secretNumber;
}

// Called back functions can return a value
// to the calling class!
function returningCalledBack($suaveNumber){
  return $suaveNumber + 5;
}

$coolObject = new coolClass('calledBack');
$coolObject->coolFunction();
$coolObject-
>fooFunction('returningCalledBack');
```

PHP Manual > Pseudo-types used in this documentation

To pass parameters by reference to a called back function, so the parameter's changes are accessible outside the called back function, use call_user_func_array, not call_user_func

```
<?php
// Unlike when functions are normally called
the '&' does not allow changes to be visible
outside the function. However, included it
for readability (You call the function with
a '&' in front of the variable)
function increment(&$var)
{
    $var++;
}

$a = 0;
call_user_func('increment', $a);
echo $a; // 0

 // Use '&' when you call the variable!
call_user_func_array('increment',
array(&$a)); // You can use this instead
echo $a; // 1
?>
```

PHP Manual > call_user_func

Dynamic Members (PHP 5)

"Overloading" in PHP doesn't mean the same thing in other object oriented languages (eg VB.NET). In other object oriented languages overloading means invoking a different method depending on the argument signature. In PHP "overloading" would have better been called "Dynamic members".

Thanks to: PHP Manual > Overloading > User Notes >Dot_Whut? (01-Oct-2004 02:52)

Dynamic Properties. Special methods `__get()` and `__set()`. Special argument order: `$name`; and `$value` (in `__set()`).

```
class Person {
  public function __get($name) {
      return $name;
  }

  public function __set($name, $value) {
    $this->$name = $value;
  }
}

$bill = new Person();
$bill->height = 130;
echo "$bill->height\n";

$bill->eyeColour = "Blue";
echo "$bill->eyeColour\n";
```

Dynamic Methods. Special method `__call()`. Special argument order: `$methodName, $argumentArray`.

```
class Caller {
   private $x = 15;

   private function __call($methodName,
$argumentArray) {
       print "Method [$methodName]
called:\n";
       var_dump($argumentArray);
       return $this->x;
   }
}

$foo = new Caller();
$a = $foo->bar(1, "2", 3.4, true);
echo $a;
```

`__unset()` and `__isset()` don't seem to work as advertised. Bug? You can just not use these methods but continue to use `isset()` and `unset()` as normal (don't define `__unset()` and `__isset()`).

```
class Person {
  public function __get($name) {
      return $name;
  }

  public function __set($name, $value) {
    $this->$name = $value;
  }

  public function __isset($name) {
    echo "Checking if [$name] is set .... ";
    if (isset($this->$name)) {
      echo "yes\n";
      return true;
    } else {
      echo "no\n";
      return false;
    }
  }

  public function __unset($name) {
    echo "Unsetting $name\n";
    unset($this->$name);
  }
}

$bill = new Person();
$bill->height = 130;
echo "$bill->height\n";

// Not working for PHP 5.1.2??
echo var_dump(isset($bill->height));
// True and doesn't fire __isset; Bug ?

unset($bill->height);
// Doesn't fire __unset; Bug?

echo var_dump(isset($bill->height));
// False and fires __isset
```

You can use dyanmic properties to serve as a property getter and/or setter. You can also perform validation.

```
class Person {
  private $age = 0;

  private function checkSupported($name) {
    $message = "[$name] not yet supported.";
    switch ($name) {
      case "age":
        break;
      default:
        throw new Exception($message);
        return false;
    }
    return true;
  }

  public function __get($name) {
    $this->checkSupported($name);
    return $name;
  }

  public function __set($name, $value) {
    $this->checkSupported($name);
    // Validation.
    if($value < 0 or $value > 100) {
      echo "[$name] must be between 0";
      echo " and 100 but was $value.\n";
    }
    $this->$name = $value;
  }
}

$bill = new Person();
$bill->age = 12;
// Ok.

$bill->age = 150;
// [age] must be between 0
// and 100 but was 150.

$bill->height = 130;
// Fatal error: Uncaught exception
// 'Exception' with message
// '[height] not yet supported.'
```

Object Comparison

(==)Two object instances are equal if they have the same attributes and values, and are instances of the same class.

(===), object variables are identical if and only if they refer to the same instance of the same class.

Namespaces

Intro

See https://atlas/Php/Examples/namespaces/namespaces-used-real.php

Namespaces can only effect: classes, interfaces, functions and constants (defined with "const", not "define").

https://www.php.net/manual/en/language.namespaces.definition.php

Define

Namespaces can be defined either:

- As a statement at the top of a file.Must be first php statement with no prior html constructs. Excepting for a php declare statement; or if a prior namespace statement has been made.

```
<?php
namespace Softmake;

// Now inside the Softmake namespace
const FUN_FACTOR = 10;

namespace Softspace;

function getIdea() {
  return 'fun';
}
?>
```

- Or with braces …

```
<?php
namespace Softmake {

  // Now inside the Softmake namespace
  const FUN_FACTOR = 10;
  ...

}
```

You can't mix namespace declaration styles in the one file. That is, you can't use statement style with braces style.

Anyway it is not a good idea to define multiple namespaces in one file.

You can define subnamesapces.

```
namespace MyProject\Sub\Level;
```

Namespace pattern. Use statement style (easier to convert old code); and a subnamespace that matches the file name.

```
// Define in string.php
<?php
namespace Softmake\String;

function getFieldWithSpaces($field) {
...

// Use in stringDemo.php
require_once
'../Libraries/StandardLibrary/string.php';
echo Softmake\String\getFieldWithSpaces
('AppealIDNowThatTheBbcHasXP').
```

C:\Users\John\Documents\Sda\Code\Php\Examples\stringDemo.php

Use

See https://atlas/Php/Examples/namespaces/namespaces-used-real.php

Use reference a namespace in one of three ways:

- With an unqualified name which will resolve to **currentnamespace\foo**

```
<?php
namespace Softmake;
// Previously defined namespace

echo FUN_FACTOR . "\n";
// Resoluves to \Softmake\FUN_FACTOR
```

- Relatively with a qualified name, without a leading backslash. Resolves to **currentnamespace\subnamespace\foo**.

```
<?php
// No 'namespace Softmake;'
// Relative reference
echo Softmake\FUN_FACTOR . "\n";
```

- Absolutely with a qualified name, prefixed with a leading backslash. Resolves to **\subnamespace\foo**.

```
<?php
// No 'namespace Softmake;'
// Relative reference
echo \Softmake\FUN_FACTOR . "\n";
```

Example class reference using namespace.

```
<?php
$myBook = new \Softmake\Book('Enchiridion',
'Epictetus', 10);

echo $myBook->author . "\n";
echo $myBook->getAuthorTitle() . "\n";
echo $myBook . "\n";
...
?>
```

Magic Methods

Summary

Magic methods entail (examined elsewhere):
__construct, __destruct
__call, __get, __set, __isset, __unset
__clone and __autoload

.. and (examined here):
__sleep, __wakeup,
__toString, __set_state

Serializing

`__sleep`, `__wakeup` are when serializing an object. Use `__sleep` to "close any database connections that the object may have, commit pending data or perform similar cleanup tasks."

```
class Person {
  private $name = "";
  public $age = 0;
  protected $updated = null;

  public function __construct(
                              $name,
                              $age,
                              $updated) {
    $this->name = $name;
    $this->age = $age;
    $this->updated = $updated;
  }

  public function __sleep() {
    echo "sleeping\n";
    // For private/protected variables
    // See PHP Manual > Magic Methods >
    // User Notes >
    // krisj1010 at gmail.com (09-Jan-2005
08:09)
    $sleepVars = array_keys((array)$this);
    return $sleepVars;
  }

  public function __wakeup() {
    echo "waking up\n";
  }

}

$mary = new Person("Mary",
                   28,
                   date(DATE_RFC2822));
echo $mary->age."\n";

$currentDirectory = dirname(__FILE__);
$outputFileName = $currentDirectory
    ."\phpTempSerialize.txt";
$dataToFile = serialize($mary);
file_put_contents($outputFileName,
                  $dataToFile);

$dataFromFile =
  file_get_contents($outputFileName);
$friend = unserialize($dataFromFile);
echo $friend->age."\n";
```

See PHP Manual > Magic Methods

__toString

The __toString method allows a class to decide how it will react when it is converted to a string.

```
class Person {
  private $name = "";
  public $age = 0;

  public function __construct(
                              $name,
                              $age) {
    $this->name = $name;
    $this->age = $age;
  }

  public function __toString() {
    return "$this->name is $this->age"."\n";
  }
}

$mary = new Person("Mary", 28);
echo $mary;
```

`__toString` is invoked in a limited number of cases.

```
// __toString called
echo $classVar;

// __toString called
echo 'text', $classVar;

// __toString not called
echo 'text' . $classVar;

// __toString not called
```

```
echo (string) $classVar;

// __toString not called
echo "text $classVar";
```

PHP Manual > *Magic Methods*

__set_state

`__set_state` is a static method is called for classes exported by `var_export()` since PHP 5.1.0.

PHP Manual > *Magic Methods*

__clone

To clone an object, creating a shallow copy (where existing property references remain as references), use the clone language construct.

```
$obj2 = clone $obj;
```

If the class has a __clone defined this will be called to allow the creation of clean copies of properties (not just a new reference).

```
class SubObject {
  static $instances = 0;
  public $instance;

  public function __construct() {
     $this->instance = ++self::$instances;
  }

  public function __clone() {
     $this->instance = ++self::$instances;
  }
}

class MyCloneable {
  public $object1;
  public $object2;

  function __clone() {
     // Force a copy of this->object,
otherwise
     // it will point to same object.
     $this->object1 = clone $this->object1;
  }
}

$obj = new MyCloneable();

$obj->object1 = new SubObject();
$obj->object2 = new SubObject();

$obj2 = clone $obj;
```

PHP Manual > *Object cloning*

Autoloading Classes

The `__autoload` function is a convenience that allows you to load class definitions that live in their own file.

```
function __autoload($class_name) {
   require_once $class_name . '.php';
}

$obj  = new MyClass1();
$obj2 = new MyClass2();
```

Loads the classes MyClass1 and MyClass2 from the files MyClass1.php and MyClass2.php respectively.

PHP Manual > *Autoloading Objects*

Object Inheritance (PHP 5)

Basics

A class can inherit methods and members of another class by using the `extends` keyword.

In the inherited class override parent methods by using the same name. Refer to the parent method with `parent::`

```
class Aircraft {
  public $engines = 0;
  public $ceiling = 0;

  public function __construct($engines = 1,
$ceiling = 40000) {
     $this->engines = $engines;
     $this->ceiling = $ceiling;
  }

  public function output() {
    echo "$this->engines $this->ceiling ";
  }
}

class PropPlane extends Aircraft {
  public $propDiameter = 0;

  public function __construct($engines,
$ceiling, $propDiameter) {
     // Call parent method
     parent::__construct($engines, $ceiling);
     $this->propDiameter = $propDiameter;
  }

  // Override the parent method
  function output() {
     parent::output();
     echo "$this->propDiameter \n";
     // Call property
     echo "These $this->engines engine(s)
will have to do.";
  }
}

$cessna172 = new PropPlane(1, 10000, 1.2);
$cessna172->output();

// 1 10000 1.2
// These 1 engine(s) will have to do.
```

Final

The "final" keyword can be used with a class or a method.

A final method cannot be overriden in a child class.

```
class BaseClass {
  final public function moreTesting() {
       echo "BaseClass::moreTesting()
called\n";
  }
}
class ChildClass extends BaseClass {
  public function moreTesting() {
       echo "ChildClass::moreTesting()
called\n";
  }
}
// Results in Fatal error.
```

A final class cannot be extened.

```
final class BaseClass {
  public function test() {
       echo "BaseClass::test() called\n";
  }
}

class ChildClass extends BaseClass {
}
// Results in Fatal error.
```

Abstract Class

Classes and methods can be marked as `abstract`.

Abstract classes: can't have instances created; and must be abstract if any method is abstract.

Abstract methods must: declare signature and do not define an implementation; be defined by any child class; have the same or weaker visibility in the child class.

```
abstract class Aircraft {
   // Method must be inherited and defined
   abstract protected function
getEngineType();
   // Method common to all inherited classes
   public function output() {
       print $this->getEngineType() . "\n";
   }
}

class Cessna extends Aircraft {
   protected function getEngineType() {
       return "Prop\n";
   }
}

$cessna172 = new Cessna;
echo $cessna172->output();
```

For example, if the abstract method is defined as protected, the function implementation must be defined as either protected or public.

PHP Manual > *Class Abstraction*

Interfaces

Interfaces declare methods without implementing them. A class "implements" an interface and must define all the methods of the interface. All methods in the interface must be public.

```
interface iAircraft {
   public function getEngineType();
}

class Cessna implements iAircraft {
  public function getEngineType(){
    return "Prop\n";
  }
}

$cessna172 = new Cessna;
echo $cessna172->getEngineType();
```

A class can implement more than one interface.

Namespaces (Simulated)

Prefix your class, function, variable, and constant names.

```
define('SoftmakeCoolFactor', 12);
$SoftmakeClouds = 5;
function SoftmakeGetIdea () {
  ...
}
class SoftmakeBook {
  ...
  public function priceDisplay () {
    return "\$$this->price";
  }
}
```

Namespaces 5.3.0

Declare/Define

Declare namespaces on first line. Only a php declare statement and the PHP escape characters may proceed.

```
// Conventional syntax
<?php
namespace Softmake\Forms;

define('CoolFactor', 12);
function getIdea () {/* ... */}
class Book {/* ... */  }
}
?>

// Alternative syntax
<?php
namespace Softmake {
  define('CoolFactor', 12);
  function getIdea () {/* ... */}
  class Book {/* ... */  }
}
?>
```

You can't nest a namespace (without a simulation) but you can declare subnamespaces

```
<?php
namespace Softmake\Forms;
```

It is bad form to declare/define multiple namespaces in one file.

Namespaces effect only classes, function and constants. Namespaces do not effect variables.

When a custom namespace is declared call the global namespace with a leading backslash '\'.

```
namespace Softmake\DateAndTime;

...

  public function __toString() {
    try {
       return (string) $output = $this-
>dateTimeUnix-
>format(\DateTimeInterface::ISO8601);
    } catch (Exception $exn) {
      return '';
    }
  }
```

Call

Call a variable, function, etc within a namespace with backslashes.

```
Softmake\ImageHandling\IptcKeyEnum::Headline
Softmake\String\getCamelCaseToTitleCase($inp
ut);
```

Reflection

Before using reflection consider the magic constants:

```
LINE__, __FILE__, __FUNCTION__, __CLASS__,
__METHOD__, __COMPILER_HALT_OFFSET__
```

Basic usage

```
Reflection::export(new
ReflectionClass('Aircraft'));
```

To reverse enginer PHP language aspects use the reflection clases:

```
ReflectionFunction
ReflectionParameter
ReflectionMethod
ReflectionClass
ReflectionObject
ReflectionProperty
ReflectionExtension
```

Reflection example

```
// Create an instance of the ReflectionClass
class
$class = new ReflectionClass('Aircraft');

// Print out basic information
printf(
   "===> The %s%s%s %s '%s' [extends %s]\n"
.
   "     declared in %s\n" .
   "     lines %d to %d\n" .
   "     having the modifiers %d [%s]\n",
       $class->isInternal() ? 'internal' :
'user-defined',
       $class->isAbstract() ? ' abstract' :
'',
       $class->isFinal() ? ' final' : '',
       $class->isInterface() ? 'interface' :
'class',
       $class->getName(),
       var_export($class->getParentClass(),
1),
       $class->getFileName(),
       $class->getStartLine(),
       $class->getEndline(),
       $class->getModifiers(),
       implode(' ',
Reflection::getModifierNames($class-
>getModifiers()))
);

// Print documentation comment
printf("---> Documentation:\n %s\n", $class-
>getDocComment());

// Print which interfaces are implemented by
this class
printf("---> Implements:\n %s\n",
var_export($class->getInterfaces(), true));

// Print class constants
printf("---> Constants: %s\n",
var_export($class->getConstants(), true));

// Print class properties
printf("---> Properties: %s\n",
var_export($class->getProperties(), true));

// Print class methods
$reflectionMethods = $class->getMethods();
$methodOutput = "";
foreach ($reflectionMethods as
$reflectionMethod) {
  $methodOutput .= "    ".$reflectionMethod-
>getName()."\n";
}
printf("---> Methods: \n%s\n",
$methodOutput);
```

You can extend the reflection clasess for convenience.

PHP Manual > *Reflection*

Runtime Analysis

Debugging

Install the debugger xdebug.

This generates a stack trace and local variable output upon errors.

For information on adding a debugger see "Server Configuation" above.

Install configure tsWebEditor.

```
View > Settings > General > Browser =
"C:\Program Files\Mozilla
Firefox\firefox.exe" [include quotes]

View > Settings > Tools [Add]
Firefox; program = browser_u; path =
C:\data\sda\code; url =
http://localhost:8080

Debug > Parmaters > Run = Remote; url =
http://localhost:8080
```

This works with xdebug to enable stepping through code and breakpoints.

http://www.tswebeditor.tk/

Use tsWebEditor

```
* Open tsWebEditor and load your script
file.
* Add a breakpoint to your desired line.
* Click on Preview Button. (eg Firefox)
* In your Browser Add to your url
?XDEBUG_SESSION_START=tswebeditor"
* In Browser Reload.
```

Basic Inspection functions.

```
$products = array("nice","mighty",1 ,2);
echo $products;
var_dump($products);
print_r($products);
debug_print_backtrace();
```

C:\Data\Sda\Code\Php\Examples\debugDemo.php

Stack Tracing

Use xdebug for a strack trace. Insert this in your code:

```
<?php print
'<pre>'.xdebug_print_function_stack( "jlb"
).'</pre>'; ?>
```

Ensure that xdebug is installed and enabled in PHP.ini.

Profiling

Install the debugger xdebug (version 2).

For information on adding a debugger see "Server Configuation" above.

Output cumulated script execution time in milliseconds with the function `xdebug_time_index()`

```
function tree() {
  echo $eol."Now: ".
round(xdebug_time_index(), 3)*1000
."ms".$eol;

// do stuff

  echo $eol."Now: ".
round(xdebug_time_index(), 3)*1000
."ms".$eol;
}
```

Output cumulated script execution time in milliseconds based on native functions (PHP 5).

```
/*
Returns: the time, in milliseconds, since
```

```
the first call of the the function.
Usage:
  // Call the function to initialise.
  benchmarkScript();
  sleep(1);
  // Subsequent calls return time taken
  since first placement of the function.
  echo "Since first: ".benchmarkScript();
**/
function benchmarkScript( $round = 3 ) {
  static $startTime;
  if ( empty( $startTime ) ) {
    $startTime = microtime(true);
  } else {
    return round( microtime(true) -
$startTime, $round) * 1000 . " ms";
  }
}
```

See > Bentley > Php\Examples\benchmarkScript.php

Log an exhaustive execution profile to find bottlenecks in your code with Xdebug (version 2) profiling.

```
* Add this to your php.ini
xdebug.profiler_output_dir =
"C:/Data/ZTemp/PhpXdebugProfiles"
xdebug.profiler_output_name = "timestamp"
xdebug.profiler_enable = 1

[In addition to:
zend_extension_ts="C:/Php/ext/php_xdebug-
2.0.0rc3-5.2.1.dll"
xdebug.remote_enable=On]

* Make sure the directory specified in
xdebug.profiler_output_dir exists.

* Restart Apache.

* Run your script and find profile output in
C:/Data/ZTemp/PhpXdebugProfiles. Named "
cachegrind.out.*"

* Install WinCacheGrind
http://sourceforge.net/projects/wincachegrind

* With WinCacheGrind open your profile
output file "cachegrind.out.*" and analyse.
```

http://www.xdebug.org/docs-profiling2.php

Error And Exception Handling Concepts

Php.ini config

Important php.ini directives for error and Exception handling ...

```
; Show all errors
; and coding standards warnings
error_reporting  =  E_ALL | E_STRICT

; During development only.
display_errors = On

; Eg to Apache text file.
log_errors = On

; Log errors to specified file.
;error_log = filename

; Log errors to syslog (Event Log on
Win2000)
;error_log = syslog

; Store the last error/warning message in
$php_errormsg (boolean).
track_errors = Off

; String to output before an error message.
error_prepend_string = "<div
style='color:#ff0000'>"

; String to output after an error message.
error_append_string = "</div>"
```

Error Handling

You should use both (classic) error handling and exception handling.

Classic Error types in decreasing severity: Strict; Notice; Warning; Error (Fatal/Stops Execution).

Error Operator, @, When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.

```
/* Intentional file error */
$my_file = @file ('non_existent_file') or
   die ("Failed opening file: error was
'$php_errormsg'");
```

If the track_errors feature is enabled, any error message generated by the expression will be saved in the variable $php_errormsg.

Exceptions

Exceptions in PHP exists only to allow you to throw them.

Exceptions are useful for bubbling up error conditions to calling code.

The hassle of manually bubbling up errors through multiple callers is one of the prime reasons for the implementation of exceptions in programming languages

PHP Error Handling > George Schlossnagle

Each try must have a catch block. Code in try block halts at first exception. Execution continues after catching exceptions.

```
try {
  $message = 'Always throw this error';
  throw new Exception($message);
  echo 'Code following an exception is not
executed.';
} catch (Exception $e) {
  echo 'Caught exception: ',  $e-
>getMessage(), "\n";
}
echo 'Execution continues after exception
caught.';
```

Fatal errors do not generate exceptions. Execution just halts (unless there is an error handler set or a try … catch clause.)

```
try {
  functionDoesNotExist();
  echo 'Fatal errors do not generate
exceptions';
} catch (Exception $e) {
  echo 'Caught exception: ',  $e-
>getMessage(), "\n";
}
echo 'Execution does not continue after
classic fatal error.';

// Output
Fatal error:  Call to undefined function
functionDoesNotExist() in
C:\Data\Sda\Code\Php\Examples\play.php on
line 51
```

Warnings do not generate exceptions. Execution continues (unless there is an error handler set).

```
try {
  $x = 12/0;
  echo "Classic warnings do not generate
exceptions\n";
} catch (Exception $e) {
  echo 'Caught exception: ',  $e-
>getMessage(), "\n";
}
echo 'Execution does continue after classic
warning.';

// Output
Warning:  Division by zero in
C:\Data\Sda\Code\Php\Examples\play.php on
line 51
Classic warnings do not generate exceptions
Execution does continue after classic
warning
```

Throwing an Exception results in a fatal error by default unless you handle it with a try…catch clause (which you can use to pass off it a central exception handler)

```
$message = 'Always throw this error';
throw new Exception($message);

// Fatal Error: Uncaught exception
'Exception' ...
```

Exceptional Code > By Matt Zandstra > 29 July 2004 > http://www.zend.com/php5/articles/php5-exceptions.php

Rethrow an exception if you want.

```
try {
  throw new Exception;
} catch (Exception $e) {
  print "Exception caught, and rethrown\n";
  throw $e;
}
```

PHP Error Handling > George Schlossnagle

Exception class

The native exception class:

```
class Exception {
   protected $message = 'Unknown exception';
   protected $code = 0;
   protected $file;
   protected $line;

   function __construct($message = null,
                        $code = 0);

   final function getMessage();
   final function getCode();
   final function getFile();
   final function getLine();

   // an array of the backtrace()
   final function getTrace();
   final function getTraceAsString();

   /* Overrideable */
   function __toString();
}
```

User Defined Exceptions

A user defined Exception class can be defined by extending the built-in Exception class.

```
class MyException extends Exception {
   // Redefine the exception so message
   // isn't optional
   public function __construct($message,
                               $code = 0) {
       // some code
```

```
        // You should always
        // call the parent constructor.
        parent::__construct($message, $code);
    }

    // Redefine for your convenience.
    public function __toString() {
        return __CLASS__ . ": "
        ."[{$this->code}]: "
        ."{$this->message}\n";
    }

    public function customFunction() {
        echo "A Custom function"
             ." for MyException\n";
    }
}
```

This allows you to catch different Exception types. You should always organize your catch clauses from the most specific to the most general.

```
try {
  throw new MyException("Strange happening",
                        600);
} catch (MyException $e) {
  echo "Caught my exception\n", $e;
  $e->customFunction();
} catch (Exception $e) {
  echo "<pre>Caught default exception\n $e
</pre>";
}
// Continue execution
echo 'Hello World ';

// Output
Caught my exception
MyException: [600]: Strange happening
A Custom function for MyException
Hello World
```

Exceptional Code > *By Matt Zandstra*> *29 July 2004* > *http://www.zend.com/php5/articles/php5-exceptions.php*

Central Handlers

The following error types cannot be handled with a user defined function: E_ERROR, E_PARSE, E_CORE_ERROR, E_CORE_WARNING, E_COMPILE_ERROR, E_COMPILE_WARNING, and most of E_STRICT raised in the file where set_error_handler() is called.

PHP Manual > *set_error_handler*

It is not possible to handle fatal errors with your own handler.

Even if you have set your own handler, fatal error will be always handled by PHP's default handler.

PHP Manual > *set_error_handler*> *User Notes* > *eregon at eregon dot info* > *02-Aug-2005 05:35*

It's not a good idea to mail from your central error handler. Rather, apply a script to an error log that mails you intelligently, rather than on every error.

"Errors have a way of clumping up together. It would be great if you could guarantee that the error would only be triggered at most once per hour (or any specified time period), but what happens more often is that when an unexpected error occurs due to a coding bug, many requests are affected by it. This means that your nifty mailing error_handler() function might send 20,000 mails to your account before you are able to get in and turn it off. Not a good thing.

If you need this sort of reactive functionality in your error-handling system, I recommend writing a script that parses your error logs and applies intelligent limiting to the number of mails it sends."

PHP Error Handling > *George Schlossnagle* > *Apr 23, 2004* > *http://www.informit.com/articles/article.asp?p=170279&rl=1*

Don't turn off the display of fatal errors.

```
// set the error reporting level directive
at runtime.
error_reporting(E_ALL | E_STRICT);

// Always display fatal error, even in
production.
ini_set("display_errors", "On");

// Only use the central error handler for
Non strict errors
set_error_handler("centralErrorHandler");
// Using code like the above allows the
// display of fatal errors. Fatal errors
can't
// be handled by a central error handler.
They
// can only be displayed or not. Better to
// have an awareness that they occur. Even
in
// a production environment.
```

In production display the fact that an error occurred but don't show details. Log details.

George says: Displaying errors to users is not only ugly but a security vunerability.

But even in production it is good to have some indictation that an error occurred! That aids getting rid of them. Better an error message that annoys people than data is slowly corrupted or bizarre results returned.

We can make the in production error message user friendly though!

PHP Error Handling > *George Schlossnagle* > *Apr 23, 2004* > *http://www.informit.com/articles/article.asp?p=170279&rl=1*

Error and Exception Handling Implementation

Common

Only use (classic) error handling unless you need to throw an exception.

In PHP you can't catch (classic) errors with an exception handler (a try… catch) clause. However, classic errors are readily traceable. See "Debugging" above.

Use a errorAndExceptionCentralHandler.php file for global error options set with ini_set(), don't use PHP.ini directives.

You may not control the production php.ini, for example, because you are using a third party web host.

During Production

Use John Bentley's errorAndExceptionCentralHandler.php to log errors and email the log at most once a day.

```
// Follow the usage instructions in
errorAndExceptionCentralHandler.php.
```

During Development

Choose either:

* The default native error reporting.

```
// In a global.php
ini_set("display_errors", "On");
// or php.ini
display_errors = On
```

* Error reporting level:

```
error_reporting = E_ALL | E_STRICT
```

* The default native error reporting with Xdebug installed.

This can give a stack trace and variable dump upon error. See "Server Configuation" and "debugging" above.

* errorAndExceptionCentralHandler.php with

```
ErrorLogMode = $native.
```

This will pass the error through to the navite handler. So either of the above 2 options will take effect.

* errorAndExceptionCentralHandler.php with

```
ErrorLogMode = $developmentLog.
```

This will display and log errors.

Exception Handling Implementaion

Choose either:

* Throw an exception and report with native central exception handling.

```
throw new Exception("Strange happening",
600);

// Causes fatal exception, handled by native
php settings (so if xdebug installed then
xdebug info will be provided).
```

Useful for assertions, pre and post conditions.

* Throw an exception and report with custom central exception handler.

```
// John Bentley's
errorAndExceptionCentralHandler.php
with

ErrorLogMode = $developmentLog.
//Or
ErrorLogMode = $productionLog.

// In your source:
require_once
'errorAndExceptionCentralHandler.php';
```

This will log (and display for development mode) exceptions as errors.

* Throw an exception and hande (sometimes mere reporting) within try … catch clause.

```
// Just use "throw err" to pass up to native
// or custom central handler…
```

```
  try {
    ...
  } catch (Exception $e) {
    throw $e;
  }
//... Or display there and then…
  try {
    ...
  } catch (Exception $e) {
    echo "<pre>Caught default exception\n $e
</pre>";
  }
```

Throw an exception and hande within try … catch clause. Simple design pattern.

```
function getLetterSimple($letterIndex) {
  $letters = array('a','b','c','d','e');
  try {
    if ($letterIndex < 0 || 4 < $letterIndex) {
      throw new Exception("\$letterIndex must be
        between 0 and 4", 100);
    }
    throw new Exception("My cool exception"
      , 200);
  } catch (Exception $exn)  {
    switch ($exn->getCode()) {
      case 100:
        echo $exn->getMessage()
          . " but was $letterIndex! <br />";
        break;
      default:
        // This ensures the unanticpated
        // exception is reported to the
        // Central Error Handler
        throw $exn;
    }
    return false;
  }
  // Executes even after exception caught
  //(so long as no prior 'return' executed).
  return "letter was: ".$letters[$letterIndex]
    ."<br />";
}
```

C:\Data\Sda\Code\Php\Libraries\StandardLibrary\errorAndExceptionHandling_TryCatchlHandlingDemo.php

Throw an exception and hande within try … catch clause. Full design pattern.

```
function getLetterFull($letterIndex) {
  $letters = array('a','b','c','d','e');

  try {

    if (!is_int($letterIndex)) {
      throw new MyException(
        "Entry was not an integer ", 100);
    }
    if ($letterIndex < 0 || 4 < $letterIndex) {
      throw new Exception("\$letterIndex must"
        ." be between 0 and 4", 200);
    }
  // Order more specific exception (those based
  // on other exceptions classes) to least
  // specific.
  } catch (MyException $exn)  {
    switch ($exn->getCode()) {
      case 100:
        // Use user defined toString in
        // MyException Class.
        echo "$exn but was $letterIndex!<br />";
        break;
      default:
        // This ensures the unanticpated
        // exception is reported to the
        // Central Error Handler
        throw $exn;
    }
    return false;

  } catch (Exception $exn) {

    switch ($exn->getCode()) {
      case 200:
        echo $exn->getMessage()
          . " but was $letterIndex! <br />";
        break;
      default:
        // This ensures the unanticpated
        // exception is reported to the
        // Central Error Handler
        throw $exn;
    }
    return false;

  } // try

  // Executes even after exception caught
  //(so long as no prior 'return' executed).
  return "letter was: ".$letters[$letterIndex]
    ."<br />";
}
```

C:\Data\Sda\Code\Php\Libraries\StandardLibrary\errorAndExceptionHandling_TryCatchlHandlingDemo.php

See also " User Defined Exceptions" Above

Regular Expressions (PCRE/Perl Compatible)

Reach first for PCRE/Perl Compatible rather than Posix Extended regular expressions.

Pattern Modifiers

Flag	Meaning
i	Peform a case-insensitive match.
m	A multiline input string should be treated as multiple lines. If the m flag is used, ^ and $ match at the start or end of any line within the input string instead of the start or end of the entire string.
s	A dot metacharacter in the pattern matches all characters, including newlines. Without it, newlines are excluded.
x	whitespace data characters in the pattern are totally ignored except when escaped or inside a character class, and characters between an unescaped # outside a character class and the next newline character, inclusive, are also ignored
e	If this modifier is set, preg_replace() does normal substitution of backreferences in the replacement string, evaluates it as PHP code, and uses the result for replacing the search string.
u	Pattern strings are treated as UTF-8.
A	the pattern is forced to be "anchored", that is, it is constrained to match only at the start of the string which is being searched
D	a dollar metacharacter in the pattern matches only at the end of the subject string. Without this modifier, a dollar also matches immediately before the final character if it is a newline (but not before any other newlines)
S	When a pattern is going to be used several times, it is worth spending more time analyzing it in order to speed up the time taken for matching. If this modifier is set, then this extra analysis is performed.
U	This modifier inverts the "greediness" of the quantifiers so that they are not greedy by default, but become greedy if followed by "?".. It can also be set by a (?U) modifier setting within the pattern or by a question mark behind a quantifier (e.g. .*?).
X	Any backslash in a pattern that is followed by a letter that has no special meaning causes an error, thus reserving these combinations for future expansion.

PHP Manual > Regular Expression Functions (Perl-Compatible) > Pattern Modifiers

Metacharacters

To match a metacharacter literally escape with a backslash "\"

```
/\./ matches the period rather than any
character.
```

Non Printing Characters

Character	Matches	Example
\e	escape	
\f	formfeed	
\n	newline	
\r	Carriage return	
\t	tab	
\xhh	Hex code	\x20 is space
\ddd	Octal Code (or Backreference)	

Character Type Metacharacters

Character	Matches	Example
\d	Numeral 0 through 9	/\d\d\d/ matches “212” and “415” but not “B17”
\D	Non-numeral	/\D\D\D/ matches “ABC” but not “212” or “B17”
\s	Single white space	/over\sbite/ matches “over bite” but not “overbite” or “over bite”
\S	Single non-white space	/over\Sbite/ matches “over-bite” but not “overbite” or “over bite”
\w	Letter, numeral, or underscore	/A\w/ matches “A1” and “AA” but not “A+”
\W	Not letter, numeral, or underscore	/A\W/ matches “A+” but not “A1” and “AA”
.	Any character except newline	/.../ matches “ABC”, “1+3”, “A 3”, or any three characters
[...]	Character set	/[AN]BC/ matches “ABC” and “NBC” but not “BBC”
[x-x]	Character set Range	[0-9] matches any digit.
[^...]	Negated character set	/[^AN]BC/ matches “BBC” and “CBC” but not “ABC” or “NBC”
\p{xx}	when UTF-8 mode: a character with the xx property	\p{Lu} always matches only upper case letters
\P{xx}	when UTF-8 mode:	

	a character without the xx property	
\x{xxxx}	an extended Unicode Sequence	/\x{0423}/u matches 'У' (http://en.wikipedia.org/wiki/U_%28Cyrillic%29). Use your regex in unicode mode. The 'u' after /

POSIX notation character classes. Names enclosed by [: and :].

alnum	letters and digits
alpha	letters
ascii	character codes 0 - 127
blank	space or tab only
cntrl	control characters
digit	decimal digits (same as \d)
graph	printing characters, excluding space
lower	lower case letters
print	printing characters, including space
punct	printing characters, excluding letters and digits
space	white space (not quite the same as \s)
upper	upper case letters
word	"word" characters (same as \w)
xdigit	hexadecimal digits

https://www.php.net/manual/en/regexp.reference.character-classes.php

Boundary/Positional Metacharacters

Character	Matches	Example
\b	Word boundary	/\bor/ matches "origami" and "or" but not "normal" /or\b/ matches "traitor" and "or" but not "perform" /\bor\b/ matches full word "or" and nothing else
\B	Word non-boundary	/\Bor/ matches "normal" but not "origami" /or\B/ matches "normal" and "origami" but not "traitor" /\Bor\B/ matches "normal" but not "origami" or "traitor"
^	At beginning of a string	/^Fred/ matches "Fred is OK" but not "I'm with Fred" or line or "Is Fred here?"
$	At end of a string or line	/Fred$/ matches "I'm with Fred" but not "Fred is OK" or "Is Fred here?"
\A	start of subject (independent of multiline mode)	
\Z	end of subject or newline at end (independent of multiline mode)	
\z	end of subject (independent of multiline mode)	
\G	first matching position in subject	
x(?<=y)	Matches 'x' only if 'x' is followed by 'y'.	For example, /Jack(?=Sprat)/ matches 'Jack' only if it is followed by 'Sprat'. /Jack(?=Sprat|Frost)/matches 'Jack' only if it is followed by 'Sprat' or 'Frost'. However, neither 'Sprat' nor 'Frost' is part of the match results.
x(?<!y)	Matches 'x' only if 'x' is not followed by 'y'.	For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. The regular expression /\d+(?!\.)/.exec("3.141") matches 141 but not 3.141.

Counting Metacharacters

Character	Matches Last Character	Example
*	Zero or more times	/Ja*vaScript/ matches "JvaScript", "JavaScript", and "JaaavaScript" but not "JovaScript"
?	Zero or one time	/Ja?vaScript/ matches "JvaScript" or "JavaScript" but not "JaaavaScript"
+	One or more times	/Ja+vaScript/ matches "JavaScript" or "JaavaScript" but not "JvaScript"
{n}	Exactly n times	/Ja{2}vaScript/ matches "JaavaScript" but not "JvaScript" or "JavaScript"
{n,}	n or more times	/Ja{2,}vaScript/ matches "JaavaScript" or "JaaavaScript" but not "JavaScript"
{n,m}	At least n, at most m time	s /Ja{2,3}vaScript/ matches "JaavaScript" or "JaaavaScript" but not "JavaScript"

Operators

Character	Meaning	Example
x|y	Matches either 'x' or 'y'.	/green|red/ matches 'green' in "green apple" and 'red' in "red apple."
(x)	Matches 'x' and remembers the match. These are called capturing parentheses.	/(foo)/ matches and remembers 'foo' in "foo bar." The matched substring can be recalled from the resulting array's elements [1], ..., [n].
(?P<name>pattern)	Array with matches will contain the match indexed by the string alongside the match indexed by a number	
x(*?options*)	Change the options for remainder of pattern. It is also possible to unset these options by preceding the letter with a hyphen, and a combined setting and unsetting such as (?im-sx)	/ab(?i)c/ matches only "abc" and "abC".
(?:x)	Matches 'x' but does not remember the match. These are called non-capturing parentheses. The matched substring can not be recalled from the resulting array's elements [1], ..., [n].	
(?#x)	A comment.	
(?R)	Recursion.	

Conditional subpatterns

```
(?(condition)yes-pattern)
(?(condition)yes-pattern|no-pattern)

3 alternatives for Condition...

Condition is digit: the condition is
satisfied if the capturing subpattern of that
number has previously matched.

Condition is string 'R': satisfied if a
recursive call to the pattern or subpattern
has been made. At "top level", the condition
is false.

Condition is an assertion.
```

Basics

Surround patterns with forward slash "/". Pattern modifiers after closing slash.

```
$pattern = '/hello/m';
```

Regular expression matching is case-sensitive by default.

Basic Matching (string functions)

Index test with string function `strpos` or `stripos` (not regex). Must use type and value comparison operator.

```
// This function may return Boolean FALSE,
// but may also return a non-Boolean value
// which evaluates to FALSE, such as 0 or ""
// We have to use type and value matching.
$pos = stripos($haystack, $needle);
if ($pos !== false) {
  echo "match at $pos<br />";
} else {
  echo "no match<br />";
}

// Or strpos for case-sensitive match.
```

Matching into an array

Use `preg_match` or `preg_match_all`.

The parentheses around the segments of the regular expression form capture groups which are feed into the $matches array. Starting index 1. Index 0 refers to entire match.

```
\

// Output
0: FUCK
```

```
1: FU
2: CK
```

Replace

Basic replace.

```
// Removes all commas.
function removeCommas($stringToSearch) {
  $pattern = '/,/';
  $replace = '';
  return preg_replace($pattern,
                      $replace,
                      $stringToSearch);
}

echo removeCommas("999,999,999")."\n";
// Result: 999999999
```

Replace with backreferences to capture patterns.

```
function insertCommas($stringToSearch) {
  $pattern = '/(-?\d+)(\d{3})/';
  $replace = '$1,$2';
  while (preg_match($pattern,
                    $stringToSearch,
                    $matches)) {
    $stringToSearch = preg_replace($pattern,
                        $replace,
                        $stringToSearch);
  }
  return $stringToSearch;
}
echo insertCommas("1000000")."\n";
// Result: 1,000,000
```

$1 ... $99 are backreferencees to the capture patterns, defined by the parentheses "( )" in the regular expression pattern. $0 refers to the text matched by the whole pattern.

Running Php as a Script

Create a script.

```
#!/usr/bin/php
<?php
echo 'Hello World PHP'."\n";
?>
```

Php Manual, Executing PHP files, *https://www.php.net/manual/en/features.commandline.usage.php*

Run a script from the command line

```
> php scriptHelloWorld.php;
Hello World PHP
```

Php Manual, Executing PHP files, *https://www.php.net/manual/en/features.commandline.usage.php*

Reference Example

Sources

⇒ PHP Manual
http://www.php.net/manual/en/index.php

⇒ PHP Error Handling > George Schlossnagle > Apr 23, 2004 >
http://www.informit.com/articles/article.asp?p=170279&rl=1

⇒ PHP5 Exception handling > Andrea Shalter >
http://www.andreashalter.ch/phpug/20040115/

Document Licence