PHP Mastery

By: Andrew Gerst

http://about.me/agerst (http://about.me/agerst)
PHP Mastery (Source) (http://hnswave.co/mastery/php.html)

Preface

```
This is a book about the PHP programming language. It is intended for programmers transitioning from another
language such as JavaScript as well as programmers who have been working with PHP at a novice level and are now
ready for a more sophisticated relationship with the language.
```

Table of Contents

What is PHP?

```
PHP is an HTML-embedded server-side scripting language. Much of its syntax is borrowed from C, Java and Perl with
a couple of unique PHP-specific features thrown in. The goal of the language is to allow web developers to write
dynamically generated pages quickly.
```

Introduction & History

PHP: Hypertext Preprocessor (almost always abbreviated PHP) is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.

Very Brief History

PHP/FI 2.0 is an early and no longer supported version of PHP. PHP 3 is the successor to PHP/FI 2.0 and is a lot nicer. PHP 5 is the current generation of PHP, which uses the » Zend Engine 2 which, among other things, offers many additional OOP features.

Differences between PHP 4 and PHP 5

While PHP 5 was purposely designed to be as compatible as possible with previous versions, there are some significant changes. Some of these changes include:

- A new OOP model based on the Zend Engine 2.0
- A new extension for improved MySQL support
- Built-in native support for SQLite
- A new error reporting constant, E_STRICT, for run-time code suggestions
- A host of new functions to simplify code authoring (and reduce the need to write your own functions for many common procedures)

Hello World

There is built-in I/O functionality in PHP. Here are a couple methods of producing output for the browser.

```php
/**
* Hello World
*/

echo "Hello World!";
print "Hello World!";
die('Hello World!');
exit('Hello World!');
dump_var($var);
print_r($var);
```

Features

TODO

Trying Programs

Now that you've been introduced to the language I'd like to provide a couple methods so you can try PHP yourself.

http://writecodeonline.com/php/ (http://writecodeonline.com/php/)
http://www.compileonline.com/execute_php_online.php (http://www.compileonline.com/execute_php_online.php) // vim and emacs editor
http://ideone.com/ (http://ideone.com/)
http://codepad.org/ (http://codepad.org/)
https://codeanywhere.net/ (https://codeanywhere.net/)
https://compilr.com/ (https://compilr.com/)

Beautify PHP Code

http://beta.phpformatter.com/ (http://beta.phpformatter.com/)

Another approach is to simply create a PHP file containing the program and load it in your browser. For example, you could create a file called *test.php* with the following content:

```
<?php
echo "Hello World";
?>
```

PHP Hosting Environments

XAMPP (http://www.apachefriends.org/en/xampp.html)
Heroku (https://www.heroku.com/)
000webhost (http://www.000webhost.com/)

Testing Script Performance

Sometimes after all day long coding your code becomes not so effective and your code (usually interface related) becomes slow. You have done so many changes and don't exactly know what is slowing it down. In cases like this (and of course, plenty other cases) you can test your PHP code performance.

A widely used threshold of time is 100 ms for when people start to notice latency in applications.

If subtracting microtimes gives you negative results, try using the function with the argument true (microtime(true)). With true, the function returns a float instead of a string (as it does if it is called without arguments).

http://www.phpbench.com/ (http://www.phpbench.com/)

PHPBench.com was constructed as a way to open people's eyes to the fact that not every PHP code snippet will run at the same speed.

PHP Performance Tips

PHP is a very popular scripting language, used on many popular sites across the web. In this article, we hope to help you to improve the performance of your PHP scripts with some changes that you can make very quickly and painlessly. Please keep in mind that your own performance gains may vary greatly, depending on which version of PHP you are running, your web server environment, and the complexity of your code.

Profile your code to pinpoint bottlenecks

Hoare's dictum states that Premature optimization is the root of all evil, an important thing to keep in mind when trying to make your web sites faster. Before changing your code, you'll need to determine what is causing it to be slow. You may go through this guide, and many others on optimizing PHP, when the issue might instead be database-related or network-related. By profiling your PHP code, you can try to pinpoint bottlenecks.

Upgrade your version of PHP

The team of developers who maintain the PHP engine have made a number of significant performance improvements over the years. If your web server is still running an older version, such as PHP 3 or PHP 4, you may want to investigate upgrading before you try to optimize your code.

Use caching

Making use of a caching module, such as Memcache, or a templating system which supports caching, such as Smarty, can help to improve the performance of your website by caching database results and rendered pages.

Use output buffering

PHP uses a memory buffer to store all of the data that your script tries to print. This buffer can make your pages seem slow, because your users have to wait for the buffer to fill up before it sends them any data. Fortunately, you can make some changes that will force PHP to flush the output buffers sooner, and more often, making your site feel faster to your users.

Output Buffering Control
Avoid writing naive setters and getters

When writing classes in PHP, you can save time and speed up your scripts by working with object properties directly, rather than writing naive setters and getters. In the following example, the dog class uses the setName() and getName() methods for accessing the name property.

```php
class dog {
    public $name = '';

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }
}
```

Notice that setName() and getName() do nothing more than store and return the name property, respectively.

```php
$rover = new dog();
$rover->setName('rover');
echo $rover->getName();
```

Setting and calling the name property directly can run up to 100% faster, as well as cutting down on development time.

```
$rover = new dog();
$rover->name = 'rover';
echo $rover->name;
```

```
Don't copy variables for no reason

Sometimes PHP novices attempt to make their code "cleaner" by copying predefined variables to variables with
shorter names before working with them. What this actually results in is doubled memory consumption (when the
variable is altered), and therefore, slow scripts. In the following example, if a user had inserted 512KB worth of
characters into a textarea field. This implementation would result in nearly 1MB of memory being used.
```

```
$description = strip_tags($_POST['description']);
echo $description;
```

```
There's no reason to copy the variable above. You can simply do this operation inline and avoid the extra memory
consumption:
```

```
echo strip_tags($_POST['description']);
```

```
Avoid doing SQL queries within a loop

A common mistake is placing a SQL query inside of a loop. This results in multiple round trips to the database,
and significantly slower scripts. In the example below, you can change the loop to build a single SQL query and
insert all of your users at once.
```

```
foreach ($userList as $user) {
    $query = 'INSERT INTO users (first_name, last_name) VALUES("' . $user['first_name'] . '", "' . $user['last_name'] . '")';
    mysql_query($query);
}
```

```
Produces:
```

```
INSERT INTO users (first_name, last_name) VALUES("John", "Doe")
```

```
Instead of using a loop, you can combine the data into a single database query.
```

```
$userData = array();
foreach ($userList as $user) {
    $userData[] = '("' . $user['first_name'] . '", "' . $user['last_name'] . '")';
}
$query = 'INSERT INTO users (first_name, last_name) VALUES' . implode(',', $userData);
mysql_query($query);
```

```
Produces:
```

```
INSERT INTO users (first_name, last_name) VALUES("John", "Doe"),("Jane", "Doe")
```

Scoping Rules

```
Scoping rules should be front and center because I think they are one of the most non-standard and thus confusing
design choices of PHP.

The variables $_GET, $_POST, $_REQUEST, $_SESSION, and $_FILES are superglobals, which means you don't need the
global keyword to use them inside a function.

Functions do not have default access to global variables. You must specify globals as such in your function using
the 'global' keyword.
```

Functions

```
It is possible to define a function from inside another function. The result is that the inner function does not
exist until the outer function gets executed. Classes too can be defined inside functions, and will not exist
until the outer function gets executed.
```

```php
<?php
function a(){
    function b(){
        echo "I am b.\n";
    }
    echo "I am a.\n";
}

if (function_exists("b")) echo "b is defined.\n"; else echo "b is not defined.\n";
a();
if (function_exists("b")) echo "b is defined.\n"; else echo "b is not defined.\n";
?>
```

```
b is not defined.
I am a.
b is defined.
```

```
You can set variable DEFAULT values for a function in the (). These will be over-written by any input values (or
non values).
```

```php
<?php
function default_values_test($a = 123, $b = 456){
    echo "a = ".$a."<br>";
    echo "b = ".$b."<br>";
    echo "<br>";
}

default_values_test(); // uses values set in 'header'
default_values_test('overwritten', 987654321); // uses these values
default_values_test($non_existent, "var A has to be overwritten."); // you can only use this for the last vars.
?>
```

```
a = 123
b = 456

a = overwritten
b = 987654321

a =
b = var A has to be overwritten.
```

Reserved Words

```
PHP has a number of "reserved words," or words that have special meaning in the language. You should avoid using
these words in your code except when using them with their intended meaning.
```

```
<?php
$keywords = array('__halt_compiler', 'abstract', 'and', 'array', 'as', 'break', 'callable', 'case', 'catch', 'class', 'clone',
'const', 'continue', 'declare', 'default', 'die', 'do', 'echo', 'else', 'elseif', 'empty', 'enddeclare', 'endfor', 'endforeach',
'endif', 'endswitch', 'endwhile', 'eval', 'exit', 'extends', 'final', 'for', 'foreach', 'function', 'global', 'goto', 'if',
'implements', 'include', 'include_once', 'instanceof', 'insteadof', 'interface', 'isset', 'list', 'namespace', 'new', 'or', 'print',
'private', 'protected', 'public', 'require', 'require_once', 'return', 'static', 'switch', 'throw', 'trait', 'try', 'unset', 'use',
'var', 'while', 'xor');

$predefined_constants = array('__CLASS__', '__DIR__', '__FILE__', '__FUNCTION__', '__LINE__', '__METHOD__', '__NAMESPACE__',
'__TRAIT__');

// RegEx to find all the keywords:

\b(
(a(bstract|nd|rray|s))|
(c(a(llable|se|tch)|l(ass|one)|on(st|tinue)))|
(d(e(clare|fault)|ie|o))|
(e(cho|lse(if)?|mpty|nd(declare|for(each)?|if|switch|while)|val|x(it|tends)))|
(f(inal|or(each)?|unction))|
(g(lobal|oto))|
(i(f|mplements|n(clude(_once)?|st(anceof|eadof)|terface)|sset))|
(n(amespace|ew))|
(p(r(i(nt|vate)|otected)|ublic))|
(re(quire(_once)?|turn))|
(s(tatic|witch))|
(t(hrow|r(ait|y)))|
(u(nset|se))|
(__halt_compiler|break|list|(x)?or|var|while)
)\b
?>
```

So, you think you know PHP?

```
#. Topic

Question?
```

Quiz Answers

#.

FAQ

String Output

echo vs. *print*

Is a there a difference between what option you use to output your content?

In reality the echo and print functions serve the exact purpose and therefore in the backend the exact same code applies. The one small thing to notice is that when using a comma to separate items whilst using the echo function, items run slightly faster.

Variable Type Checking

isset() vs. *empty()* vs. *is_array()*

What is the performance of isset() and empty()?

isset() and empty() are identical. Always check if the value is set at all before using type-checking because it's twice as slow on a non set value.

E.g. *if (isset($foo) && is_array($foo))*

Quote Types

double (") vs. single (') quotes

Is a there a difference in using double (") and single (') quotes for strings?

In today's versions of PHP it looks like this argument has been satisfied on both sides of the line.

Using the &-ref-operator

...as a so called "alias"

Is it a good idea to use the &-ref-operator to substitute (or alias) a complex mutidim-array?

```
$person = &$aHash["country"]["zip"]["street"]["number"]["name"]
```

Whilst only using a one dimensional array, it's actually faster to use an alias, but anything larger will result in a performance drop.

Counting Loops

for vs. *while*

Is there an actual difference between counting up between the for loop and the while loop?

The while loop 90% of the time is indeed slightly faster.

Using the =&-ref-operator

$obj = $someClass->f() vs. *$obj =& $someClass->f()*

Is a good idea to use the =&-ref-operator when calling a function in an object?

Unless your extremely worried about how much RAM your using, leaving the &-ref-operator out seems like the slightly faster option.

Read Loop:

foreach() vs. *for()* vs. *while(list() = each())*

What is the best way to loop a hash array?

In all cases I've found that the foreach loop is substantially faster than both the while() and for() loop procedures. One thing to note is that when using an entire loop from the start it's extremely good to use the reset() function in all examples.

Given that the previous version of the tests have been very controvercial and incorrect, I must appologise for forgetting to implement the reset() function to allow the while() loops to start from the beginning instead of the end.

Using the =&-ref-operator

$obj = new SomeClass() vs. *$obj =& new SomeClass()*

Is a good idea to use the =&-ref-operator when creating a new object?

There seems to be no difference in performance.

Control Structures

switch/case/default vs. *if/elseif/else*

Is a there a difference between switch and if structures?

Using a switch/case or if/elseif is almost the same. Note: Using === (is exactly equal to) is slightly faster then using == (is equal to).

Counting Loops

For-loop test

Is it worth the effort to calculate the length of the loop in advance?

```
for ($i = 0; $i < $size; $i++)
// instead of
for ($i = 0; $i < sizeOf($x); $i++)
```

Unsurprising results... this is one of the easiest things to implement in any application and is the widest agreed upon benchmarking item within the online PHP community. Always precompute length of arrays.

Modify Loop:

foreach() vs. *for* vs. *while(list() = each())*

What would happen if we alter the reading loop test to test the results of a loop created to simply alter the data in each of the values in the array?

```
// BAD
foreach ($aHash as $key=>$val) $aHash[$key] .= "a";

// BAD
while (list($key) = each($aHash)) $aHash[$key] .= "a";

// GOOD
$key = array_keys($aHash);
$size = sizeOf($key);
for ($i = 0; $i < $size; $i++) $aHash[$key[$i]] .= "a";
```

Proof in this example shows how functionally murderous the foreach() loop can be. If you are modifying your array don't use foreach.

Additional Reading / Watching

Here I provide additional references for you to continue your journey of mastering the best server-side programming language in the world!

https://compilr.com/learn/php-course/introduction-to-php (https://compilr.com/learn/php-course/introduction-to-php)

Recommended YouTube Videos

Next Video ()

- Description

Thanks for reading my book, if you have any questions or suggestions feel free to email me at gerst20051@gmail.com