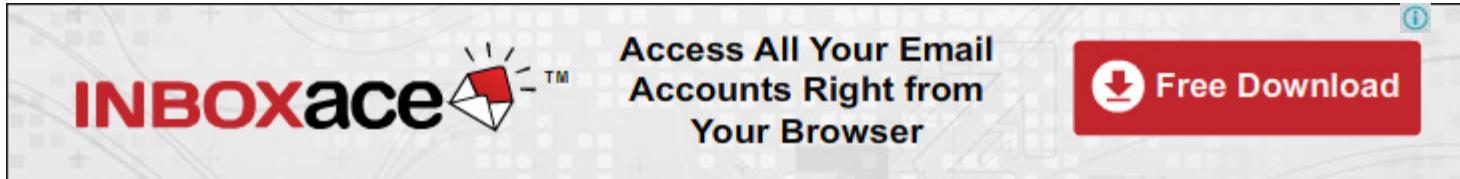


Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Objects

The object-oriented programming (OOP) paradigm has been around for many years now, although the degree to which it is support varies widely across languages. C++, for example, is object-oriented C, and, as some purists would say, implements more OOP functionality than even Java does.

Before PHP 5 came along, OOP support in PHP was quite flaky and more of a hack than a serious attempt. As a result the few that used it often regretted the choice, and it is not surprising that the whole system got a full rewrite in PHP 5 - it is now much more advanced and flexible, and should please just about everyone.

Topics covered in this chapter are:

- Objects and classes defined
- Class inheritance
- Access control
- Runtime type information
- Abstract and final properties and functions
- Constructors and destructors
- Magic functions

## Chapter contents

- 6.1. [Conceptual overview](#)
- 6.2. [Classes](#)
  - 6.2.1. [Defining a class](#)
  - 6.2.2. [How to design your class](#)

- 6.2.3. Basic inheritance
- 6.2.4. Overriding functions
- 6.3. Objects
- 6.4. Variables
- 6.5. The 'this' variable
- 6.6. Objects within objects
- 6.7. Access control modifiers
  - 6.7.1. Public
  - 6.7.2. Private
  - 6.7.3. Protected
  - 6.7.4. Final
  - 6.7.5. Abstract
  - 6.7.6. Iterating through object variables
- 6.8. Object type information
- 6.9. Class type hints
- 6.10. Constructors and destructors
  - 6.10.1. Parent constructors
  - 6.10.2. Destructors
  - 6.10.3. Deleting objects
- 6.11. Copying objects
- 6.12. Comparing objects with == and ===
- 6.13. Saving objects
- 6.14. Magic functions
  - 6.14.1. \_\_autoload()
  - 6.14.2. \_\_get()
  - 6.14.3. \_\_set()
  - 6.14.4. \_\_call()
  - 6.14.5. \_\_toString()
- 6.15. Static data
- 6.16. Helpful utility functions
- 6.17. Interfaces
- 6.18. Deferencing object return values
- 6.19. The Object-Oriented Website
  - 6.19.1. A basic OOP site
  - 6.19.2. A more complex OOP website
- 6.20. Summary
- 6.21. Exercises

6.22. [Further reading](#)

6.23. [Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

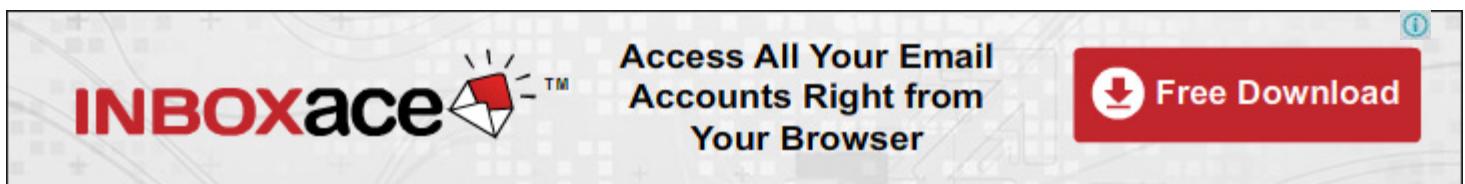
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Conceptual overview >>](#)

Previous chapter: [Next chapter](#)

Jump to: Objects

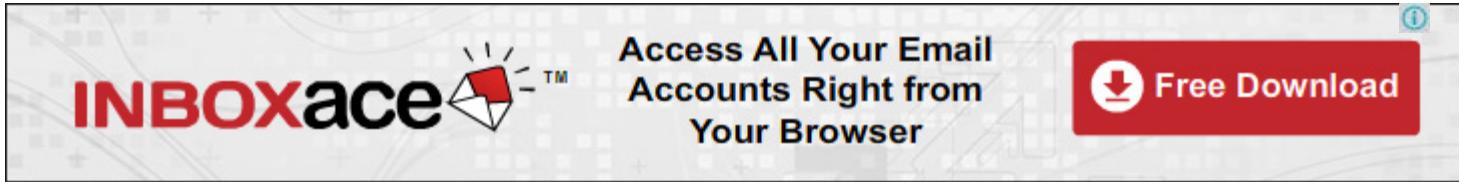
Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word "INBOXace" in red and black with a small envelope icon, and "TM" in blue. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". On the far right is a red button with a white download icon and the text "Free Download". A small blue circular icon with a question mark is in the top right corner of the banner area.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Conceptual overview

OOP was designed to allow programmers to more easily model their programs upon real world scenarios - programmers literally define things (objects) in their world (program), set a few basic properties, then ask them to do things. Consider an object of type "dog" - there are many dogs in the world, but only one animal "dog". As such, we have a blueprint for dogs, from which all dogs are made. While dogs have different breeds that vary a great deal, at the end of the day they all have four legs, a wet nose, and a dislike of cats and squirrels.

So, we have our dog blueprint, from which we might create a Poodle breed, a Chihuahua breed, and an Alsatian breed. Each of these are also blueprints, but they are based upon the Dog blueprint. From our Poodle breed we can then create a Poodle, which we will call Poppy. Poppy is an actual dog, based upon the Poodle breed, and therefore also based upon the Dog blueprint. We can create other Poodles (or indeed Chihuahuas or Alsatians) simply by creating an instance of that breed.

As all dogs are able to bark, we can add a *bark()* function to our dog blueprint, which in turn means that the Poodle breed has a *bark()* function and therefore Poppy can *bark()* too. We can also define variables inside the dog blueprint such as **\$Name**, **\$Age**, and **\$Friendliness** - again, this becomes available in the Poodle breed, which stems from the dog animal, and therefore also into Poppy. Each object of type Poodle would have its own set of variables - its own **\$Name**, its own **\$Age**, etc.

Because the breeds stem from the Dog blueprint, we can also add functions and variables to breeds individually without having them in the Dog blueprint. For example, Poodles come in three general sizes: standard, miniature, and toy. Last time I checked, you don't get toy Alsatians, and so putting a **\$Size** variable into the Dog blueprint would just create a variable that is not used in one-third of the dogs.

If you're still with me, then you're on the way to fully understanding how object-oriented code works - there is a lot more to it, but we'll be getting there!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

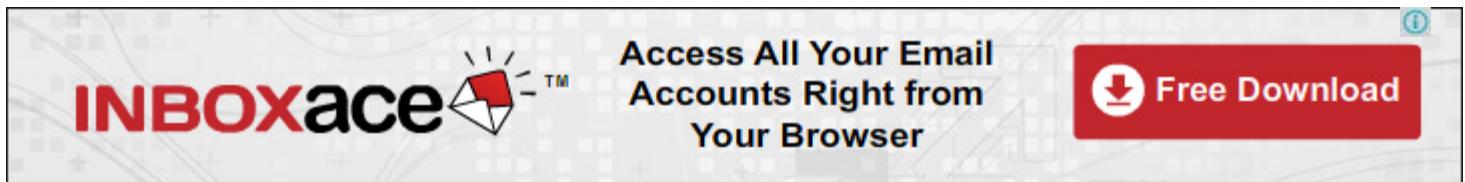
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Classes >>](#)

Previous chapter: [Objects](#)

Jump to: [Conceptual overview](#)

Home: [Table of Contents](#)



The image shows a promotional banner for INBOXace. On the left is the INBOXace logo, which consists of the word "INBOX" in red and "ace" in black, with a small envelope icon containing a play button symbol to the right. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". To the right of that is a red button with a white download icon and the text "Free Download". In the top right corner of the banner is a small blue circular icon with a white question mark.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



**INBOXace** TM

**Access All Your Email  
Accounts Right from  
Your Browser**

**Free Download**

# Classes

The blueprints of dog breeds and animals are known as classes - they define the basic architecture of the objects available in our programs. Each class is defined as having a set of functions and variables, and you can inherit one class from another - our Breed classes, for example, inherited from the Dog class, thereby getting all the Dog functions and variables available. Inheriting is often referred to a sub-classing - "poodle" would be a sub-class of "dog".

Some languages, such as C++, allow you to inherit from more than one class, which is known as multiple inheritance. This technique allows you to have a class bird and a class horse, then create a new class called "Flying Horse", which inherits from both bird and horse, to give you animals like the mythical Pegasus. PHP does not allow you to do this because it generally makes for very confusing programs, and is quite rare even in C++.

PHP allows you to inherit from precisely one parent class, and you can inherit as many times as you want. For example, the dog class could inherit from the class Carnivora, which would contain cats, dogs, bears, etc. Carnivora could inherit from Mammalia, which holds all mammals, which could in turn inherit from Vertebrata, holding all animals with a backbone, etc - the higher up you go, the more vague the classes become. This is because each class inherits the functions and variables from its parent class, as well as adding its own.

**Author's Note:** people often use the terms "parent", "child", "grandparent", etc, to define their class structure. A child class is one that inherits from another - "poodle" is a child of "dog", and would be a grandchild of "carnivora". "Carnivora" would be the parent of "dog" and grandparent of "poodle" - this will make more sense later, when you are creating your own classes and sub-classing freely.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Defining a class >>](#)

Previous chapter: [Conceptual overview](#)

Jump to: [Classes](#)

Home: [Table of Contents](#)



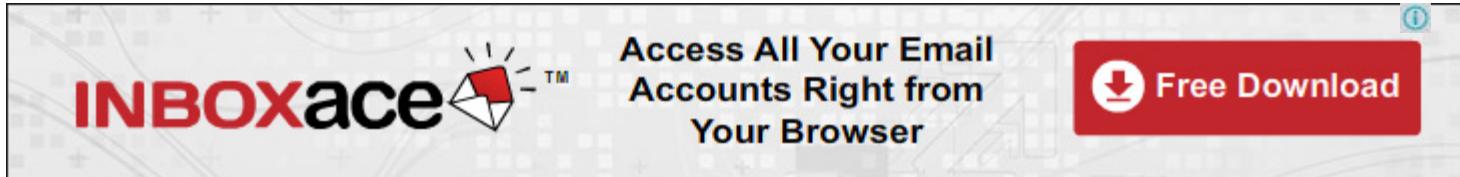
INBOXace TM

Access All Your Email  
Accounts Right from  
Your Browser

[!\[\]\(e8b022b3854e616deefe690b838622fe\_img.jpg\) Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Defining a class

Enough with generalisations - let's get into the specifics. Given the class structure of dogs and breeds discussed above, it's time to take a look at how that translates into PHP code. Here is the PHP code necessary to define a very basic dog class :

```
class dog {
    public function bark() {
        print "Woof!\n";
    }
}
```

Here the dog class has just one function, *bark()*, which outputs "Woof!" - simple enough, really. Don't worry about the "public" part - that just means "can be called by anyone", and we'll be looking at that later. If we create an object of type dog we could call its *bark()* function to have it output the message.

Author's Note: class naming conventions follow the same rules as variable naming, excluding the dollar sign at the beginning. You can use any name for your functions, except "stdClass" and "\_\_PHP\_Incomplete\_Class" - both of these are reserved by PHP.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

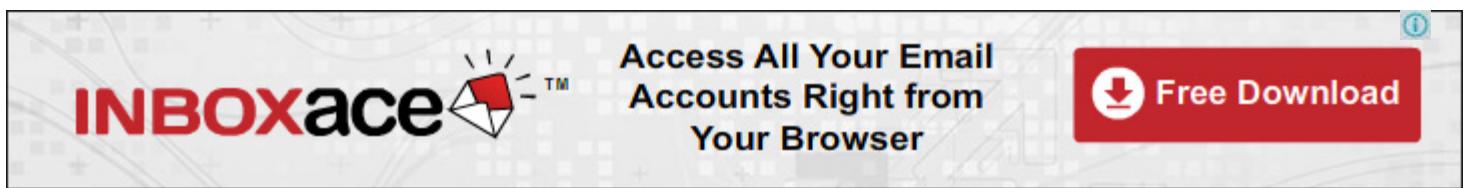
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [How to design your class >>](#)

Previous chapter: [Classes](#)

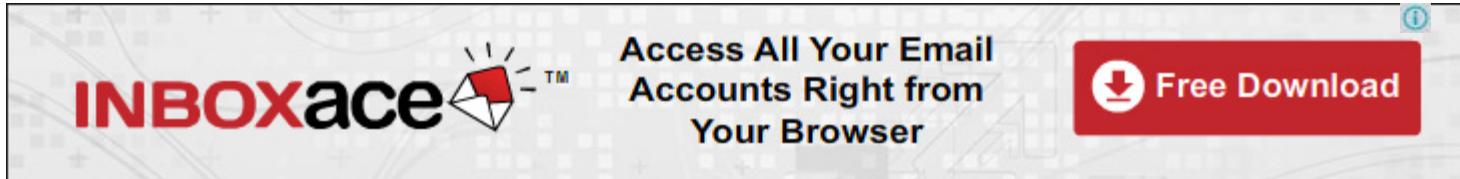
Jump to: [Defining a class](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# How to design your class

When designing your classes, there is one golden rule: keep to real-world thinking. However, although that one rule sounds simple, it's quite nebulous - what exactly is real-world thinking? Fortunately there are a number of more simple rules you can follow that will help keep your code particularly readable:

- Start or end local variables with a special character so that you are always clear about what variable is being set. The most common method is to start local variables with an underscore, e.g. `_Name`, `_Age`, etc.
- To strictly follow OOP guidelines, nearly all of your variables should be either private or protected - they should not be accessible from outside of an object.
- Write accessor functions to set and get private variables. These functions should be how you interface with the object. To get a variable called `_Age`, write a function `Age()`. To set a variable called `_Age`, write a function `SetAge()`.
- Always put variables and functions as low down in your inheritance as they can go without repetition - if you find one object has variables and functions it is not supposed to have, you have gone wrong somewhere. For example, while dolphins can swim, gorillas cannot, so do not put a `swim()` function into a mammal class "just to save time".
- Always keep in mind Muir's law: "When we try to separate anything out by itself, we find it hitched to everything else in the universe". Get your classes distinct and separate from each other to begin with rather than try to hack them apart later on.

If you are wondering why it is that accessor functions should be used to read and write variables, it is because OOP practice dictates that objects should be self-contained - other parts of your program should be able to work with them using simple function calls. For example, imagine you are programming a strategy game where players can control multiple cities. Each city brings in various amount of food depending on

how many workers are there. Now, if the player changes the number of workers in a city to 800, you could just effect the change with code like this:

```
$City->_Workers = 800;
```

However, how would that make any change to the amount of food coming in? It wouldn't, so you would need to make your code this:

```
$City->_Workers = 800; $City->_FoodSurplus = WORKER_SPEED * 800;
```

Now, what if a few weeks later you decide that people can upgrade their cities to build farms and processing facilities that increase the amount of food being made? You'd have to search through your code for all the times you change the \_FoodSurplus variable, and make sure it takes that into account. Later, if you want to make more changes, you need to repeat the procedure again and again and again - the calling code needs to have explicit knowledge of how to handle changes to the number of workers in a city.

This is not how OOP works. In the OOP world, a city would be supposed to handle all these calculations itself, leaving the calling code looking like this:

```
City->SetWorkers(800);
```

The *SetWorkers()* function would contain all the other changes such as altering the \_FoodSurplus level, but the key is that the calling code wouldn't need to know anything about that - any changes to the *SetWorkers()* calculation only needs to be made in one place.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

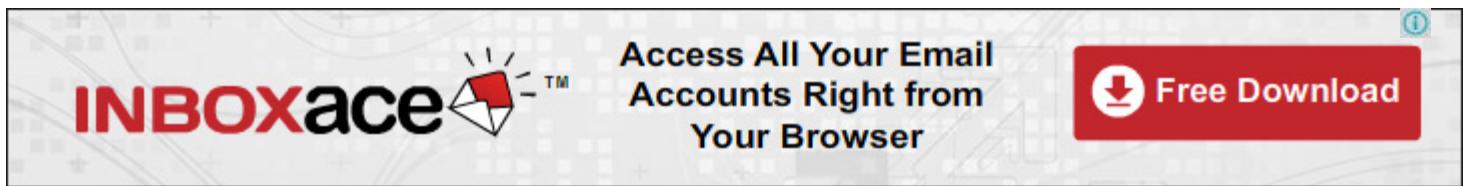
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Basic inheritance >>](#)

Previous chapter: [Defining a class](#)

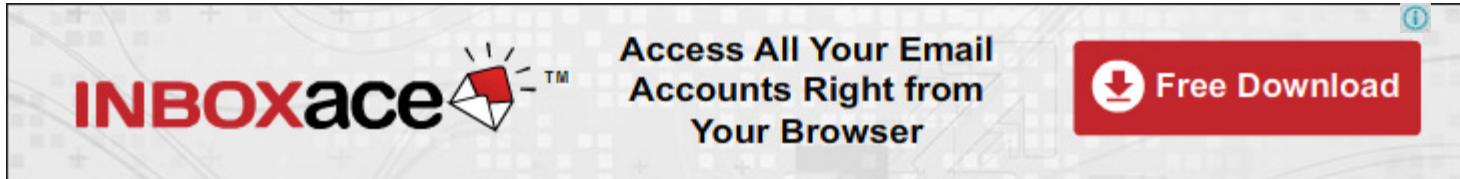
Jump to: [How to design your class](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Hacking with PHP

Welcome to the new home of Practical PHP Programming - now updated for PHP 5.6 and renamed to Hacking with PHP. I've taken this opportunity to brighten up the design, update the content, and make the site much more useful on mobile devices.

While updating the text, I have left chapters in place even if I think they are no longer the smartest option - after all, it's not for me to decide what you should use. In places where I recommend one solution over another, you'll find this clearly marked.

If you enjoy Hacking with PHP, why not [check out my iOS programming book: Hacking with Swift](#).

## Chapter index

[1. Preface](#)

[2. Introducing PHP](#)

[3. Simple variables and operators](#)

[4. Functions](#)

[5. Arrays](#)

[6. Objects](#)

[7. HTML Forms](#)

[8. Files](#)

[9. Databases](#)

[10. Cookies and Sessions](#)

[11. Multimedia](#)

[12. XML & XSLT](#)

[13. Output Buffering](#)

[14. Java and COM](#)

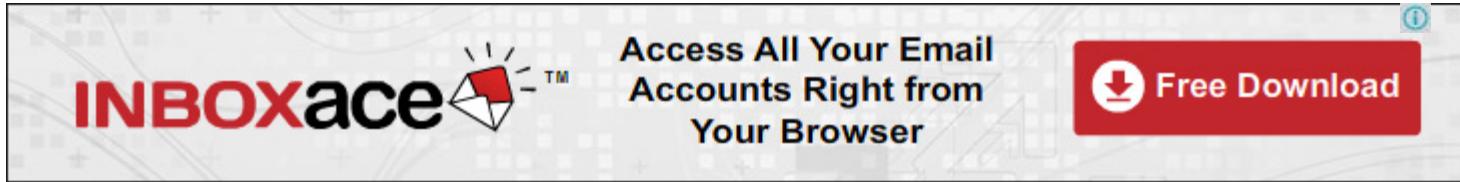
- [15. Networks](#)
- [16. Miscellaneous topics](#)
- [17. Security concerns](#)
- [18. Performance](#)
- [19. Writing PHP](#)
- [20. Writing extensions](#)
- [21. Alternative PHP uses](#)
- [22. Practical PHP](#)
- [23. Bringing it to a close](#)
- [24. Answers to Exercises](#)
- [25. The future of PHP](#)
- [26. Glossary](#)



The banner features the INBOXace logo on the left, which includes the word "INBOXace" in red and black with a small envelope icon and a trademark symbol. In the center, the text reads "Access All Your Email Accounts Right from Your Browser". On the right, there is a red button with a white download icon and the text "Free Download". A small blue info icon is in the top right corner of the banner area.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Basic inheritance

To extend the dog class to breeds, the "extends" keyword is needed, like this:

```
class dog {  
    public function bark() {  
        print "Woof!\n";  
    }  
}  
  
class poodle extends dog {  
    // nothing new yet  
}
```

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

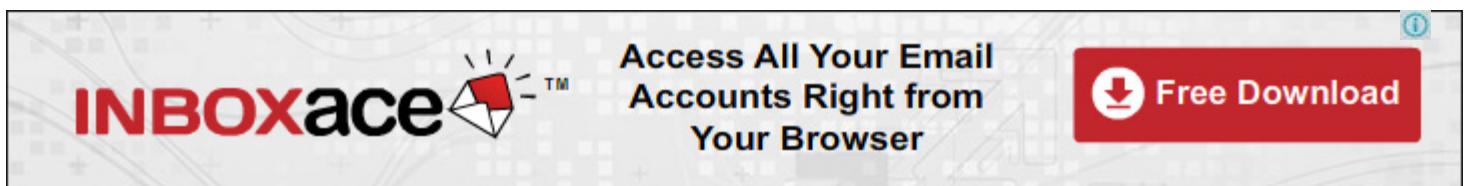
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Overriding functions >>](#)

Previous chapter: [How to design your class](#)

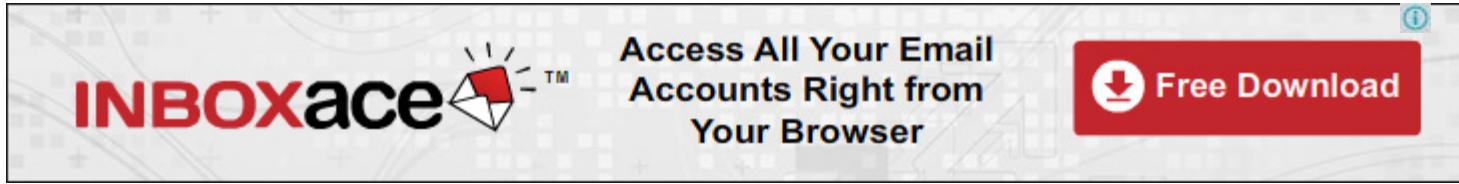
Jump to: Basic inheritance

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Overriding functions

PHP allows us to redefine functions in sub-classes, which means we can make the poodle class have its own version of *bark()*. This is done by simply redefining the function the function inside the child class, making the poodle class look like this:

```
class poodle extends dog {  
    public function bark() {  
        print "Yip!\n";  
    }  
}
```

We'll come back to inheritance and all the neat ways you can use it after we take a look at objects - actual instances of our classes.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

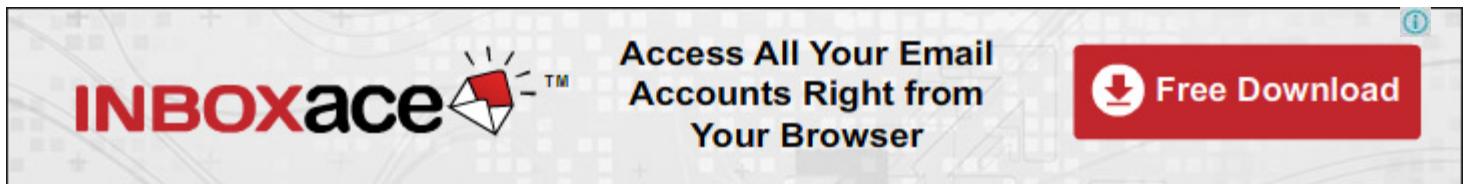
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Objects >>](#)

Previous chapter: [Basic inheritance](#)

Jump to: [Overriding functions](#)

Home: [Table of Contents](#)

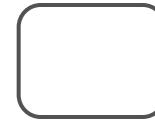


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Product PDF Manual (Free)

Find Manuals For 1000s Of Products For Free. Install AtoZManuals App!



# Objects

Classes are mere definitions - you cannot play fetch with the definition of a dog, you need a real, live, slobbering dog. Naturally we cannot create live animals in our PHP scripts, but we can do the next best thing - creating an instance of our class.

In our earlier example, "Poppy" was a dog of type "poodle". We can create Poppy by using the following syntax:

```
$poppy = new poodle;
```

That creates an instance of the class poodle, and places it into the variable `$poppy`. Poppy, being a dog, can bark by using the `bark()` function, and to do this you need to use the special `->` dereference marker. Here is a complete script demonstrating creating objects - note that the function override for `bark()` is commented out.

```
<?php
    class dog {
        public function bark() {
            print "Woof!\n";
        }
    }

    class poodle extends dog {
        /* public function bark() {
            print "Yip!\n";
        } */
    }

    $poppy = new poodle;
    $poppy->bark();
?>
```

Execute that script, and you should get "Woof!". Now try taking out the comments around the `bark()` function in the poodle class, and running it again you should see "Yip!" instead.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Variables >>](#)

Previous chapter: [Overriding functions](#)

Jump to: [Objects](#)

Home: [Table of Contents](#)

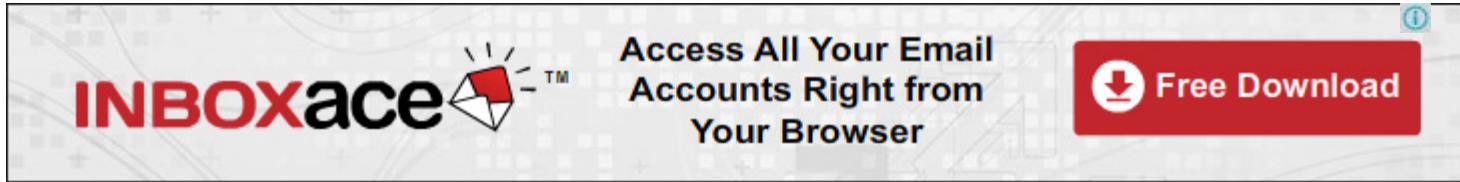


Access All Your Email  
Accounts Right from  
Your Browser

Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Variables

You've seen that it is quite easy to define functions for your classes, and it is just as easy to define variables. Take a look at this new class definition for our dog class:

```
class dog {
    public $Name;

    public function bark() {
        print "Woof!\n";
    }
}
```

The line `public $Name;` is the key - it defines a public variable called `$Name` that all objects of class `dog` will have. PHP allows you to specify how each variable can be accessed, and we will be covering that in depth soon - for now, we will just be using "public".

Author's Note: when designing your own classes, consider putting an underscore in front of object variables so as to distinguish them from parameters passed into an object's functions and also local variables. This is not done here for the sake of maximum readability, plus of course I don't want people who missed this note to think using an underscore is required!

We can now set Poppy's name by using this code:

```
$poppy->Name = "Poppy";
```

Notice that `->` is used again to work with the object `$poppy`, and also that there is no dollar before Name.

The following would be incorrect:

```
$poppy->$Name = "Poppy";
```

PHP's variables have one dollar sign and one only. The reason for this is because the incorrect line of code works like a variable variable - PHP would look up the value of `$Name`, then try to set that value inside `$poppy`. The following code would work, but is kind of crazy:

```
$poppy= new poodle; $foo = "Name"; $poppy->$foo = "Poppy";
print $poppy->Name;
```

Author's Note: Just in case you were skimming over this section and missed out on the warning, PHP's variables have one dollar sign and one only. If you have more, eg `$$foo` or `$foo->$bar`, it means something entirely different and is generally left to the realm of advanced programmers.

As mentioned already, each object has its own set of variables that are independent of other objects of the same type. Consider the following code:

```
$poppy = new poodle; $penny = new poodle; $poppy->Name = "Poppy"; $penny->Name = "Penny";
print $poppy->Name;
```

That will still output "Poppy", because Penny's variables are separate from Poppy's - hopefully you can now see quite how well OOP models real world scenarios!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

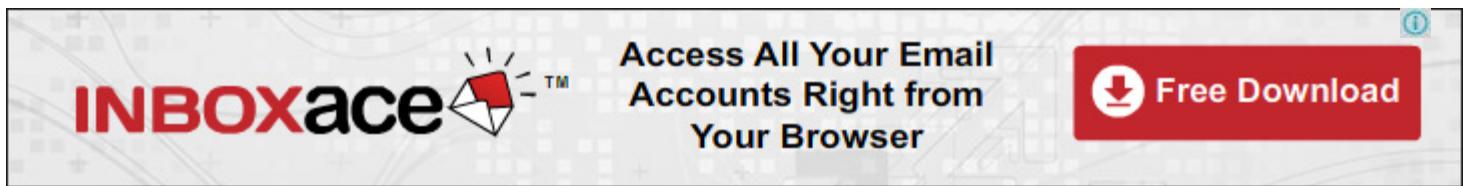
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [The 'this' variable >>](#)

Previous chapter: [Objects](#)

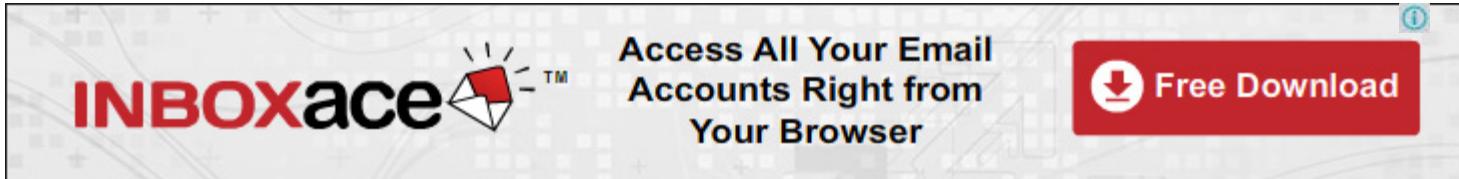
Jump to: [Variables](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# The 'this' variable

Once inside an object's function, you have complete access to its variables, but to set them you need to be more specific than just using the variable name you want to work with. To properly specify you want to work with a local variable, you need to use the special `$this` variable, which PHP always sets to point to the object you are currently working with.

For example:

```
function bark() {
    print "{$this->Name} says Woof!\n";
}
```

Whenever you are inside a function of an object, PHP automatically sets the `$this` variable contains that object - you do not need to do anything to have access to it.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

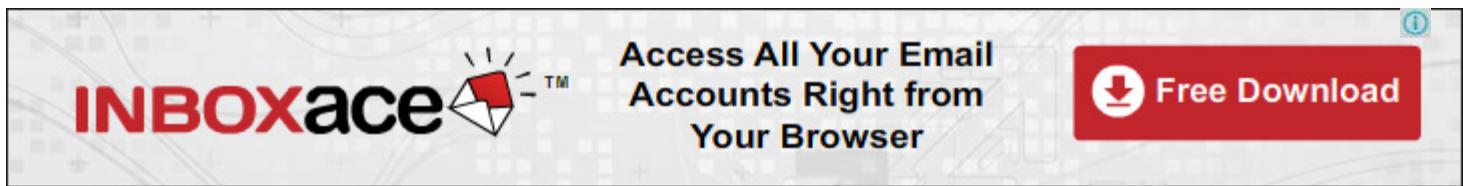
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Objects within objects >>](#)

Previous chapter: [Variables](#)

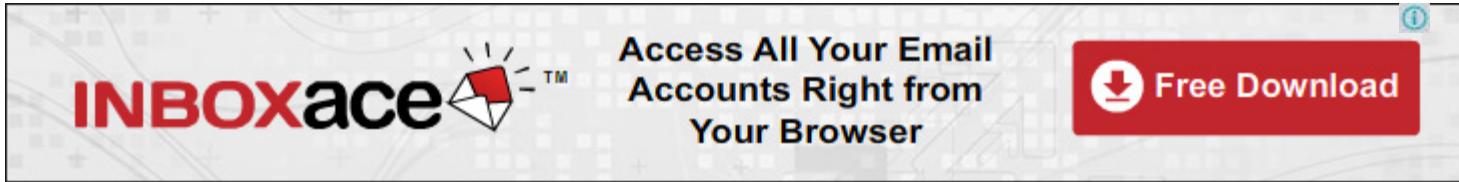
Jump to: [The 'this' variable](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Objects within objects

You can use objects inside other objects in the same as other variable types. For example, we could define a dogtag class and give each dog a dogtag object like this:

```
class dogtag {
    public $Words;
}

class dog {
    public $Name;
    public $DogTag;

    public function bark() {
        print "Woof!\n";
    }
}
```

Accessing objects within objects is as simple as using `->` again:

```
$poppy = new dog; $poppy->Name = "Poppy"; $poppy->DogTag = new dogtag; $poppy->DogTag->Words = "
My name is
Poppy. If you find me, please call 555-1234";
```

Note that `$DogTag` variable is declared like any other, but needs to be created with "new" once `$poppy` has been created.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

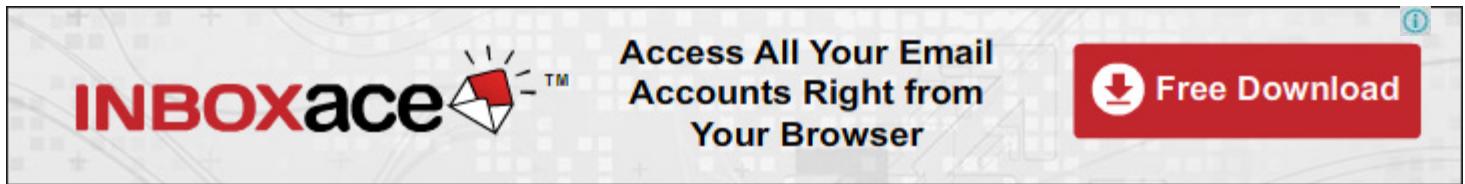
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Access control modifiers >>](#)

Previous chapter: [The 'this' variable](#)

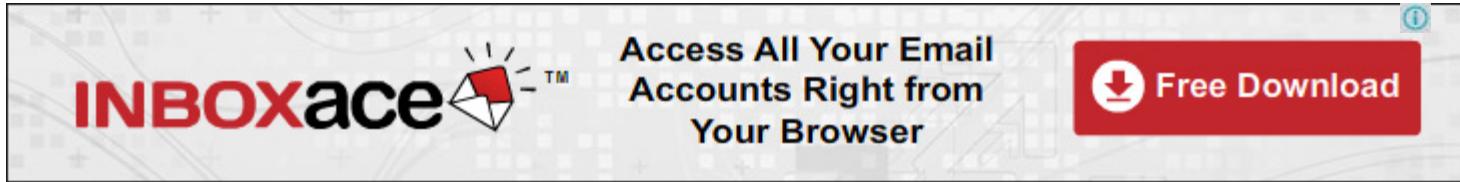
Jump to: [Objects within objects](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Access control modifiers

There are a number of special keywords you can place before a class, a class function definition, or a class variable to alter the way PHP works with it - here's the full list, along with what each of them does:

- Public: This variable or function can be used from anywhere in the script
- Private: This variable or function can only be used by the object it is part of; it cannot be accessed elsewhere
- Protected: This variable or function can only be used by the object it is part of, or descendants of that class
- Final: This variable or function cannot be overridden in inherited classes
- Abstract: This function or class cannot be used directly - you must inherit from them first

That is just a vague description of what each of them do - to make sure you fully understand each of them, here are examples:

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

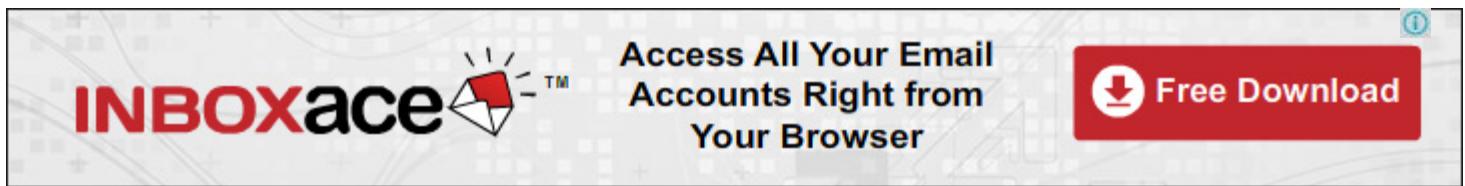
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Public >>](#)

Previous chapter: [Objects within objects](#)

Jump to: [Access control modifiers](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Public

Public variables and functions are accessible from anywhere in your script, which makes this modifier the easiest to use. In PHP 4, all object variables were declared with "var" and were essentially public, but using this terminology is deprecated and may generate compiler warnings. Take a look at this following code:

```
<?php
class dog {
    public $Name;

    public function bark() {
        print "Woof!\n";
    }
}

class poodle extends dog {
    public function bark() {
        print "Yip!\n";
    }
}

$poppy = new poodle;
$poppy->Name = "Poppy";
print $poppy->Name;
?>
```

If you try that code out, you will see it works in precisely the same way as before - the public keyword does not make any difference. The reason behind is that, by default, all class functions are public, because before PHP 5 there was no way to make them anything else.

While the public keyword is not needed, I recommend you use it anyway - it is a good way to remind people who read your code that a given function is indeed public, and also it is possible that class functions without

an access modifier may be deprecated in the future.

When you use public for variables, it is needed - you always need to specify an access modifier for variables, because otherwise there'd be no way to define what variables a class has. Previous versions of PHP used the "var" keyword to declare class variables, again because it had no concept of access modifiers - you should avoid this, and be more specific with public or one of the other keywords.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Private >>](#)

Previous chapter: [Access control modifiers](#)

Jump to:      [Public](#)

Home: [Table of Contents](#)



**INBOXace** ™

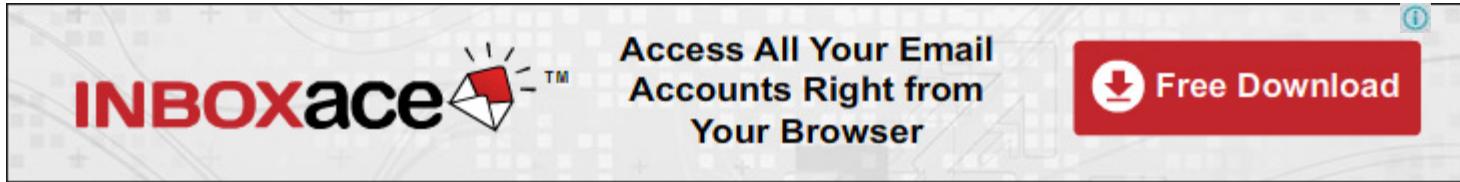
Access All Your Email  
Accounts Right from  
Your Browser

 [Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).



Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Private

The problem with public variables is that they allow functions to be called and variables to set from anywhere within your script, which is generally not a smart thing. Think back to before - we had a dogtag object `$DogTag` inside each dog object as well as a `$Name` variable, but they had repeated information inside.

If we had changed the `$Name` variable, would `$DogTag` have been updated automatically? Of course not - it would have remained the same, which was different to the new `$Name` value:

```
$poppy = new poodle; $poppy->Name = "Poppy"; $poppy->DogTag = new dogtag; $poppy->DogTag->Words = "My name is Poppy. If you find me, please call 555-1234"; $poppy->Name = "Penny"; print $poppy->DogTag->Words;
```

If you try that, you will see the problem. This has arisen because we have allowed any part of our script to directly access the internals of our dog objects. Here is one solution:

```
class dog {
    public $Name;
    public $DogTag;

    public function setName($NewName) {
        $Name = $NewName;
        $DogTag->Words = "My name is $NewName. If you find me, please call 555-1234";
    }
}
```

This time the dog object has embedded logic in that knows how to handle a renaming properly. As long as people use the `setName()` function, the dog's name and its dog tag will get updated with just one call.

Given the above script, however, it is still possible for some unscrupulous, lazy, or ignorant programmer to write `$poppy->Name = "Rover"`, thereby not using the special `setName()` function we've provided. This is where private variables come in - we can instruct PHP that the variable `$Name` is private, and can therefore only be changed with the class its part of. Here is the new code:

```
class dog {
    private $Name;
    private $DogTag;

    public function setName($NewName) {
```

Note that both `$Name` and `$DogTag` are private, which means no one can access them unless doing so in a function that is part of the object, such as `setName()`. Note that `setName()` itself remains public - we want this to be accessible by anyone.

Now if our lazy programmer comes along and tries to set `$Name` directly using code like `$poppy->Name`, they will not get what they were expecting. You see, if they try to alter a private variable directly PHP will automatically spit out an error message. However, if that private variable was inherited from another class, PHP will try to accommodate their request by having a private variable and a public variable. Yes, this is somewhat confusing, however the following code should clear things up:

```
<?php
class dog {
    private $Name;
}

class poodle extends dog { }

$poppy = new poodle;
$poppy->Name = "Poppy";
print_r($poppy);
?>
```

Running that script will output the following:

```
poodle Object
(
    [Name:private] =>
    [Name] => Poppy
)
```

Notice that there are two Name variables - one that is private and cannot be touched, and another that PHP creates for local use as requested. Clearly this is confusing, and you should try to avoid this situation if possible.

Note that private functions and variables can only be accessed by the exact class that owns them - child classes cannot access private parent functions variables. To do this, you need the protected keyword instead.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Protected >>](#)

Previous chapter: [Public](#)

Jump to:      [Private](#)

Home: [Table of Contents](#)



The banner for Marg ERP 9+ features the Marg logo with the tagline 'The Business Backbone'. It highlights 'Most Advance Trade Specific INVENTORY & ACCOUNTING SOFTWARE'. It shows a price of 'Starts Onward ₹ 7,200/-' and mentions '370+ Sales & Support Center Over 6 Lakh User'. A phone number '9999999364' is displayed with a call icon. A small image of a person working on a computer is shown. The banner also includes a 'Best Product 2013' award from The Economic Times and a 'Best Tech Brands 2013' badge.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Protected

Variables and functions marked as protected are accessible only through the object that owns them, whether or not they are declared in that object's class or whether they have descended from a parent class. Consider the following code:

```
<?php
    class dog {
        public $Name;
        private function getName() {
            return $this->Name;
        }
    }

    class poodle extends dog {
        public function bark() {
            print "'Woof', says " . $this->getName();
        }
    }

    $poppy = new poodle;
    $poppy->Name = "Poppy";
    $poppy->bark();

?>
```

In that code, the class poodle extends from class dog, class dog has a public variable `$Name` and a private function `getName()`, and class poodle has a public function called `bark()`. So, we create a poodle, give it a `$Name` value of "Poppy" (the `$Name` variable comes from the dog class), then ask it to `bark()`. The `bark()` function is public, which means we can call it as shown above, so this is all well and good.

However, note that the `bark()` function calls the `getName()` function, which is part of the dog class and was marked private - this will stop the script from working, because private variables and functions cannot be

accessed from inherited classes. That is, we cannot access private dog functions and variables from inside the poodle class.

Now try changing `bark()` to protected, and all should become clear - the variable is still not available to the world as a whole, but handles inheritance as you would expect, which means that we *can* access `getName()` from inside poodle.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Final >>](#)

Previous chapter: [Private](#)

Jump to:      [Protected](#)

Home: [Table of Contents](#)



INBOXace TM

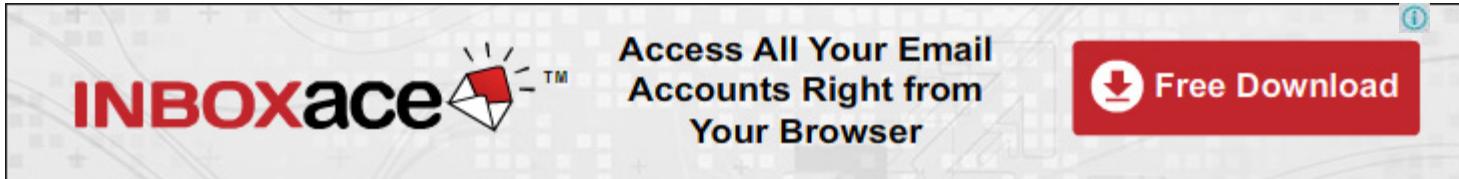
Access All Your Email Accounts Right from Your Browser

[Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).



Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Final

The final keyword is used to declare that a function or class cannot be overridden by a sub-class. This is another way of stopping other programmers using your code outside the bounds you had planned for it.

Take a look at the following code:

```
class dog {
    private $Name;
    private $DogTag;
    final public function bark() {
        print "Woof!\n";
    }
}
```

The dog *bark()* function is now declared as being final, which means it cannot be overridden in a child class. If we have *bark()* redefined in the poodle class, PHP outputs a fatal error message: Cannot override final method *dog::bark()*. Using the final keyword is entirely optional, but it makes your life easier by acting as a safeguard against people overriding a function you believe should be permanent.

For stronger protection, the final keyword can also be used to declare a class as uninheritable - that is, that programmers cannot extend another class from it. Take a look at this script:

```
<?php
final class dog {
    public $Name;
    private function getName() {
        return $this->Name;
    }
}

class poodle extends dog {
```

```
    public function bark() {
        print "'Woof', says " . $this->getName();
    }
?>
```

Attempting to run that script will result in a fatal error, with the message "Class poodle may not inherit from final class (dog)".

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Abstract >>](#)

Previous chapter: [Protected](#)

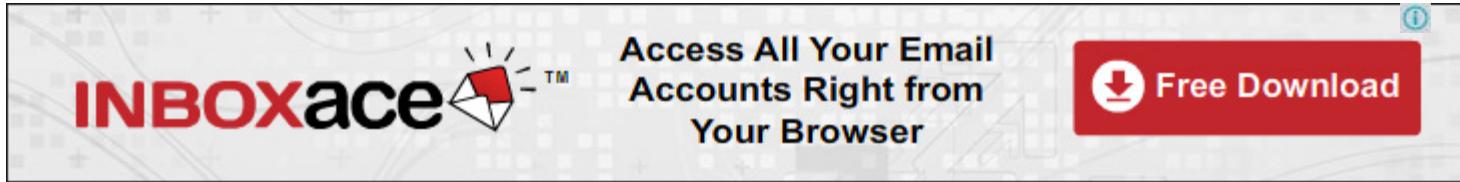
Jump to:      [Final](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Abstract

The abstract keyword is used to say that a function or class cannot be created in your program as it stands.

This might not make sense at first - after all, why bother defining a class then saying no one can use it?

Well, it is helpful because it does not stop people inheriting from that abstract class to create a new, non-abstract (concrete) class.

Consider this code:

```
$poppy = new dog;
```

The code is perfectly legal - we have a class "dog", and we're creating one instance of that and assigning it to **\$poppy**. However, given that we have actual breeds of dog to choose from, what this code actually means is "create a dog with no particular breed". Have you ever seen a dog with no breed? Thought not - even mongrels have breed classifications, which means that a dog without a breed is impossible and should not be allowed.

We can use the abstract keyword to back this up. Here is some code:

```
abstract class dog {
    private $Name; // etc
    $poppy = new dog;
```

The dog class is now abstract, and **\$poppy** is now being created as an abstract dog object. The result? PHP halts execution with a fatal error, "Cannot instantiate abstract class dog".

As mentioned already, you can also use the abstract keyword with functions, but if a class has at least one abstract function the class itself must be declared abstract. Also, you will get errors if you try to provide any

code inside an abstract function, which makes this illegal:

```
abstract class dog {  
    abstract function bark() {  
        print "Woof!";  
    }  
}
```

It even makes *this* illegal...

```
abstract class dog {  
    abstract function bark() { }  
}
```

Instead, a proper abstract function should look like this:

```
abstract class dog {  
    abstract function bark();  
}
```

Author's Note: If it helps you understand things better, you can think of abstract classes as being quite like interfaces, which are discussed shortly.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

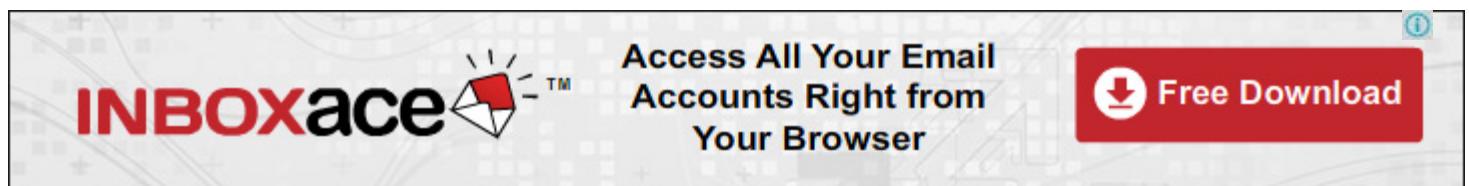
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Iterating through object variables >>](#)

Previous chapter: [Final](#)

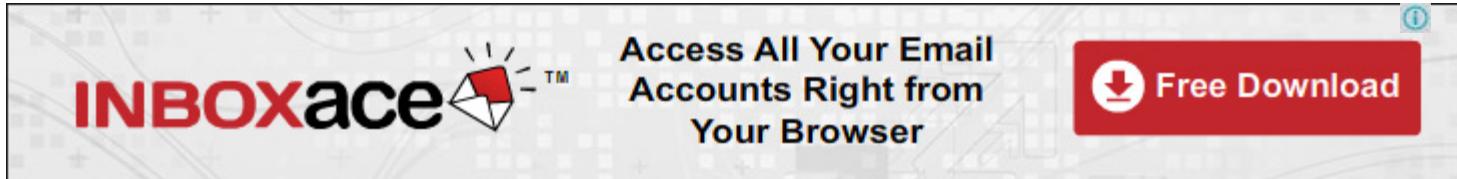
Jump to: [Abstract](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Iterating through object variables

Deep down in the heart of PHP, an object isn't really all that different from an array - an object has pointers to the functions of its class, sure, but otherwise it just stores variables. As a result, we can treat an object as an array with the foreach loop, and it will iterate through each of the variables inside that object - as long as they are accessible. That is, private and protected variables will not be accessible in the general scope.

Take a look at this script:

```
<?php
    class person {
        public $FirstName = "Bill";
        public $MiddleName = "Terence";
        public $LastName = "Murphy";
        private $Password = "Poppy";
        public $Age = 29;
        public $HomeTown = "Edinburgh";
        public $FavouriteColour = "Purple";
    }

    $bill = new person();

    foreach($bill as $var => $value) {
        echo "$var is $value\n";
    }
?>
```

Save that script as arrayobject.php and try running it. Your output should be this:

FirstName is Bill

```
MiddleName is Terence
LastName is Murphy
Age is 29
HomeTown is Edinburgh
FavouriteColour is Purple
```

Note that the `$Password` variable is nowhere in sight, because it is marked Private and we're trying to access it from the global scope. If we re-jig the script a little so that the foreach loop is called by the object, we should be able to see the variable:

```
<?php
class person {
    public $FirstName = "Bill";
    public $MiddleName = "Terence";
    public $LastName = "Murphy";
    private $Password = "Poppy";
    public $Age = 29;
    public $HomeTown = "Edinburgh";
    public $FavouriteColour = "Purple";

    public function outputVars() {
        foreach($this as $var => $value) {
            echo "$var is $value\n";
        }
    }
}

$bill = new person();
$bill->outputVars();
?>
```

Now the output is this:

```
FirstName is Bill
MiddleName is Terence
LastName is Murphy
Password is Poppy
Age is 29
HomeTown is Edinburgh
FavouriteColour is Purple
```

Now that it's the object itself looping through its variables, we can see private variables just fine. Looping through objects this way is a great way to hand-write serialization functions - just remember to put the function inside the object, otherwise private and protected data will get ignored!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Object type information >>](#)

Previous chapter: [Abstract](#)

Jump to: [Iterating through object variables](#)

Home: [Table of Contents](#)



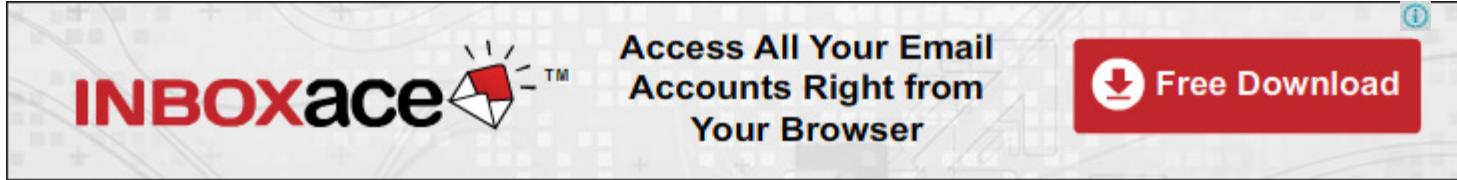
**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

 [Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Object type information

```
bool is_subclass_of ( mixed object, string class_name)
```

As you can see, inheriting from class to class is an incredibly powerful way to build up functionality in your scripts. However, very often it is easy to get lost with your inheritance - how can you tell what class a given object is?

PHP comes to the rescue with a special keyword, "instanceof", which can be used like an operator. Instance of will return true if the object on the left-hand side is of the same class or a descendant of the class given on the right-hand side. For example, given the code \$poppy = new poodle;:

```
if ($poppy instanceof poodle) { }
if ($poppy instanceof dog) { }
```

Both of those if statements would evaluate to be true, because \$poppy is an object of the poodle class, and also is a descendant of the dog class.

**Author's Note:** Java programmers will be happy to know that instanceof is the same old friend they've grown used to over the years. It is a great keyword to keep to hand, as any Java veteran will tell you, and you will almost certainly find yourself using it quite often in your scripts.

If you only want to know whether an object is a descendant of a class, and not of that class itself, you can use the *is\_subclass\_of()* function. This takes an object as its first parameter (or a string containing a class name), a class name string as its second parameter, and returns either true or false depending on whether

the first parameter is descended from the class specified in the second parameter.

Understanding the difference between instanceof and *is\_subclass\_of()* is crucial - this script should make it clear:

```
<?php
    class dog { }
    class poodle extends dog { }
    $poppy = new poodle();
    print (int)($poppy instanceof poodle);
    print "\n";
    print (int)is_subclass_of($poppy, "poodle");
?>
```

That should output a 1 then a 0. Typecasting to int is used because boolean false is printed out as "" (blank), but by typecasting to an integer this becomes 0. As you can see, using instanceof reports true that **\$poppy** is either a poodle or a dog, whereas *is\_subclass\_of()* reports false because **\$poppy** is not descended from the class "poodle" - it *is* a poodle.

**Author's Note:** You can also use the instanceof keyword to see whether an object implements an interface.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

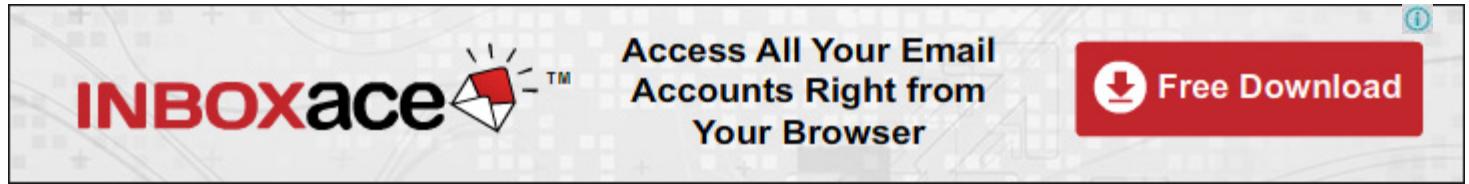
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Class type hints >>](#)

Previous chapter: [Iterating through object variables](#)

Jump to: Object type information

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Class type hints

Although PHP remains a loosely typed language, which means you do not need to specify what types a variable is when it is declared or passed to a function, PHP 5 introduced class type hints, which allow you to specify what kind of class should be passed into a function. These are not required, and are also not checked until the script is actually run, so they aren't strict by any means.

Furthermore, they only work for classes right now - you can't specify, for example, that a parameter should be an integer or a string. Having said that, I suspect future versions may introduce the ability to request that arrays are passed in - that's pure speculation, though! (Edit: this was subsequently added in PHP 5.1 - hurrah!)

Here is an example of a type hint in action:

```
<?php
    class dog {
        public function do_drool() {
            echo "Sluuuuurp\n";
        }
    }

    class cat { }

    function drool(dog $some_dog) {
        $some_dog->do_drool();
    }

    $poppy = new cat();
    drool($poppy);
?>
```

Note that the *drool()* function will accept one parameter, *\$some\_dog*, but that the parameter name is

preceded by the class hint - I have specified that it should only accept a parameter of type "dog". In the example, I have made [\\$poppy](#) a cat object, and so running that will give the following output:

```
Fatal error: Argument 1 must be an instance of dog in C:\home\classhint.php on line 12
```

Note that providing a class hint for a class type that does not exist will cause a fatal error. As you can see, class hints are essentially a way for you to skip having to use the instanceof keyword again and again to verify your functions have received the right kind of objects - using a class hint is essentially implicitly using instanceof, without the extra code. You should make use of class hints regularly as they are a very simple way to make your code regulate itself, and helps solve bugs faster.

Author's Note: As with the instanceof keyword, you can specify an interface as the class hint and only classes that that interface will be allowed through.

As of PHP 5.1, your type hints can include arrays by specifying "array" as the type the function should receive.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

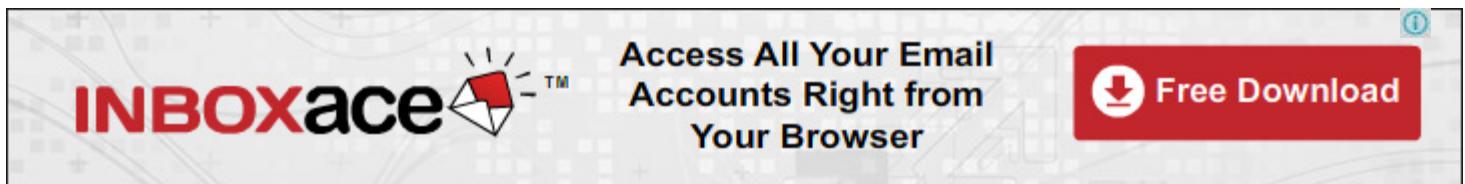
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Constructors and destructors >>](#)

Previous chapter: [Object type information](#)

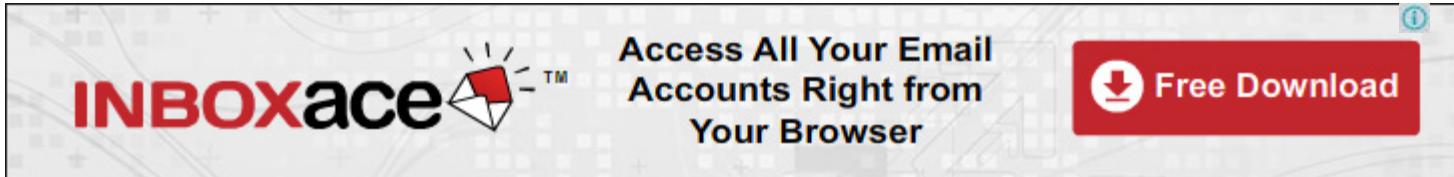
Jump to: [Class type hints](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Constructors and destructors

If you think back to the example where each dog had a dogtag object in it. This led to code like this:

```
$poppy = new poodle; $poppy->Name = "Poppy"; $poppy->DogTag = new dogtag; $poppy->DogTag->Words
= "My name is
Poppy. If you find me, please call 555-1234";
```

That's pretty clumsy, huh? Imagine how more clumsy it would be if we had other objects inside each poodle object - we would need to create the poodle, plus all its other associated objects!

Luckily, there is a way to avoid this, called constructors. A constructor is a special class function that is called by PHP whenever you create an instance of the class. Take a look at this script:

```
<?php
class dogtag {
    public $Words;
}

class dog {
    public $Name;
    public $DogTag;

    public function bark() {
        print "Woof!\n";
    }

    public function __construct($DogName) {
        print "Creating $DogName\n";
        $this->Name = $DogName;
        $this->DogTag = new dogtag;
        $this->DogTag->Words = "My name is $DogName. If you find me, please call 555-1234";
    }
}
```

```

class poodle extends dog {
    public function bark() {
        print "Yip!\n";
    }
}

$polly = new poodle("Poppy");
print $polly->DogTag->Words . "\n";
?>

```

Yes, that's quite a long script, but it's just a combination of the code we've seen so far, with the new twist that constructors are being used. Note the `__construct()` function in the `dog` class, which takes one variable - that is our constructor. Whenever we instantiate a `poodle` object, PHP calls the relevant constructor.

There are three important things to note:

- The constructor is not in the `poodle` class - it is in the `dog` class. What happens is that PHP looks for a constructor in `poodle`, and if it fails to find one there, it goes to its parent class (where `poodle` inherited from), and if it fails to find one there, it goes up again, and up again, ad infinitum, until it reaches the top of the class structure. As the `dog` class is the top of our class structure, PHP does not have far to go.
- PHP only ever calls *one* constructor for you. If you have several constructors in a class structure, PHP will only call the first one it finds.
- The `__construct()` function is marked `public` - that's not by accident. If you don't mark the constructor as `public` you can only instantiate objects of a class from within the class itself, which is almost an oxymoron. If you make this `private`, you need to use a static function call - quite messy, but used in some places.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

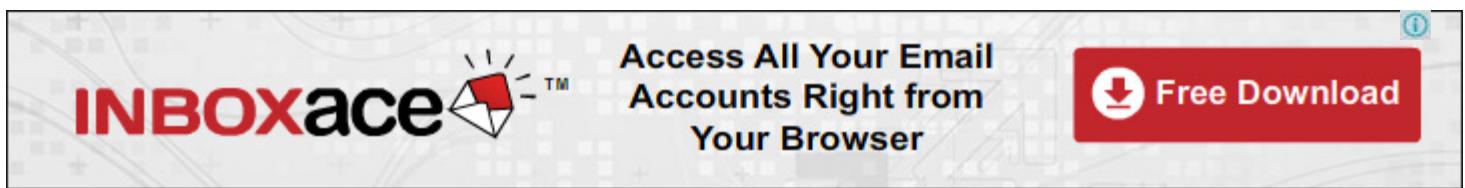
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Parent constructors >>](#)

Previous chapter: [Class type hints](#)

Jump to: Constructors and destructors

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



**INBOXace**™

Access All Your Email  
Accounts Right from  
Your Browser

 Free Download

# Parent constructors

Take a look at this code:

```
class poodle extends dog {
    public function bark() {
        print "Yip!\n";
    }

    public function __construct($DogName) {
        print "Creating a poodle\n";
    }
}
```

If you replace the original poodle definition with this new one and try running the script again, you will get the error "Trying to get property of non-object" on the line where we have `print $poppy->DogTag->Words`. The reason for this is because DogTag is only defined as being an instance of our dogtag object in the dog class constructor, and, as PHP will only ever call one constructor for us, the dog class constructor is not called because PHP finds the poodle constructor first.

The fact that PHP always calls the "nearest" constructor, that is if there is no child constructor it will call the parent constructor and not the grandparent constructor, means that we need to call the parent constructor ourselves. We can do this by using the special function call `parent::__construct()`. The "parent" part means "get the parent of this object, and use it", and the `__construct()` part means "call the construct function", of course. So the whole line means "get the parent of this object then call its constructor".

As you can see, the call to the parent's `__construct()` is just a normal function call, and the dog constructor needs a dog name as its parameter. So, to make the poodle class work properly, we would need the following:

```
class poodle extends dog {  
    public function bark() {  
        print "Yip!\n";  
    }  
  
    public function __construct($DogName) {  
        parent::__construct($DogName);  
        print "Creating a poodle\n";  
    }  
}
```

Try running that - you should get output like to this:

```
Creating Poppy  
Creating a poodle  
My name is Poppy. If you find me, please call 555-1234
```

Note that "Creating Poppy" is output before "Creating a poodle", which might seem backwards, but it makes sense given that we call the dog constructor before we do any poodle code. It is always best to call `parent::__construct()` first from the constructor of a child class in order to make sure all the parent's variables are set up correctly before you try and set up the new stuff.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Destructors >>](#)

Previous chapter: [Constructors and destructors](#)

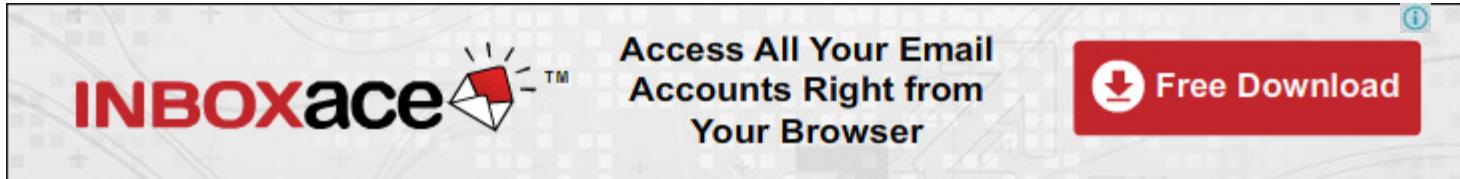
Jump to: Parent constructors

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Destructors

Constructors are very useful, as I am sure you will agree, but there is more: PHP also allows you to define class destructors - a function to be called when an object is deleted. PHP calls destructors as soon as objects are no longer available, and the destructor function, `__destruct()`, takes no parameters.

Take a look at the following code:

```
public function __destruct() {
    print "{$this->Name} is no more...\n";
}
```

If you add that function into the poodle class, all Poodles created will have that function called before being destroyed. Add that into the same script as the constructor we just defined for poodles, and run it again - here's what it outputs:

```
Creating Poppy
Creating a poodle
My name is Poppy. If you find me, please call 555-1234
Poppy is no more...
```

Like constructors, destructors are only called once - you need to use `parent::__destruct()`. The key difference is that you should call `parent::__destruct()` after the local code for the destruction so that you are not destroying variables before using it, for example:

```
public function __destruct() {
    print "{$this->Name} is no more...\n";
    parent::__destruct();
}
```

}

Destructors are useful to free up resources once you are done with them, but they are also good for keeping track of your objects - you can perform any actions you want in destructors, so take advantage of them!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

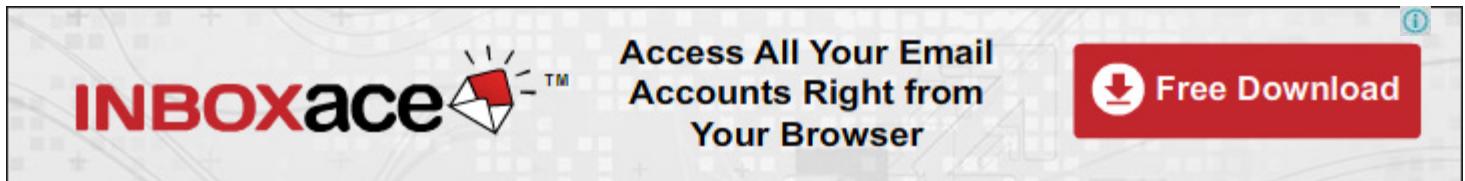
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Deleting objects >>](#)

Previous chapter: [Parent constructors](#)

Jump to:      [Destructors](#)

Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word "INBOXace" in a bold, black, sans-serif font next to a stylized red envelope icon with a "TM" symbol. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". On the far right is a red button with a white download icon and the text "Free Download". A small blue circular icon with a white question mark is located in the top right corner of the banner area.

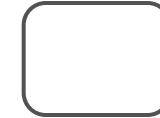
Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).



Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Repair Manual Online

Free Online Instruction Manuals For Popular Brands. Use AtoZManuals  
Now



# Deleting objects

So far our objects have been automatically destroyed at the end of the script they were created in, thanks to PHP's automatic garbage collection. However, you will almost certainly want to arbitrarily delete objects at some point in time, and this is accomplished using *unset()* in the same way as you would delete an ordinary variable.

Calling *unset()* on an object will call its destructor before deleting the object, as you would expect.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

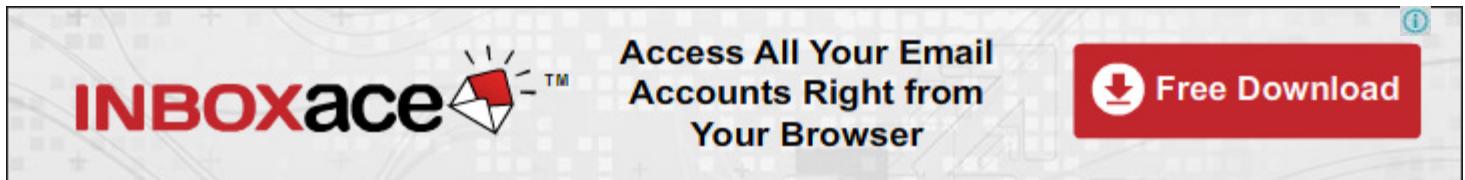
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Copying objects >>](#)

Previous chapter: [Destructors](#)

Jump to:      [Deleting objects](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# Copying objects

In PHP, objects are always handled as references. This means that when you pass an object into a function, any changes you make to it in there are reflected outside the function. For example, consider this piece of code:

```
public function namechange($dog) {  
    $dog->Name = 'Dozer';  
}  
  
namechange($poppy);  
print $poppy->Name . "\n";
```

Here we define a function that accepts one variable, `$dog`, then changes its name to Dozer. We then pass our `$poppy` dog into the function, and output its name - unsurprisingly, it outputs "Dozer" rather than "Poppy". Sometimes it is important to only work on copies of objects - you might not want to affect the state of the original. To do this, we use the built-in keyword "clone", which performs a complete copy of the object. For example, we could use the `namechange()` function above like this:

```
namechange(clone $poppy);
```

That would create a copy of `$poppy` and pass it into `namechange()`, leaving the original `$poppy` untouched. Here is the output of the code now:

```
Creating Poppy  
Creating a poodle  
My name is Poppy. If you find me, please call 555-1234
```

```
Dozer is no more...
```

```
Poppy
```

```
Poppy is no more...
```

Note that Dozer is still mentioned - that is because the copied object passed into `namechange()` gets its name changed to Dozer, then, when the function ends, the copied object is automatically destroyed by PHP, and its destructor is called. However, `$poppy` lives on untouched, as you can see from the last two lines.

Internally, the `clone` keyword copies all the variables from the first object to a new object, then calls a magic function `__clone()` for the class it is copying. You can override `__clone()` if you want, thereby giving you the flexibility to perform extra actions when a variable is copied - you can think of it as a constructor for copied object. Have a look at this piece of code:

```
public function __clone() {
    $this->Name .= '++';
}
```

That function will be called on the copied object, and will set the copied object to have the same name as the original, with `++` tacked onto the end. So, rather than the clone being called Poppy, it will be called Poppy`++`. If we clone the clone, it will be called Poppy`++++`, and so on.

For really advanced functionality, you can also call `parent::__clone()` to work your way up the inheritance chain and call the `__clone()` function of the parent class. Again, all the copying of data is already done, so all the `__clone()` function would be required to do is make any last-minute tweaks to the copy. Here's how that looks:

```
<?php
abstract class dog {
    public function __clone() {
        echo "In dog clone\n";
    }
}

class poodle extends dog {
    public $Name;

    public function __clone() {
        echo "In poodle clone\n";
        parent::__clone();
    }
}

$poppy = new poodle();
```

```
$poppy->Name = "Poppy";  
  
$rover = clone $poppy;  
?>
```

## Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Comparing objects with == and === >>](#)

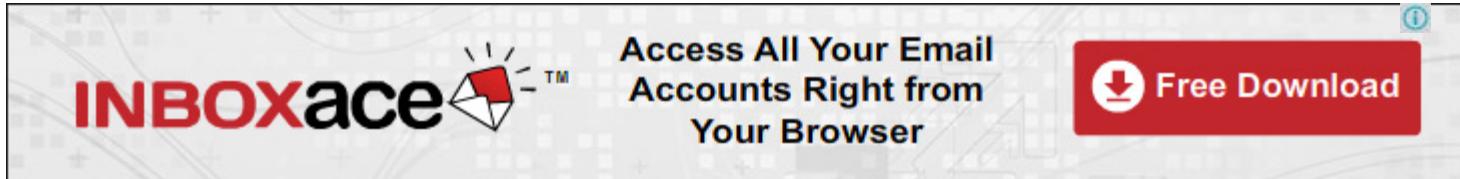
Previous chapter: [Deleting objects](#)

Jump to: [Copying objects](#)

Home: [Table of Contents](#)



Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Comparing objects with == and ===

When comparing objects, == and === may not work quite as you expect them to. If you were comparing two integers of the same value (eg 5), then == and === would both return true, however with objects == compares the objects' contents and === compares the objects' handles.

There is a difference there, and it's crucial: if you create an object and clone it, its clone will have exactly the same values as it. It will, therefore, return true for == as the two objects are the same in terms of their values. However, if you use ===, you will get false back because it compares the handles of the object and finds them to be different. This code example shows it nicely:

```
<?php
class foo { }

$wom = new foo();
$bat = clone $wom;

print (int)($wom == $bat) . "\n";
print (int)($wom === $bat) . "\n";
?>
```

That will output a 1 then a 0, as expected. Now, apart from basic comparison differences, this also matters because very early versions of PHP 5 encountered problems when doing an == comparison in very specific objects. Take a look at this code:

```
<?php
class foo {
```

```

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

$wom = new foo();
$bat = clone $wom;

print (int)($wom == $bat) . "\n";
print (int)($wom === $bat) . "\n";
?>

```

So, what we have there is a class that puts a reference to itself in the `$myself` variable on construction. Naturally, this is a silly thing to do, but the example is simplified - in a real scenario it might store a reference to another object that has a reference back to itself, which would cause the same problem. If you execute that script, you won't get 1 and 0. Instead, you'll get "PHP Fatal error: Nesting level too deep - recursive dependency?" Why? Because with `==`, PHP compares each individual value of the object, so it looks at the value of `$myself`, finds it to be an object, looks inside it, finds `$myself`, looks inside it, finds `$myself`, etc, etc, etc, and carries on looping.

The solution to this is to use `===` in the comparison, which will allow PHP to compare object handles and therefore immediately tell that the two objects are identical.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

[Next chapter: Saving objects >>](#)

[Previous chapter: Copying objects](#)

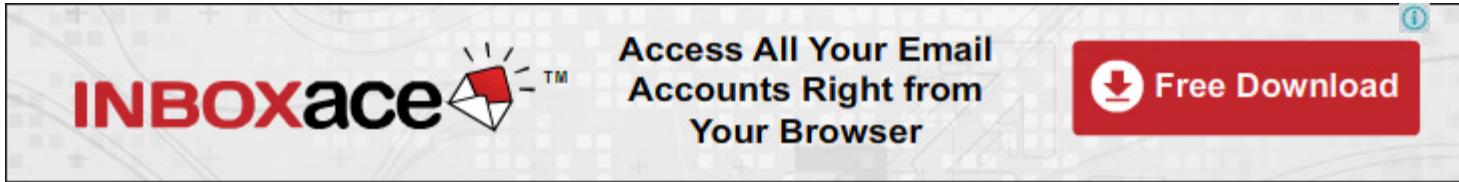
Jump to: Comparing objects with == and ===

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Saving objects

```
array get_object_vars ( object input)
```

Previously we covered how to save arrays in PHP using `serialize()`, `unserialize()`, `urlencode()`, and `urldecode()`. Saving objects works precisely the same way - you `serialize()` them into a string to make a format that can be saved, then `urlencode()` them to get a format that can be passed across the web without problem.

For example:

```
$poppy = new poodle('Poppy');
$safepoppy = urlencode(serialize($poppy));
```

There is one special feature with saving objects, though, and that is the fact that when `serialize()` and `unserialize()` are called, they will look for a `__sleep()` and `__wakeup()` function on the object they are working with respectively. These functions, which you have to provide yourself if you want them to do anything, allow you to properly keep an object working during its hibernation period (when it is just a string of data).

For example, when `__sleep()` is called, a logging object should save and close the file it was writing to, and when `__wakeup()` is called the object should reopen the file and carry on writing. Although `__wakeup()` need not return any value, `__sleep()` must return an array of the values you wish to have saved. If no `__sleep()` function is present, PHP will automatically save all variables, but you can mimic this behaviour in code by using the `get_object_vars()` function - more on that soon.

In code, our logger example would look like this:

```
class logger {
```

```

private function __sleep() {
    $this->saveAndExit();
    // return an empty array
    return array();
}

private function __wakeup() {
    $this->openAndStart();
}

private function saveAndExit() {
    // ...[snip]...
}

```

Any objects of this class that are serialized would have `__sleep()` called on them, which would in turn call `saveAndExit()` - a mythical clean up function that saves the file and such. When objects of this class are unserialized they would have their `__wakeup()` function called, which would in turn call `openAndStart()`.

To have PHP save all variables inside a `__sleep()` function, you need to use the `get_object_vars()` function. This takes an object as its only parameter, and returns an array of all the variables and their values in the object. You need to pass the variables to save back as the values in the array, so you should use the `array_keys()` function on the return value of `get_object_vars()`, like this:

```

private function __sleep() {
    // do stuff here
    return array_keys(get_object_vars($this));
}

```

If there is nothing in your `__sleep()` function other than returning the array, you should delete the function altogether as it is no different to what PHP will do by default.

If you find yourself needing to save objects, keep `__sleep()` and `__wakeup()` in mind - together they allow you to keep objects fully working across pages.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Magic functions >>](#)

Previous chapter: [Comparing objects with == and ===](#)

Jump to: [Saving objects](#)

Home: [Table of Contents](#)



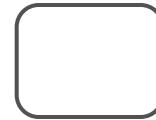
The banner features the INBOXace logo on the left, which includes the word 'INBOX' in red and 'ace' in black, with a small envelope icon and 'TM' next to it. To the right of the logo is the text 'Access All Your Email Accounts Right from Your Browser'. On the far right is a red button with a white download icon and the text 'Free Download'.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Grids, Trees on Your Web?

Don't write any code yourself, made it easy&fast with  
EJSTreeGrid!



# Magic functions

Whenever you see a function name start with a double-underscore, it is a "magic" function - one that PHP has provided that you have not declared yourself. PHP reserves all functions starting with \_\_ as magic, which means while you can use them yourself, you may find that a later version of PHP uses them as a magic function, and causes conflict.

So far we've seen \_\_sleep(), \_\_wakeup(), \_\_clone(), \_\_construct(), and \_\_destruct() - functions that give you special control over your objects that you would not otherwise be able to have. In order to have a full understanding of OOP in PHP there are five more functions you should know about: \_\_autoload(), \_\_get(), \_\_set(), \_\_call(), and \_\_toString().

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [\\_\\_autoload\(\) >>](#)

Previous chapter: [Saving objects](#)

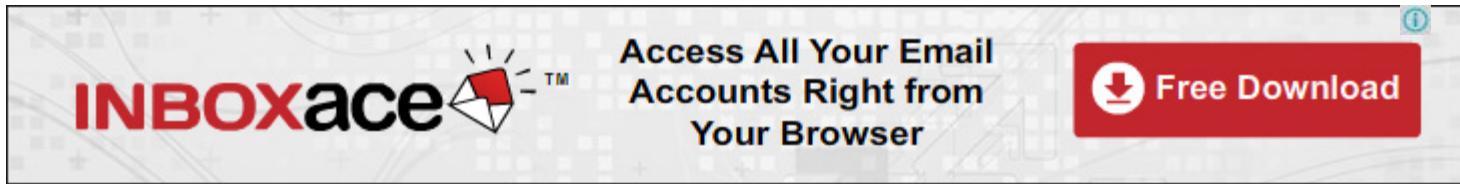
Jump to: [Magic functions](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# \_\_autoload()

This global function is called whenever you try to create an object of a class that hasn't been defined. It takes just one parameter, which is the name of the class you have not defined. If you define an object as being from a class that PHP does not recognise, PHP will run this function, then try to re-create the object - you have a second chance to have the right class.

As a result, you can write scripts like this:

```
<?php
    function __autoload($Class) {
        print "Bad class name: $Class!\n";
        include "barclass.php";
    }

    $foo = new Bar;
    $foo->wombat();
?>
```

Here we try and create a new object of type Bar, but as you can see it does not exist. `__autoload()` is therefore called, with "Bar" being passed in as its first parameter. `__autoload()` then `include()`s the file `barclass.php`, which contains the class definition of Bar. PHP will again try and create a new Bar, and this time it will succeed, which means we can work with `$foo` as normal.

For more advanced scripts, you can try `include()`ing the parameter passed into `__autoload()` - that way you just need to define each class in a file of its own, with the file named after the class. This has been optimised so that calls to `__autoload()` are cached - don't be afraid to make good use of this technique. One of the lead developers of PHP, Andi Gutmans, said, "after having written many examples and worked with it for some time, I'd only ever code this way" - as firm an endorsement as anyone could ask for!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

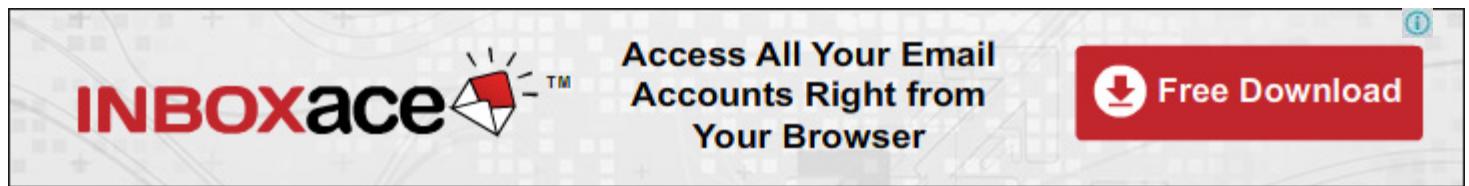
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [\\_\\_get\(\) >>](#)

Previous chapter: [Magic functions](#)

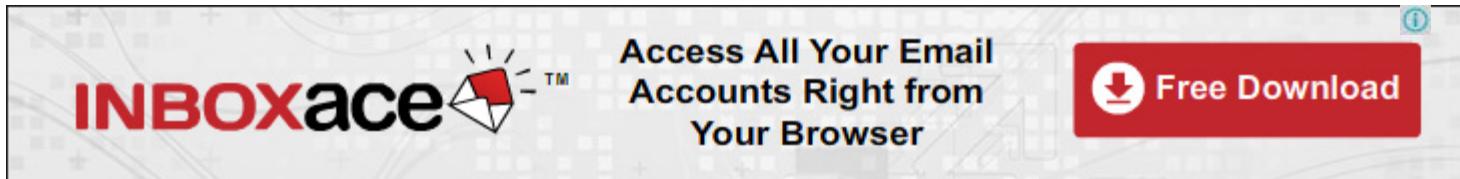
Jump to: [\\_\\_autoload\(\)](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# \_\_get()

This is the first of three slightly unusual magic functions, and allows you to specify what to do if an unknown class variable is read from within your script. Take a look at the following script:

```
<?php
class dog {
    public $Name;
    public $DogTag;
    // public $Age;

    public function __get($var) {
        print "Attempted to retrieve $var and failed...\n";
    }
}

$poppy = new dog;
print $poppy->Age;
?>
```

Note that our dog class has `$Age` commented out, and we attempt to print out the Age value of `$poppy`. When this script is called, `$poppy` is found to not have an `$Age` variable, so `__get()` is called for the dog class, which prints out the name of the property that was requested - it gets passed in as the first parameter to `__get()`. If you try uncommenting the `public $Age;` line, you will see `__get()` is no longer called, as it is only called when the script attempts to read a class variable that does not exist.

From a practical point of view, this means values can be calculated on the fly without the need to create and use accessor functions - not quite as elegant, perhaps, but a darn site easier to read and write.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [\\_\\_set\(\) >>](#)

Previous chapter: [\\_\\_autoload\(\)](#)

Jump to: [\\_\\_get\(\)](#)

Home: [Table of Contents](#)



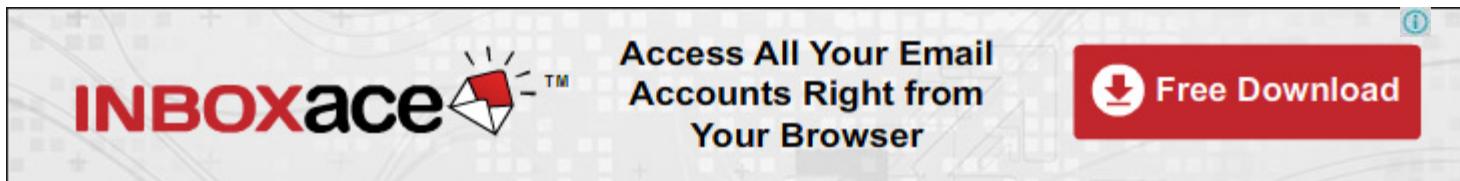
**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

[!\[\]\(1eb8b1d905abf9dc2065a819da74ba8f\_img.jpg\) Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# \_\_set()

The `__set()` magic function complements `__get()`, in that it is called whenever an undefined class variable is set in your scripts. This is a little harder to use with good reason, however, and is more likely to confuse than help.

Here is one example of how you could use `__set()` to create a very simple database table class and perform ad hoc queries as if they were members of the class:

```
<?php
//...[snip - add your MySQL connection code here]...

class mytable {
    public $Name;

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

    public function __set($var, $val) {
        GLOBAL $db;
        mysqli_query($db, "UPDATE {$this->Name} SET $var = '$val';");
    }

    // public $AdminEmail = 'foo@bar.com';
}

$systemvars = new mytable("systemvars");
$systemvars->AdminEmail = 'telrev@somesite.net';
?>
```

In that script `$AdminEmail` is commented out, and therefore does not exist in the `mytable` class. As a result, when `$AdminEmail` is set on the last line, `__set()` is called, with the name of the variable being set and the

value it is being set to being passed in as parameters one and two respectively. This is used to construct an SQL query in conjunction with the table name passed in through the constructor. While this might seem like an odd way to solve the problem of setting key database values, it is pretty hard to deny that the last line of code (\$systemvars->AdminEmail...) is actually very easy to read.

This system could be extended to more complicated objects as long as each object knows their own ID number.

Author's Note: By default, PHP lets you set arbitrary values in objects, even if their classes don't have that value defined. If this annoys you (if, for example, you used OPTION EXPLICIT in your old Visual Basic scripts) you can simulate the behaviour by using `__get()` and `__set()` to print errors.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [\\_\\_call\(\) >>](#)

Previous chapter: [\\_\\_get\(\)](#)

Jump to: [\\_\\_set\(\)](#)

Home: [Table of Contents](#)





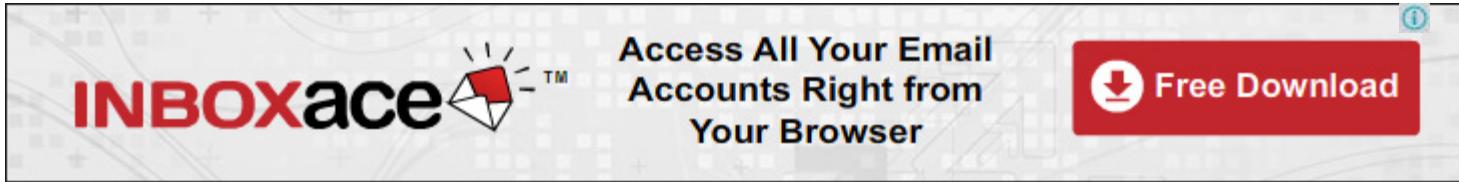
Access All Your Email  
Accounts Right from  
Your Browser



Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# \_\_call()

The *call()* magic function is to class functions what *get()* is to class variables - if you call *meow()* on an object of class *dog*, PHP will fail to find the function and check whether you have defined a *\_\_call()* function. If so, your *\_\_call()* is used, with the name of the function you tried to call and the parameters you passed being passed in as parameters one and two respectively.

Here's an example of *\_\_call()* in action:

```
<?php
class dog {
    public $Name;

    public function bark() {
        print "Woof!\n";
    }

    // public function meow() {
    //     print "Dogs don't meow!\n";
    // }

    public function __call($function, $args) {
        $args = implode(' ', $args);
        print "Call to $function() with args '$args' failed!\n";
    }
}

$poppy = new dog;
$poppy->meow("foo", "bar", "baz");
?>
```

Again, note that the *meow()* function is commented out - if you want to be sure that *\_\_call()* is not used if the function already exists, remove the comments from *meow()*.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

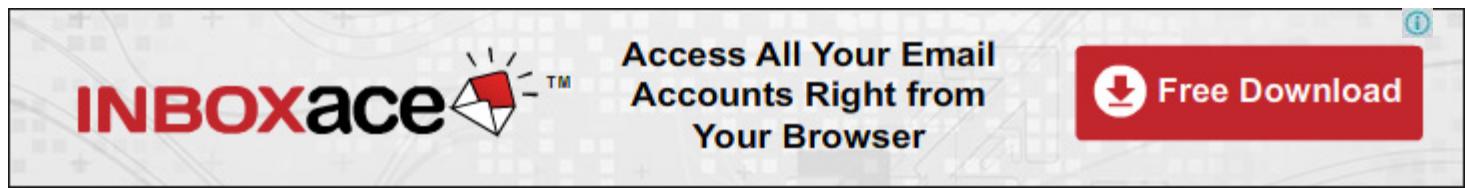
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [\\_\\_toString\(\) >>](#)

Previous chapter: [\\_\\_set\(\)](#)

Jump to: [\\_\\_call\(\)](#)

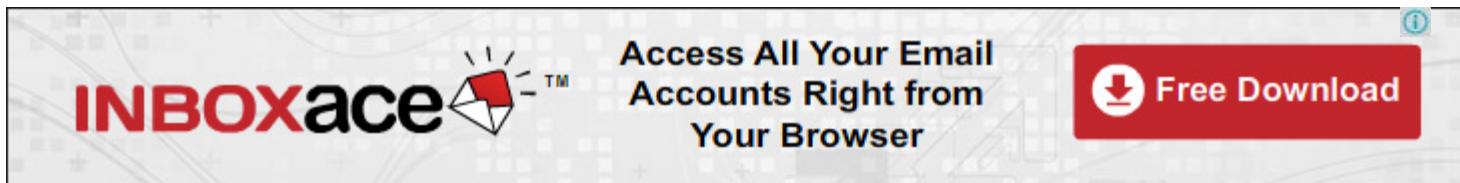
Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word "INBOX" in red, "ACE" in black, and a red envelope icon with "TM" next to it. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". On the far right is a red button with a white download icon and the text "Free Download". A small blue info icon is in the top right corner of the banner.

Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# \_\_toString()

The last magic function you need to know about is `__toString()`, and allows you to set a string value for the object that will be used if the object is ever used as a string. This is a fairly simple magic function, and works like this:

```
<?php
class cat {
    public function __toString() {
        return "This is a cat\n";
    }
}

$toby = new cat;
print $toby;
?>
```

To make this work in PHP 5 caused quite a lot of headaches for the PHP developers - getting the balance right as to when objects should be converted and when they should not took a lot of debating. This feature is quite likely to change in future releases, and if it was not for the fact that it is perfect for use with the SimpleXML extension I really doubt it would have made it into PHP 5 at all.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

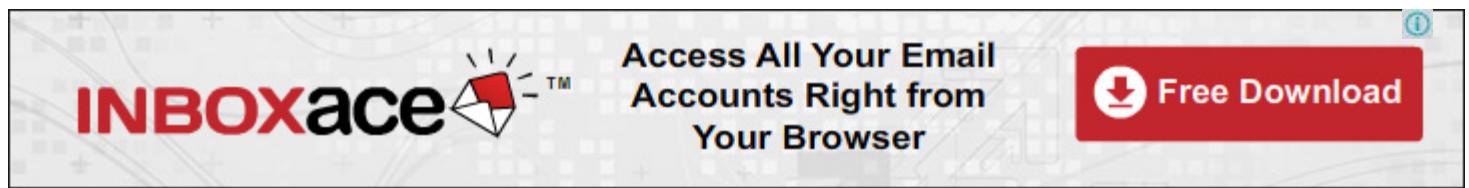
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Static data >>](#)

Previous chapter: [\\_\\_call\(\)](#)

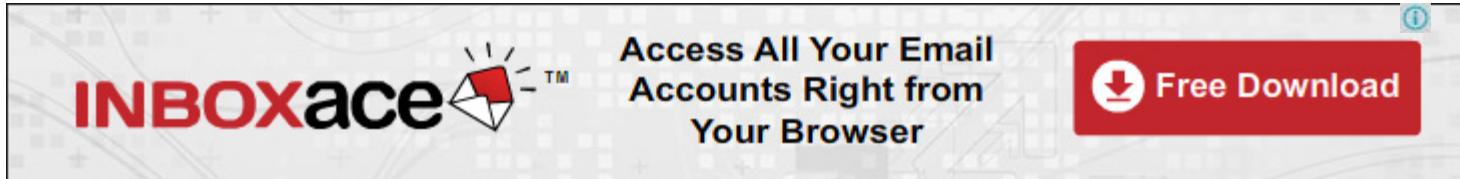
Jump to: [`\_\_toString\(\)`](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Static data

There is one other special thing you can do with OOP in PHP that I have left till last because it is a little hard to get your head around. Put simply, you can declare functions and variables from a class as "static", meaning that they are always available.

For example, if we wanted our *bark()* function to be called as if it were a normal function, we could declare it static. That way, we could call *bark()* directly from the script without the need for any dog objects. This might sound counter-intuitive at first, but it is actually helpful as it allows you to make use of a particularly helpful function without needing to instantiate an object first.

You can also make class variables static, which results in there being only one of that variable for the entire class - all objects share that one variable. So, if we wanted to have an object for every employee in a business, we might have a static variable **\$NextID** that holds the next available employee ID number - when we create a new employee, it takes **\$NextID** for its own **\$ID**, then increments it by one.

To declare your variables and functions as being static, use the "static" keyword. Here is an example:

```
<?php
class employee {
    static public $NextID = 1;
    public $ID;

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

$bob = new employee;
$jan = new employee;
$simon = new employee;

print $bob->ID . "\n";
```

```
print $jan->ID . "\n";
print $simon->ID . "\n";
print employee::$NextID . "\n";
?>
```

That will output 1 2 3 4, which are the employee IDs of Bob, Jan, and Simon respectively, as well as the next available ID number, 4. Note that the scope resolution operator, ::, is used to read the static variable from the employee class.

Also of note is the use of "self" inside the constructor - this refers to the class of the current object, just as earlier on we used "parent" to refer to the parent class of the current object.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Helpful utility functions >>](#)

Previous chapter: [\\_\\_toString\(\)](#)

Jump to: [Static data](#)

Home: [Table of Contents](#)

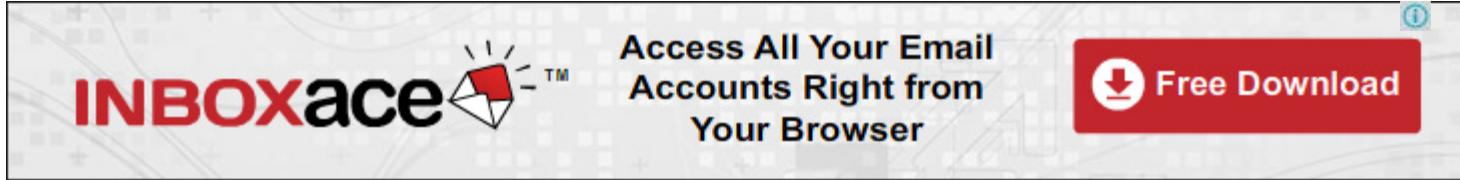


Access All Your Email  
Accounts Right from  
Your Browser

 [Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Helpful utility functions

```
bool class_exists ( string class_name)
bool get_class ( object object)
array get_declared_classes ( void )
```

There are three particular OOP-related functions that will make your life easier, and these are *class\_exists()*, *get\_class()*, and *get\_declared\_classes()*. In order, *class\_exists()* returns true if the specified class has been declared, *get\_class()* returns the class name of the object you pass to it, and *get\_declared\_classes()* returns an array of all classes that you can currently create an object of.

Here are some examples:

```
<?php
    if ($foo == $bar) {
        $sam = new employee;
    } else {
        $sam = new dog;
    }

    print "Sam is a " . get_class($sam) . "\n";
    print "Class animal exists: " . class_exists("animal") . "\n\n\n\n";
    print "All declared classes are: " . get_declared_classes() . "\n";
?>
```

As you can see, the most common use for *get\_class()* is when one object can be of several possible types, as in the code above. C++ users will be familiar with the concept of Runtime Type Information (RTTI), and this is pretty much the same thing.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Interfaces >>](#)

Previous chapter: [Static data](#)

Jump to: Helpful utility functions

Home: [Table of Contents](#)

Put Ads on your blog with Adsense

Get paid for doing what you love



Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Interfaces

```
array get_declared_interfaces ( void )
```

Before you feel all warm and fuzzy about having mastered object-oriented programming in PHP, there is one further thing you need to understand. If you had a class for boats and a class for planes, how would you implement a boat plane class? The functions found in the boat class would be helpful to give you code such as *sink()*, *scuttle()*, *dock()*, etc, and the functions found in the *plane()* class would be helpful to give you code such as *takeoff()*, *land()*, and *bailout()*. What is really needed here is the ability inherit from both the boat class *and* the plane class, a technique known as multiple inheritance.

Sadly, PHP has no support for multiple inheritance, which means it is a struggle to implement this particular scenario. If you find yourself in this situation, there are two options: use two stages of single inheritance, or use an entirely new technique of *interfacing*. We will get to the latter option shortly, but first here is how it would look to have two stages of single inheritance:

```
<?php
    class boat {
        public function sink() { }
        public function scuttle() { }
        public function dock() { }
    }

    class plane extends boat {
        public function takeoff() { }
        public function land() { }
        public function bailout() { }
    }

    class boatplane extends plane {
    }
```

```
$obj = new boatplane();
?>
```

In that example, our grandparent class is "boat", and we inherit all of its functions in class "plane". This is then inherited by class "boatplane", giving the child class all six functions and making it a valid boatplane class. However, it should be clear that the ends do not justify the means: the example above gives our plane class the *dock()* function, which is clearly incorrect.

So, the solution, such as it is, is to use interfaces. An interface can be thought of as an abstract class where you can define sets of abstract functions that will be used elsewhere. If we were to use interfaces in the above example, both boat and plane would be an interface, and class boatplane would implement both of these interfaces. A class that implements an interface has to have concrete functions for each of the abstract functions defined in the interface, so by making a class implement an interface you are in fact saying, "this class is able to do everything the interface says it should". In essence, using interfaces is a way to form contracts with your classes - they must implement functions A, B, and C otherwise they will not work.

The above example could be rewritten using interfaces like this:

```
<?php
interface boat {
    function sink();
    function scuttle();
    function dock();
}

interface plane {
    function takeoff();
    function land();
    function bailout();
}

class boatplane implements boat, plane {
    public function sink() { }
    public function scuttle() { }
    public function dock() { }
    public function takeoff() { }
    public function land() { }
    public function bailout() { }
}

$obj = new boatplane();
?>
```

I strongly recommend you type all that in and try it yourself - it will print out nothing, but we're going to play

around with it now and you will learn the most if you see the results for yourself. Note that there are no access modifiers for the functions in the interface: they are all public by default, because it doesn't make sense to have them as anything else. Similarly you shouldn't try to use abstract or static modifiers with your interfaces - if you get an error like "PHP Fatal error: Access type for interface method *boat::sink()* must be omitted" you know you've gone wrong somewhere.

The first change I would like you to make is to comment out the *bailout()* function in the boatplane class, so that it only has five functions as opposed to six. Now run the script again - what do you get back? If you have done it correctly, PHP should quit with the following fatal error:

```
Fatal error: Class boatplane contains 1 abstract methods and must therefore be declared abstract (plane::bailout)
```

Our boatplane class, by implementing both the boat and plane interfaces, has essentially promised PHP it will have a function *bailout()*. Therefore, PHP gives it one by default - the *bailout()* function from the plane interface. However, as interfaces and their functions are entirely abstract, and by commenting out that one line we have not re-implemented *bailout()* in the boatplane class, the abstract function will be used and thereby make the entire boatplane class abstract - hence the error.

What this has proved is that when a class implements an interface, it makes an unbreakable contract with PHP that it will implement each function specified in that interface. Uncomment the *bailout()* function in the boatplane class, and try commenting out both the boat and plane interfaces, as well as rewriting the boatplane class so that you remove the "implements" part. Now what happens?

This time the script should run fine, just as it did the first time around. Essentially, there is nothing different - the boatplane class has all the same functions as it did before, so why bother with interfaces at all? The key is the "unbreakable contract" aspect, because by having a class implement an interface you know for a fact that it must implement *all* the functions specified in the interface and not just one or two.

The use of interfaces should be considered in the same light as the use of access modifiers - declaring a variable as private changes nothing, really, except that it forces other programmers (and perhaps yourself also) to live up to various expectations about objects of that class. The same applies to interfaces, and, although they are perhaps likely to remain one of the more niche aspects of PHP, they are certainly here to stay.

**Author's Note:** There is one situation in which interfaces actually make a concrete difference to your code, and that's with the Standard PHP Library (SPL). When trying to use functionality from the SPL you must always implement the appropriate interfaces - just implementing the functions isn't good enough.

The function `get_declared_interfaces()` will return an array of all the interfaces currently available to you, and it takes no parameters.

If you really want to delve deep into the world of interfaces, you can also have one interface inheriting from another using the same syntax as you would inherit classes.

As a result, this next script is the same as the previous one, as the plane interface inherits from the boat interface, and the boatplane class implements the plane interface:

```
<?php
    interface boat {
        function sink();
        function scuttle();
        function dock();
    }

    interface plane extends boat {
        function takeoff();
        function land();
        function bailout();
    }

    class boatplane implements plane {
        public function sink() { }
        public function scuttle() { }
        public function dock() { }
        public function takeoff() { }
        public function land() { }
        public function bailout() { }
    }

    $obj = new boatplane();
?>
```

The primary reason for inheriting interfaces is to avoid code like this:

```
class foo implements InterfaceA, InterfaceB, InterfaceC, InterfaceD, .... InterfaceK { }
```

Although it may well be a requirement for the foo class to implement all those interfaces, it does not make for easy reading. As a result, you should try to group interfaces together if possible.

**Author's Note:** It's important to note that although interfaces can extend other interfaces, and classes can implement interfaces, interfaces cannot extend classes. If you try this, you'll get an error along the lines

of "Fatal error: boat cannot implement dog - it is not an interface".

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

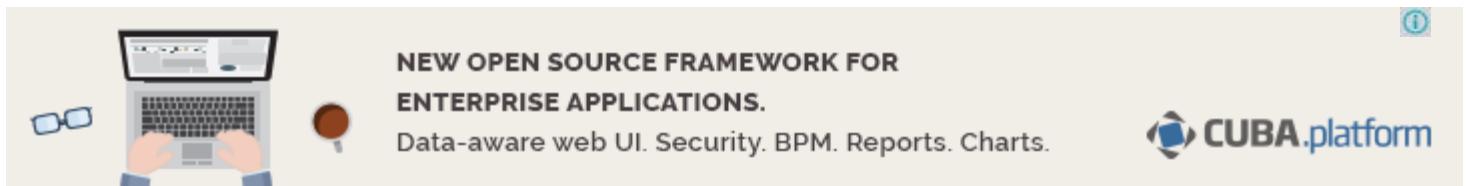
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Deferencing object return values >>](#)

Previous chapter: [Helpful utility functions](#)

Jump to: [Interfaces](#)

Home: [Table of Contents](#)

A horizontal advertisement banner for CUBA.platform. On the left, there's an illustration of a person working at a laptop with a magnifying glass nearby. In the center, the text reads "NEW OPEN SOURCE FRAMEWORK FOR ENTERPRISE APPLICATIONS." and "Data-aware web UI. Security. BPM. Reports. Charts.". On the right, the CUBA.platform logo is displayed, consisting of a blue circular icon with a white 'C' and the text "CUBA.platform".

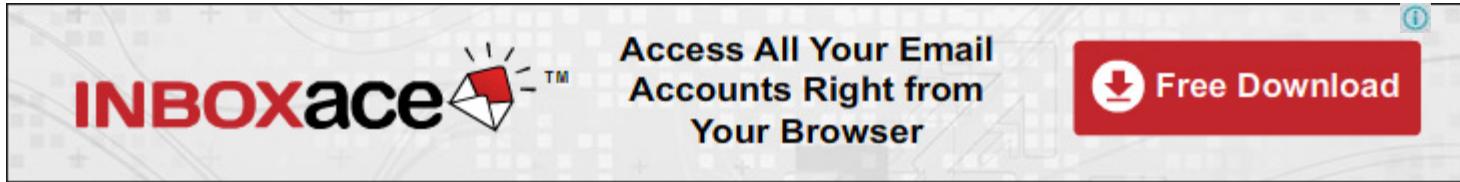
①

**NEW OPEN SOURCE FRAMEWORK FOR  
ENTERPRISE APPLICATIONS.**  
Data-aware web UI. Security. BPM. Reports. Charts.

**CUBA.platform**

Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Deferencing object return values

This is a bit of an odd section - one that doesn't really fit anywhere, but definitely has to go *somewhere*. Dereferencing object return values is a fancy way of saying that if you call a function that returns an object, you can treat the return value of that function as an object from the calling line and access it directly. Did that clear things up? Good, now onto the next topic...

Just kidding - I realise that wasn't the clearest explanation! This code should explain:

```
<?php
    $lassie = new dog();
    $collar = $lassie->getCollar();
    echo $collar->Name;

    $poppy = new dog();
    echo $poppy->getCollar()->Name;
?>
```

In the first example, we need to call `getCollar()` and save the returned value into `$collar`, before echoing out the Name variable of `$collar`. In the second example, we use the return value from `getCollar()` immediately from within the same line of code, and echo out Name without an intermediate variable like `$collar`.

A curious thing about this functionality is that you if you ask a group of people which of the two blocks of code is easier to read, it will split about 50/50 into two groups of people who swear that one way is easier and that the other way is simply impossible to read. So, if you thought the first way was better (or vice versa, like me), do at least keep it in mind that not everyone will agree!

Author's Note: Please remember that, for now at least, return value dereferencing only applies to objects. If you have a function `someFunc()` that returns an array, for example, you cannot use `$obj->someFunc()` [3] to access an element in the return value - you need to store the return value in another variable, then access it. This might be added in the future, but the syntax does look a little clumsy.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [The Object-Oriented Website >>](#)

Previous chapter: [Interfaces](#)

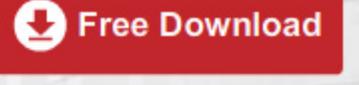
Jump to: [Deferencing object return values](#)

Home: [Table of Contents](#)



**INBOXace** ™

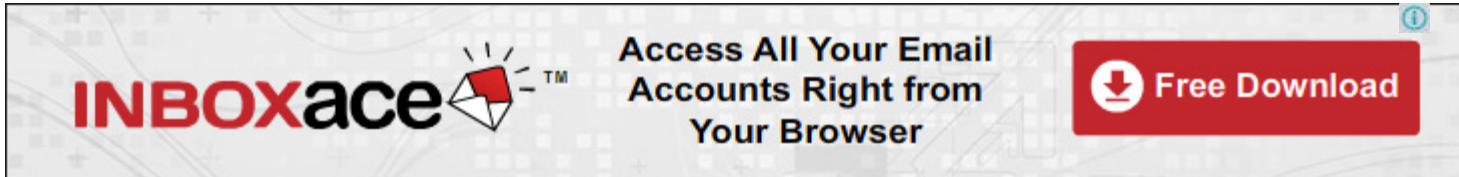
Access All Your Email  
Accounts Right from  
Your Browser

 [Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).



Hacking with PHP has been updated for PHP 7 - only \$20! >>



# The Object-Oriented Website

One of the most-requested additions to this book was a discussion on how to use OOP to render web objects - namely a messageboard, a poll, or even the website's HTML itself. After all, for a book called "Hacking with PHP", there's an awful lot in here about dogs!

The first thing we're going to look at is make your site object-oriented in a very simple way - we'll make a site class and a page class. Then we'll move onto looking at how we could subclass the page class so that we can have different types of page rendering differently. If you're looking for information on how to make an object-oriented messageboard, this is under the Practical PHP chapter, at the end of the section on messageboards.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [A basic OOP site >>](#)

Previous chapter: [Deferencing object return values](#)

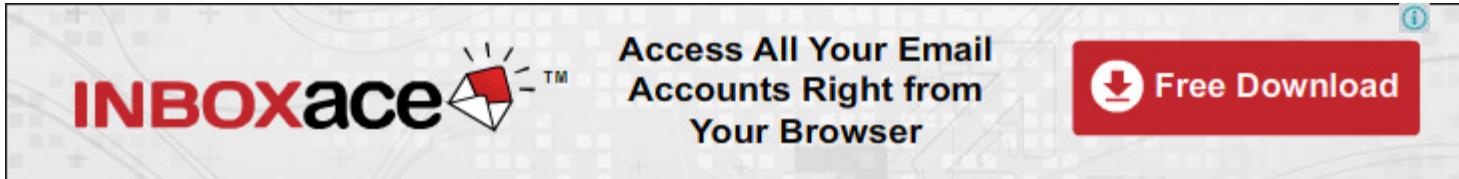
Jump to: [The Object-Oriented Website](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# A basic OOP site

The first thing we're going to produce will be a very simple OOP site that has a site class, csite, and a page class, cpage. These two classes will be kept "clean" in that they won't have any idea of the design of the site around them or their content - they'll just render themselves. This means we'll need several site-specific files as well.

First, let's look at the contents of an example file in our site, index.php:

```
<?php
  include 'stdlib.php';

  $site = new csite();

  // this is a function specific to this site!
  initialise_site($site);

  $page = new cpage("Welcome to my site!");
  $site->setPage($page);

  $content = <<<EOT Welcome to my personal web site! EOT;
  $page->setContent($content);

  $site->render();
?>
```

The stdlib.php file will contain the site-specific information - all the information that shouldn't be inside our classes and shouldn't be repeated in every page. We set `$site` to be an instance of our site, and then call a function called `initialise_site()` to set it up with our basic, site-specific settings - that function will be in stdlib.php.

Next we create a cpage object, passing into the constructor the title of the page, then use the `setPage()`

function of our csite class to add the page to our site for rendering.

The next few lines set up the page-specific content for our cpage, and then calls the *setContent()* function of the object to add the text to the page. Finally, we call the *render()* function of our csite object to output the HTML.

Don't worry if that's all as clear as mud right now - we've yet to look at the other files involved. Here's stdlib.php:

```
<?php
    function __autoload($class) {
        include "$class.php";
    }

    function initialise_site(csite $site) {
        $site->addHeader("header.php");
        $site->addFooter("footer.php");
    }
?>
```

The first part is the magic *\_\_autoload()* function. Note how in index.php we don't specify either csite or cpage - we just presume they exist. This is accomplished through the magic *\_\_autoload()* function, but, when you think about it, that function couldn't go inside one of the classes (there's a recursive bit of logic there!), and neither should it be replicated inside each of the page-specific scripts, as that would be a nightmare to maintain. So, it's in stdlib.php.

The second part is the *initialise\_site()* function, where we add site-specific header and footer files for this site. The *addHeader()* and *addFooter()* functions are both part of the csite class, which we'll look at in a moment. Note that the *\_\_autoload()* function will look for csite.php and cpage.php, so we need to supply those two. Here's csite.php:

```
<?php
class csite {
    private $headers;
    private $footers;
    private $page;

    public function __construct() {
        $this->headers = array();
        $this->footers = array();
    }

    public function __destruct() {
        // clean up here
    }
}
```

```

    public function render() {
        foreach($this->headers as $header) {
            include $header;
        }

        $this->page->render();

        foreach($this->footers as $footer) {
            include $footer;
        }
    }

    public function addHeader($file) {
        $this->headers[] = $file;
    }

    public function addFooter($file) {
        $this->footers[] = $file;
    }

    public function setPage(cpage $page) {
        $this->page = $page;
    }
}

?>
```

The `$headers` and `$footers` variables are both arrays that will store file names of scripts to include at render-time - they are both initialised to be empty arrays when the site is created. The `setPage()` function takes a `cpage` object (note the type hint!), and stores it away for rendering.

The main function here is clearly `render()`, which is what converts the various disparate parts into a working HTML file. It does that by first including each of the scripts stored in `$headers`, then calling the `render()` function of the `cpage` object, then including all the footer files. In reality it's not likely a site will have more than one header and footer file, but there's no harm being flexible!

Next up, the `cpage.php` file - here it is:

```

<?php
class cpage {
    private $title;
    private $content;

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

    public function __destruct() {
        // clean up here
    }
}
```

```

    public function render() {
        echo "<H1>{$this->title}</H1>";
        echo $this->content;
    }

    public function setContent($content) {
        $this->content = $content;
    }
}

?>

```

So we've got `$title` and `$content` variables there, which is predictable enough. Note how the constructor takes the page title as its only parameter - if you look back to index.php you'll see that being used. The `setContent()` function is pretty predictable, as is the main `render()` function itself - it's all very basic right now.

The header.php file is site-specific, and so could be anything you want. Here's an example:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose
.dtd">
<html>
<head>
<title>My Website</title>
</head>

<body>

```

Here's a footer.php example also:

```

</body>
</html>

```

That finishes up the code example - go ahead and run it, and you should see a very simple site being outputted. This is one of the big downsides to using OOP for your site: it takes an awful lot of work to get very little. Of course, once you've done that work, the rest of the site is quite easy!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

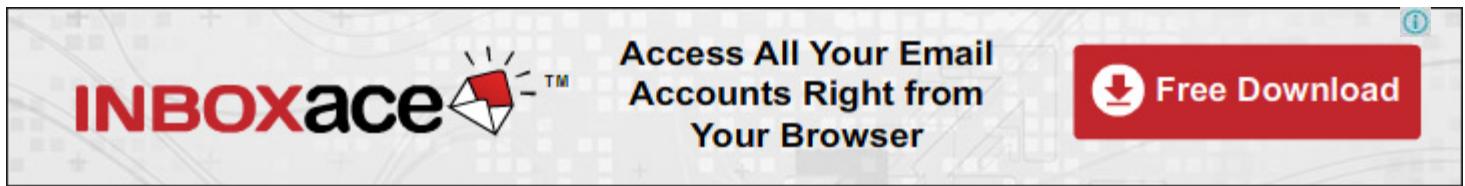
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [A more complex OOP website >>](#)

Previous chapter: [The Object-Oriented Website](#)

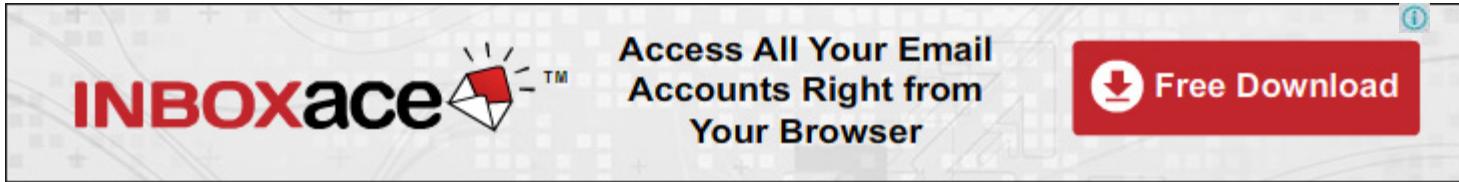
Jump to: [A basic OOP site](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# A more complex OOP website

Now that we've got our very basic site down, I want to briefly discuss how you could extend it to be more interesting. The key here is to take the skeleton framework we have currently and extend it with site-specific classes, so that rather than using generic classes like cpage we can use a specific class like cfrontpage or ctutorialpage.

So, as there will be specific pages to handle each type of page on the site, we no longer really need cpage to be instantiatable. We still *need* the class, because we want to be able to use type hints to check that we have a cpage being used, but we no longer need to create a page without a specific type - people should be using subclasses of the cpage class. Therefore, the first step is to mark cpage as abstract: don't let people create cpage classes any more.

Then you need to decide what types of page there will be: tutorials? Messageboards? Reviews? News? Some may be have the same style, and so can be clumped together into a single class. Once you have a list of the classes that are unique, you can create files for each of them and define them. Make sure they extend from cpage, otherwise the type hints will fail. Then you just need to override the call to *render()* with page-specific rendering - perhaps some pages have a left-hand menu, others have extra footers or a special picture in the header.

If you want to go for really advanced pages, you can even subclass the subclasses - perhaps you have a PHP tutorial series that should have a purple header bar but otherwise have the same as the main tutorials class, so create a cpagetutorialphp class. Don't go too far, though!

One thing you should definitely consider is site-wide database and session management using your csite class. If you always want to have a database connection and session data to hand with your site, you have

two options: take away the purity of the `csite` class and have it always open a connection and start a session, or simply subclass the `csite` class to have a class specifically for the current site, eg: `csitehudzillaorg` for `hudzilla.org`. This would enable you to remove the `initialise_site()` function, because you can hard-code standard elements directly into the `csite` subclass.

Another thing to keep in mind is that you should allow your objects to be used in more ways than you were planning. By that I mean that it's all very well if you create a beautiful object-oriented messageboard, but wouldn't it be even better if you could re-use parts of that code elsewhere - for example, be able to say:

```
$mb = new messageboard("chat");
$mb->getNumMessages();
```

So while your messageboard class is capable of doing all sorts of clever threading and emoticons, it also has a neat little accessor function that grabs the number of messages posted on there so you can print the count on your front page without having to write the necessary SQL queries. Remember, this encapsulation is what makes OOP great: if you had to embed the SQL to grab the number of messages directly into your index page, then it will simply break if the table schema for the messageboard changes. Similarly, if you update the code to check for new messages, the messageboard and the index page get the improvements automatically. OOP, when used correctly, can really make adding features to your site a cinch.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Summary >>](#)

Previous chapter: [A basic OOP site](#)

Jump to:      [A more complex OOP website](#)

Home: [Table of Contents](#)



**INBOXace** TM

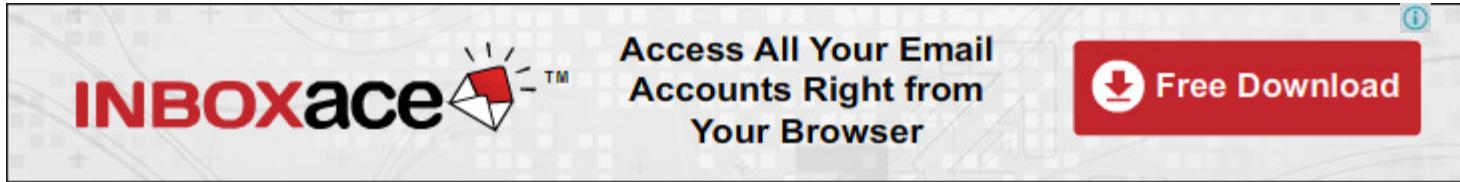
Access All Your Email  
Accounts Right from  
Your Browser

[!\[\]\(ef95a37f44c3c599fa2909992d822352\_img.jpg\) Free Download](#)

①

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Summary

- Using OOP in your PHP scripts allows you to model your code very closely after real life, which make things much easier for you. It also allows you to easily re-use code by sharing your classes.
- PHP has a very rich OOP architecture, much of which you may never find the need to use - if `__get()`, `__set()` and other magic functions seem pointless to you, do not worry about it.
- Use access modifiers like private and protected to make sure your data is accessed precisely as you intend for it to be accessed - don't let other programmers abuse your classes!
- Class inheritance allows you to build up a complicated infrastructure of classes, each building on its parent - this is one of OOP's most valued features, so make good use of it.
- If you are looking for multiple inheritance, you are looking at the wrong language - the best PHP supports is interfaces, which are a simple way to ensure a given class supports a set of functions.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

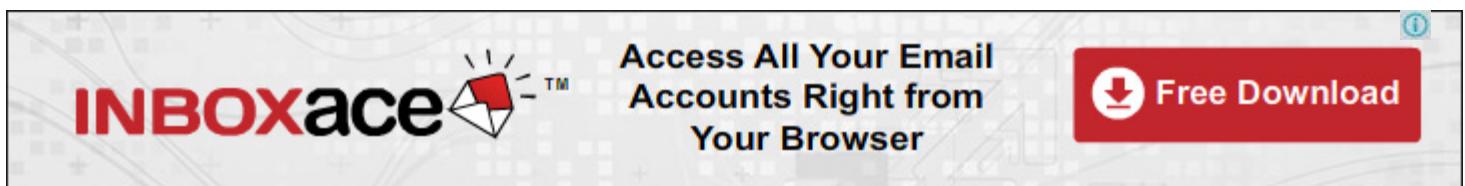
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Exercises >>](#)

Previous chapter: [A more complex OOP website](#)

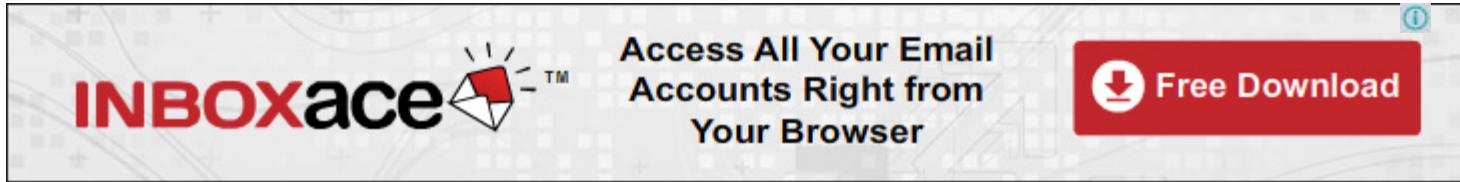
Jump to: [Summary](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Exercises

1. To make a variable shared across all objects of a given class, which keyword do you use?
2. Objects are always passed by reference: true or false?
3. Given that dog is a class, poodle is a class inheriting from dog, and `$poppy` is an object of class poodle, would `$poppy` be considered an instance of the class dog when using the `instanceof` keyword?
4. To list all classes currently available to your script, which function should be used:
  - a) `list_classes()`
  - b) `get_classes()`
  - c) `list_declared_classes()`
  - d) `get_declared_classes()`
5. `Instanceof` and `is_subclass_of()` are identical: true or false?
6. Which of the following statements are true about interfaces:
  - a) They allow you to force classes to implement a set of functions
  - b) They can be based on other interfaces
  - c) They can be based on other classes
  - d) They are faster than using class inheritance
  - e) All of the above
7. When must the "override" keyword be used:
  - a) Whenever a child class overrides a function inherited from its parent

- b) Whenever a child class inherits from an interface
  - c) Whenever a child class implements an interface and inherits from a parent class
  - d) Always
  - e) Never
8. Objects of an abstract class can only be created when the class is declared final: true or false?

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Further reading >>](#)

Previous chapter: [Summary](#)

Jump to: [Exercises](#)

Home: [Table of Contents](#)



**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

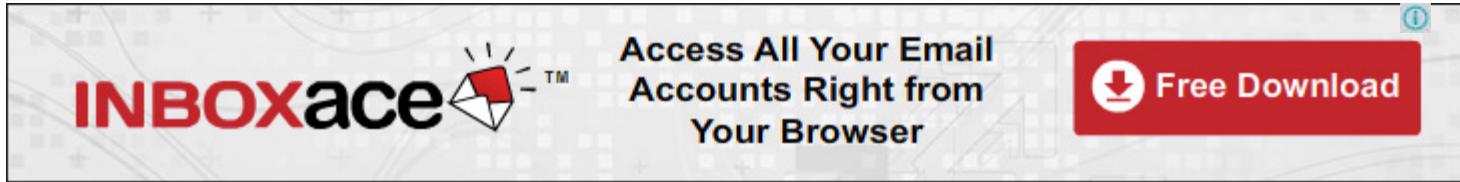
[!\[\]\(547df233bfa443663c2de9b7fe28ce9a\_img.jpg\) Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

Further reading

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Further reading

- In my experience, the best place to look for text on object-oriented programming is beginners' C++ books. Sure, many of the features described there aren't available in PHP, but the ones that are are usually presented in a very slow, very easy to grasp manner with copious amounts of examples to make sure you get the point. My favourite is "Teach yourself C++ in 21 Days" by Jesse Liberty - this has so much content on what OOP is and why it is helpful that nearly all readers can benefit.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

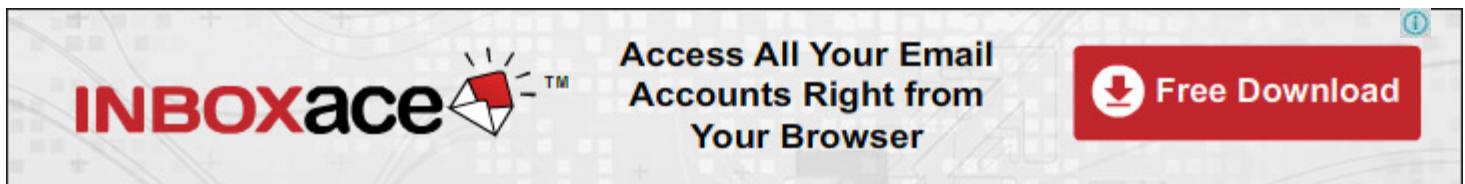
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Next chapter >>](#)

Previous chapter: [Exercises](#)

Jump to: Further reading

Home: [Table of Contents](#)

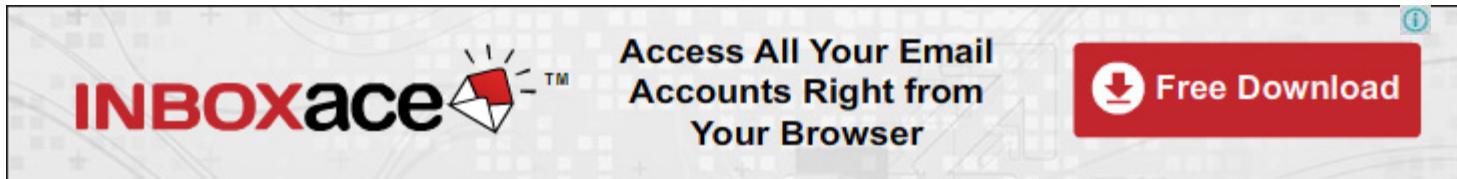


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

Next chapter

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Next chapter

At the beginning of its life, PHP was all about working with data sent in by HTML forms, and there is a lot to learn about how best to interact with your users over the web - all this and more is in next chapter!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

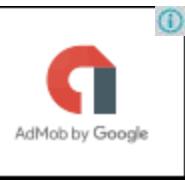
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [HTML Forms >>](#)

Previous chapter: [Further reading](#)

Jump to: [Next chapter](#)

Home: [Table of Contents](#)

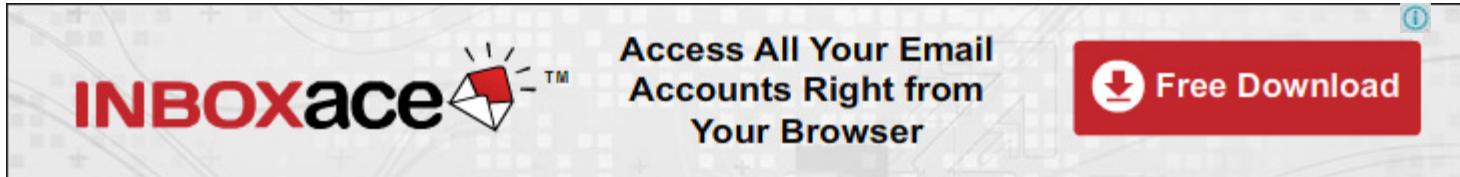


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

Next chapter

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Next chapter

PHP 5 has a more mature implementation of object-oriented programming, which is the practice of architecting your code for maximum reusability by sectioning up functionality into discrete packages. Indeed, the power offered by PHP's OOP support is so great that it deserves a whole chapter to itself, coming up next.

Want to learn PHP 7?

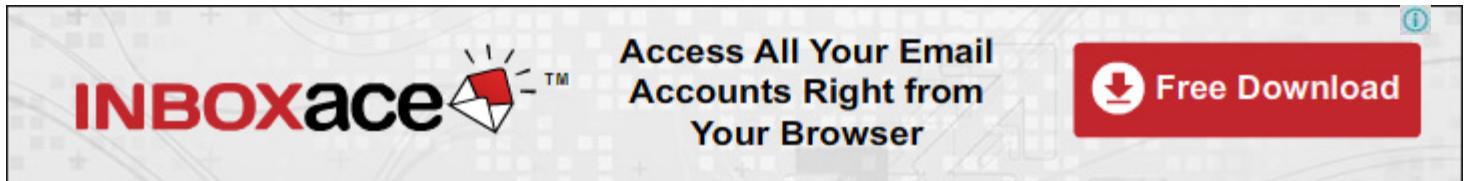
Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Objects >>](#)

Previous chapter: [Further reading](#)

Jump to:    [Next chapter](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# Preface

Welcome, reader, to the world of PHP. You're about to learn to use the most powerful web development language that can be found, and, what's more, you will do so with the minimum of fuss.

I have written this book with the goal of making the task of learning PHP something fun that you don't have to worry about. As such, you will find lots of information for newcomers, even those who have not programmed much before. On the other end of the scale, I have worked hard to put in this book lots of information on advanced functionality in PHP - if you're a veteran user looking to take your PHP skills above and beyond where they are right now, I hope you'll find there is lots to be had here.

My goal for this book has been to produce something you can read in whatever order you want. Once you have the basics down, I encourage you to flick through the table of contents, find something that interests you most, and start reading from there.

## Chapter contents

- 1.1. [Is this book for you?](#)
- 1.2. [Who this book is not for](#)
- 1.3. [What you will get out of this book](#)
- 1.4. [How to use this book](#)
- 1.5. [A note for programmers coming from Perl](#)
- 1.6. [A note for programmers coming from C, C++, or Java](#)
- 1.7. [Cross-platform PHP](#)
- 1.8. [Tips for success](#)
- 1.9. [About the Publisher](#)
- 1.10. [Copyright and usage rights](#)

[1.11. Give something back!](#)

[1.12. About me](#)

[1.13. Acknowledgements](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Is this book for you? >>](#)

Previous chapter: [Table of Contents](#)

Jump to: Preface

Home: [Table of Contents](#)

Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Introducing PHP

The purpose of this chapter is to bring you up to speed on the development of PHP, where it's at now, and the reasons for programming with it. Towards the end of the chapter, you'll see your first PHP code as you start writing your first scripts.

Unless you find history fascinating, I recommend you [skip directly here](#).

## Chapter contents

### 2.1. History

- 2.1.1. [Background](#)
- 2.1.2. [Early versions of PHP](#)
- 2.1.3. [Current release](#)
- 2.1.4. [Upgrading from PHP 3](#)
- 2.1.5. [Upgrading from PHP 4](#)
- 2.1.6. [The creators of PHP](#)
- 2.1.7. [The Zend Relationship](#)

### 2.2. Advantages of PHP

- 2.2.1. [The HTML relationship](#)
- 2.2.2. [Interpreting vs. Compiling](#)
- 2.2.3. [Output Control](#)
- 2.2.4. [Performance](#)
- 2.2.5. [Competing Languages](#)
- 2.2.6. [When to use PHP](#)
- 2.2.7. [When not to use PHP](#)
- 2.2.8. [Selling PHP to your boss](#)

- [2.3. Extending PHP](#)
- [2.4. PEAR](#)
- [2.5. Running PHP scripts](#)
- [2.6. How PHP is written](#)
  - [2.6.1. Whitespace](#)
  - [2.6.2. Escape sequences](#)
  - [2.6.3. Heredoc](#)
  - [2.6.4. Brief introduction to variable types](#)
  - [2.6.5. Code blocks](#)
  - [2.6.6. Opening and closing code islands](#)
  - [2.6.7. Comments](#)
  - [2.6.8. Conditional statements](#)
  - [2.6.9. Case switching](#)
  - [2.6.10. Loops](#)
  - [2.6.11. Infinite loops](#)
  - [2.6.12. Special loop keywords](#)
  - [2.6.13. Loops within loops](#)
  - [2.6.14. Mixed-mode processing](#)
  - [2.6.15. Including other files](#)
- [2.7. Abnormal script termination](#)
- [2.8. Editing your PHP configuration](#)
- [2.9. Summary](#)
- [2.10. Exercises](#)
- [2.11. Further reading](#)
- [2.12. Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [History >>](#)

Previous chapter: [Acknowledgements](#)

Jump to: [Introducing PHP](#)

Home: [Table of Contents](#)

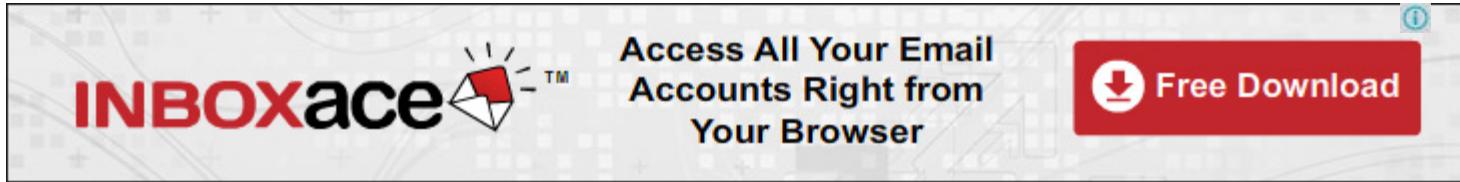
# Grids, Trees on Your Web?

Don't write any code yourself, made it easy&fast with EJSTreeGrid!



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Simple variables and operators

Once you are up and running with your own PHP programming, you'll find that the majority of your scripts are variables, operators, and under ten functions that you use commonly.

In this chapter we will be looking at the different types of variables used in PHP, which to use and when, and also how to convert between the different types. We'll also be looking at operators - these are things that have any effect on variables, such as adding, subtracting, and multiplying.

This chapter is designed to give you complete information on how PHP's variables work - you needn't understand some of the more complicated parts, such as references or variable variables, unless you really want full comprehension of the language.

The topics covered here are:

- What types of data are available to you
- References, typecasting, and variable variables
- Script variables, pre-set variables, script constants, and pre-set constants
- Operators such as plus, minus, multiply, and divide

## Chapter contents

- 3.1. [Types of Data](#)
- 3.2. [Checking a variable is set](#)
- 3.3. [Automatic type conversion](#)

- [3.4. Forcing a type with type casting](#)
- [3.5. Non-decimal number systems](#)
- [3.6. Variable scope](#)
- [3.7. Variable variables](#)
- [3.8. Superglobals](#)
- [3.9. Pre-set variables](#)
- [3.10. References](#)
- [3.11. Constants
  - \[3.11.1. Pre-set constants\]\(#\)](#)
- [3.12. Operators
  - \[3.12.1. Shorthand unary operators\]\(#\)
  - \[3.12.2. Comparison operators\]\(#\)
  - \[3.12.3. Complete operator list\]\(#\)
  - \[3.12.4. The Ternary Operator\]\(#\)
  - \[3.12.5. The scope resolution operator\]\(#\)
  - \[3.12.6. The execution operator\]\(#\)
  - \[3.12.7. Operator precedence and associativity\]\(#\)](#)
- [3.13. Summary](#)
- [3.14. Exercises](#)
- [3.15. Further reading](#)
- [3.16. Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

[Next chapter: Types of Data >>](#)

[Previous chapter: Next chapter](#)

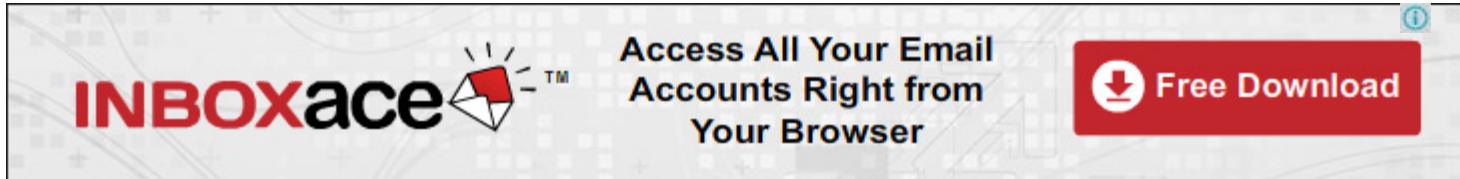
Jump to: Simple variables and operators

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Functions

Functions, both ones built into PHP and ones you define yourself, make coding much easier - they take away lots of hard work because you can reuse other people's code, and they allow you to keep your scripts shorter and easier to maintain. As PHP 5 includes more than 2,500 functions, you might assume it's a very easy language indeed, but the truth is that each function needs to be used in different ways and so needs to be learnt individually. In this chapter you will learn your first PHP functions, with the most helpful and easy first.

Rather than writing pieces of code time after time whenever you want to execute the same functionality, PHP allows you to encapsulate code into a named function that you can call from elsewhere in your script.

PHP comes with hundreds of predefined functions that perform all manner of tasks from reading files and manipulating strings up to querying databases and connecting to an IRC server. If you find something is missing, you can add your own functions on a script basis, and these are called *user functions*.

In this section we will be covering a variety of the most important basic functions in PHP - more specialised functions can be found spread throughout the book under various sections, and should be looked up using the index.

Topics covered in this chapter are:

- Working with date and time
- Mathematical functions
- String manipulation
- Creating data hashes
- Regular expressions
- Extension handling

- Writing your own functions
- Recursive, variable, and callback functions

## Chapter contents

- 4.1. [Functions overview](#)
- 4.2. [How to read function prototypes](#)
- 4.3. [Working with variables](#)
- 4.4. [Controlling script execution](#)
- 4.5. [Working with Date and Time](#)
  - 4.5.1. [Reading the current time](#)
  - 4.5.2. [Converting from a string](#)
  - 4.5.3. [Converting to a string](#)
  - 4.5.4. [Converting from components](#)
- 4.6. [Mathematics](#)
  - 4.6.1. [Rounding](#)
  - 4.6.2. [Randomisation](#)
  - 4.6.3. [Trigonometrical conversion](#)
  - 4.6.4. [Other mathematical conversion functions](#)
  - 4.6.5. [Base conversion](#)
  - 4.6.6. [Mathematical constants](#)
- 4.7. [Playing with strings](#)
  - 4.7.1. [Reading from part of a string](#)
  - 4.7.2. [Replacing parts of a string](#)
  - 4.7.3. [Converting to and from ASCII](#)
  - 4.7.4. [Measuring strings](#)
  - 4.7.5. [Finding a string within a string](#)
  - 4.7.6. [Returning the first occurrence of a string](#)
  - 4.7.7. [Trimming whitespace](#)
  - 4.7.8. [Wrapping your lines](#)
  - 4.7.9. [Changing string case](#)
  - 4.7.10. [Making a secure data hash](#)
  - 4.7.11. [Alternative data hashing](#)
  - 4.7.12. [Automatically escaping strings](#)
  - 4.7.13. [Pretty-printing numbers](#)
  - 4.7.14. [Removing HTML from a string](#)

- 4.7.15. Comparing strings
- 4.7.16. Padding out a string
- 4.7.17. Complex string printing
- 4.7.18. Parsing a string into variables
- 4.8. Regular expressions
  - 4.8.1. Basic regexes with preg\_match() and preg\_match\_all()
  - 4.8.2. Novice regexes
  - 4.8.3. Advanced regexes
  - 4.8.4. Guru regexes
  - 4.8.5. Regular expression replacements
  - 4.8.6. Regular expression syntax examples
  - 4.8.7. A regular expression assistant
- 4.9. Checking whether a function is available
- 4.10. Extension functions
- 4.11. Pausing script execution
- 4.12. Executing external programs
- 4.13. Connection-related functions
- 4.14. Altering the execution environment
- 4.15. User functions
  - 4.15.1. Return values
  - 4.15.2. Parameters
  - 4.15.3. Passing by reference
  - 4.15.4. Returning by reference
  - 4.15.5. Default parameters
  - 4.15.6. Variable parameter counts
- 4.16. Variable scope in functions
- 4.17. Overriding scope with the GLOBALS array
- 4.18. Recursive functions
- 4.19. Variable functions
- 4.20. Callback functions
- 4.21. The declare() function and ticks
- 4.22. Handling non-English characters
- 4.23. Undocumented functions
- 4.24. Summary
- 4.25. Exercises
- 4.26. Further reading
- 4.27. Next chapter

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

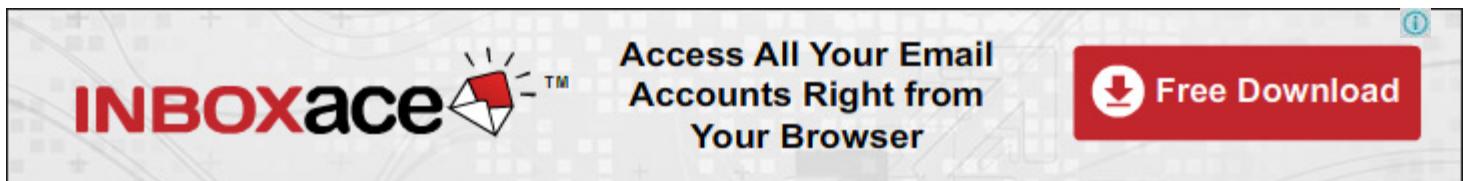
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Functions overview >>](#)

Previous chapter: [Next chapter](#)

Jump to: [Functions](#)

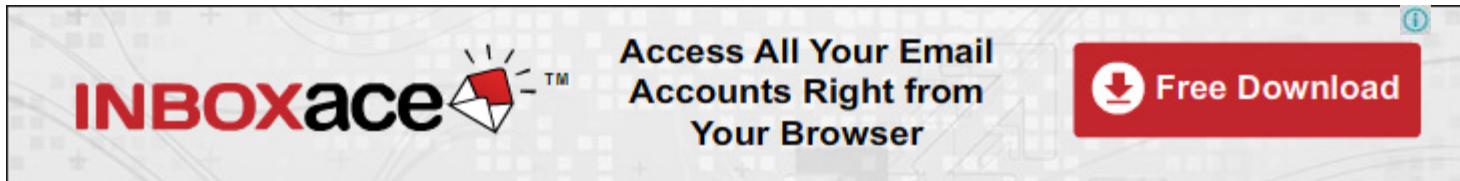
Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word "INBOXace" in red and black with a stylized envelope icon. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". On the far right is a red button with a white download icon and the text "Free Download". A small blue circular icon with a white question mark is located in the top right corner of the banner area.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Arrays

So far we've looked at the basic variables types such as strings and integers, as well as a variety of functions you can use to manipulate these data types. Beyond the basic data types are arrays and objects, which take a little bit more work in order for you to take advantage of them properly. More importantly, arrays take a solid understanding of functions before you try to use them, which is why they are separated here! As mentioned, objects are covered exclusively in their own chapter - this chapter is dedicated to the topic of arrays.

In order to model our surroundings in a programming environment accurately, it is important to recognise that some types of data naturally group together - colours, for example, naturally clump together into one group. Rather than having hundreds of separate variables - one for each colour - the reality is that it makes more sense to have one variable that holds a list, or array of colours.

Topics covered in this chapter are:

- Reading arrays
- Manipulating arrays
- Multidimensional arrays (arrays of arrays)
- Saving arrays

## Chapter contents

- 5.1. [First steps](#)
- 5.2. [Associative arrays](#)
- 5.3. [The two ways of iterating through arrays](#)
- 5.4. [The array operator](#)

[5.5. Returning arrays from functions](#)

[5.6. Array-specific functions](#)

[5.6.1. Chopping and changing arrays](#)

[5.6.2. Stripping out duplicate values](#)

[5.6.3. Filtering your array through a function](#)

[5.6.4. Converting an array to individual variables](#)

[5.6.5. Checking whether an element exists](#)

[5.6.6. Using an array as a double-ended queue](#)

[5.6.7. Swapping keys and values](#)

[5.6.8. Sorting arrays](#)

[5.6.9. Grabbing keys and values](#)

[5.6.10. Randomising your array](#)

[5.6.11. Creating an array of numbers](#)

[5.7. Multidimensional arrays](#)

[5.8. The array cursor](#)

[5.9. Holes in arrays](#)

[5.10. Arrays in strings](#)

[5.11. Saving arrays](#)

[5.12. Summary](#)

[5.13. Exercises](#)

[5.14. Further reading](#)

[5.15. Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

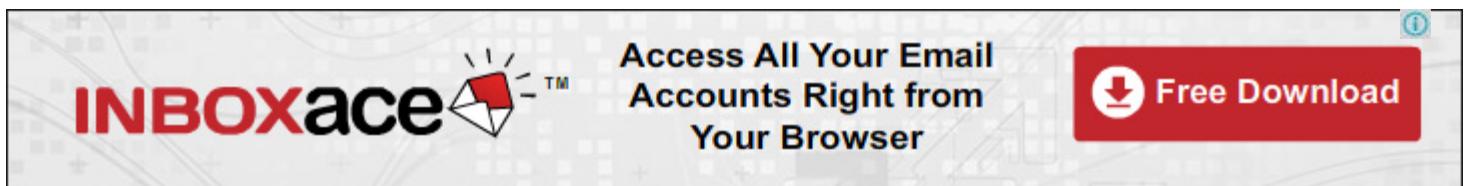
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [First steps >>](#)

Previous chapter: [Next chapter](#)

Jump to: [Arrays](#)

Home: [Table of Contents](#)

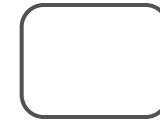


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Editable AJAX TreeGrid

The most complex tables, charts or grids on Internet. New version 12.0



# HTML Forms

PHP was originally designed for use on the Internet, and although you can now use it for command-line apps and GUIs, its main purpose remains working on the web. When it comes to the web, HTML has ruled unchallenged for some years now as the de facto standard for displaying information, even more so now that WAP usage has evaporated. This means that if you want to write a front-end for your PHP web applications, you need to understand HTML.

Topics covered in this chapter are:

- Form design using HTML
- Sending and receiving form data with PHP
- Splitting forms across pages
- Validating input

## Chapter contents

- 7.1. [The state of play](#)
- 7.2. [What does it mean to be dynamic?](#)
- 7.3. [Designing a form](#)
  - 7.3.1. [GET and POST](#)
  - 7.3.2. [Available elements](#)
  - 7.3.3. [A working form](#)
- 7.4. [Handling data](#)
  - 7.4.1. [register\\_globals](#)
  - 7.4.2. [Working around register\\_globals](#)

- [7.4.3. Magic quotes](#)
- [7.4.4. Data handling summary](#)
- [7.4.5. Handling our form](#)
- [7.5. Splitting forms across pages](#)
- [7.6. Files sent through forms](#)
- [7.7. Validating input](#)
  - [7.7.1. Client-side validation](#)
  - [7.7.2. Server-side validation](#)
  - [7.7.3. Validation in practice](#)
  - [7.7.4. Advanced variable validation using CTYPE](#)
  - [7.7.5. Key validation points](#)
- [7.8. Form design](#)
- [7.9. Summary](#)
- [7.10. Exercises](#)
- [7.11. Further reading](#)
- [7.12. Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [The state of play >>](#)

Previous chapter: [Next chapter](#)

Jump to: [HTML Forms](#)

Home: [Table of Contents](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



**INBOXace** TM

**Access All Your Email  
Accounts Right from  
Your Browser**

**Free Download**

# Files

Once you master the art of working with files, a wider world of PHP web development opens up to you. Files aren't as flexible as databases by any means, but they do offer the chance to easily and permanently store information across scripts, which makes them popular amongst programmers.

Files, as you can imagine, can store all sorts of information. However, most file formats (e.g. picture formats such as PNG and JPEG) are binary, and very difficult and/or impossible to write using normal text techniques - in these situations you should use the library designed to cope with each format.

One reminder: if you are using an operating system that uses backslash \ as the path separator (e.g. Windows), you need to escape the backslash with another backslash, making \\. Alternatively, just use /, because Windows understands that just fine too.

Topics covered in this chapter are:

- Reading and writing files
- Temporary files
- How to make a counter
- Handling file uploads
- File permissions

**Author's Note:** CPUs work in billions of operations per second - a 3GHz CPU is capable of performing three billion operations every second. RAM access time is measured in nanoseconds (billions of a second) - you can usually access some data in RAM in about 40 nanoseconds. Hard drive access time, however, is measured in milliseconds (thousandths of a second) - most hard drive have about a 7ms access time.

What this means is that hard drives are much, much slower than RAM, so working with files from your hard drive is the slowest part of your computer, excluding your CD ROM. Databases are able to store their data in RAM for much faster access time, whereas storing hard drive data in RAM is tricky.

Files are good for storing small bits of information, but it is not recommended that you use them too much for anything other than your PHP scripts themselves - counters are fine in files, as are other little things, but anything larger would almost certainly benefit from using a database. Having said that, please try to avoid the newbie mistake of putting everything into your database - if you find yourself trying to figure out what field type is right to store picture data, please have a rethink!

## Chapter contents

### 8.1. [Reading files](#)

#### 8.1.1. [readfile\(\)](#)

#### 8.1.2. [file\\_get\\_contents\(\) and file\(\)](#)

#### 8.1.3. [fopen\(\) and fread\(\)](#)

### 8.2. [Creating and changing files](#)

#### 8.2.1. [file\\_put\\_contents\(\)](#)

#### 8.2.2. [fwrite\(\)](#)

### 8.3. [Moving, copying, and deleting files](#)

#### 8.3.1. [Moving files with rename\(\)](#)

#### 8.3.2. [Copying files with copy\(\)](#)

#### 8.3.3. [Deleting files with unlink\(\)](#)

### 8.4. [Temporary files](#)

### 8.5. [Other file functions](#)

### 8.6. [Checking whether a file exists](#)

### 8.7. [Retrieving a file's status](#)

### 8.8. [Dissecting filename information](#)

### 8.9. [A working example: making a counter](#)

### 8.10. [Handling file uploads](#)

#### 8.10.1. [Advanced file upload handling](#)

#### 8.10.2. [Checking uploaded files](#)

### 8.11. [Locking files with flock\(\)](#)

### 8.12. [Permissions](#)

#### 8.12.1. [Setting permissions](#)

#### 8.12.2. [Changing file ownership](#)

### 8.13. [Working with directories](#)

- 8.13.1. [Deleting directories](#)
- 8.13.2. [One last directory function](#)
- 8.14. [Remote files](#)
- 8.15. [File checksums](#)
- 8.16. [Parsing a configuration file](#)
- 8.17. [Summary](#)
- 8.18. [Exercises](#)
- 8.19. [Further reading](#)
- 8.20. [Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Reading files >>](#)

Previous chapter: [Next chapter](#)

Jump to: [Files](#)

Home: [Table of Contents](#)



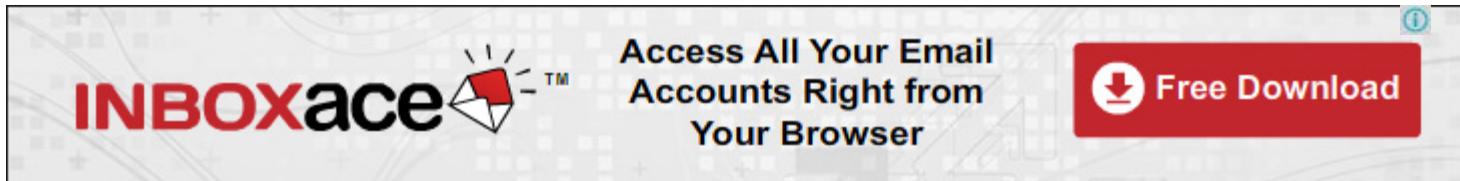
**INBOXace** ™

Access All Your Email  
Accounts Right from  
Your Browser

 [Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Databases

Many people believe database access in PHP is its most important feature, and the PHP team have indeed made it as easy as they can to interact with databases using the language. I think it is fair to say that a PHP developer who has yet to come into contact with databases really has only touched the tip of the PHP iceberg!

In this chapter we will start off with a brief description of what makes up a database, what is provided by a database manager (the program that you use to interact with the database), and a history of the most popular database manager. In the section entitled SQL you will learn how to create and store information in a database using the Structured Query Language, and also how to get it out again just as you like it. We'll also be extensively covering how to interact with your database manager using PHP, how to format your data, and much more - this is a big chapter!

In order to be able to get into serious depth on this topic, we're going to be looking specifically at the MySQL database manager - more precisely, version 5 of MySQL. This is no great loss, because MySQL 5 is the most popular open-source database in existence, so chances are you will be using it. Although we will be sticking fairly closely to MySQL, much of what we cover here will apply to other databases easily.

It is important to note that you do not need to know everything covered here - normalisation, for example, is *helpful* to know if you really want to perfect your database skills, but you can wing it without such knowledge. Similarly, you can skip over the detailed information on the PEAR database system if you have no intention to use it.

Topics covered in this chapter are:

- What makes a database
- What databases are available
- SQL commands using MySQL

- Connecting to MySQL through PHP
- Using PEAR::DB for database abstraction
- SQLite for systems without a database system
- Normalisation and table joins
- Table design considerations
- Persistent connections and transactions

## Chapter contents

### 9.1. [Introduction](#)

- 9.1.1. [Database hierarchy](#)
- 9.1.2. [Types of data](#)
- 9.1.3. [Date and time](#)
- 9.1.4. [Transactions](#)
- 9.1.5. [Stored procedures](#)
- 9.1.6. [Triggers](#)
- 9.1.7. [Views](#)
- 9.1.8. [Keys](#)
- 9.1.9. [Referential integrity](#)
- 9.1.10. [Indexes](#)
- 9.1.11. [Persistent connections](#)
- 9.1.12. [Temporary Tables](#)
- 9.1.13. [Table handlers](#)
- 9.1.14. [Round up](#)

### 9.2. [History](#)

- 9.2.1. [MySQL](#)
- 9.2.2. [PostgreSQL](#)
- 9.2.3. [Oracle](#)
- 9.2.4. [Microsoft SQL Server](#)

### 9.3. [SQL](#)

- 9.3.1. [SQL comments](#)
- 9.3.2. [Interacting with MySQL](#)
- 9.3.3. [Creating tables](#)
- 9.3.4. [Making table changes](#)
- 9.3.5. [Deleting tables](#)

- 9.3.6. [Inserting data](#)
- 9.3.7. [Selecting data](#)
- 9.3.8. [Extra SELECT keywords](#)
- 9.3.9. [Updating data](#)
- 9.3.10. [Deleting data](#)
- 9.3.11. [MySQL for dummies](#)
- 9.3.12. [A working example](#)
- 9.3.13. [Multiple WHERE conditions](#)
- 9.3.14. [Grouping rows together with GROUP BY](#)
- 9.3.15. [MySQL functions](#)
- 9.3.16. [Managing indexes](#)
- 9.3.17. [Simple text searching using LIKE](#)
- 9.3.18. [Advanced text searching using full-text indexes](#)
- 9.3.19. [Range matching](#)
- 9.3.20. [Working with NULL](#)
- 9.3.21. [Default values](#)
- 9.4. [Using MySQL with PHP](#)
  - 9.4.1. [Connecting to a MySQL database](#)
  - 9.4.2. [Querying and formatting](#)
  - 9.4.3. [Disconnecting from a MySQL database](#)
  - 9.4.4. [Reading in data](#)
  - 9.4.5. [Mixing in PHP variables](#)
  - 9.4.6. [Results within results](#)
  - 9.4.7. [Advanced formatting](#)
  - 9.4.8. [Reading auto-incrementing values](#)
  - 9.4.9. [Unbuffered queries for large data sets: MYSQLI\\_USE\\_RESULT](#)
- 9.5. [phpMyAdmin](#)
- 9.6. [PEAR::DB](#)
  - 9.6.1. [Quick PEAR::DB calls](#)
  - 9.6.2. [Query information](#)
  - 9.6.3. [Advanced PEAR::DB](#)
  - 9.6.4. [Impeared performance?](#)
- 9.7. [SQLite](#)
  - 9.7.1. [Using SQLite](#)
  - 9.7.2. [Before you begin](#)
  - 9.7.3. [Getting started with SQLite 3](#)
  - 9.7.4. [Advanced usage: SQLite3::lastInsertRowID\(\) and SQLite3::querySingle\(\)](#)

## 9.7.5. Mixing SQLite and PHP: SQLite3::createFunction()

### 9.8. Normalisation

- 9.8.1. Why separate data?
- 9.8.2. So, what is the solution here?
- 9.8.3. Why not separate data?
- 9.8.4. First normal form
- 9.8.5. Second normal form
- 9.8.6. Other normal forms
- 9.8.7. Conclusion

### 9.9. Table joins

- 9.9.1. Complex joins
- 9.10. Using temporary tables
- 9.11. Adjusting the priority queue
- 9.12. How to design your tables
- 9.13. Picking the perfect data type
- 9.14. When MySQL knows best
- 9.15. Persistent connections
- 9.16. Choosing a table type
- 9.17. Transactions
- 9.18. MySQL Improved
- 9.19. Subselects, views, and other advanced functions
  - 9.19.1. Subselects
  - 9.19.2. Views
  - 9.19.3. Referential integrity
- 9.20. Summary
- 9.21. Exercises
- 9.22. Further reading
- 9.23. Next chapter

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Introduction >>](#)

Previous chapter: [Next chapter](#)

Jump to: [Databases](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Cookies and Sessions

Owing to the fact that HTTP is stateless - that is, any data you have stored is forgotten about when the page has been sent to the client and the connection is closed - it took a little work to find a solution to the problem. Eventually, Netscape put a solution into their browser known as "cookies" - tiny bits of information that a web site could store on the client's machine that were sent back to the web site each time a new page was requested. Each cookie could only be read by the web site that had written it, meaning that it was a secure way to store information across pages.

Cookies earned a bad name at first because they allowed people to track how often a visitor came to their site, what they did on the site, and such, and many people believed that cookies signalled the end of privacy on the web. Urban myths popped up in many places saying that cookies can read any information from your hard drive, and people were encouraged to disable cookies across the board. The reality is, of course, that cookies are relatively harmless, and are now commonly accepted.

Sessions grew up from cookies as a way of storing data on the server side, because the inherent problem of storing anything sensitive on clients' machines is that they are able to tamper with it if they wish. In order to set up a unique identifier on the client, sessions still use a small cookie - this cookie simply holds a value that uniquely identifies the client to the server, and corresponds to a data file on the server.

Topics covered in this chapter are:

- How cookies and sessions compare
- Which to use and when
- How to use sessions
- Using a database to store your sessions
- Storing complex objects

## Chapter contents

- 10.1. [Cookies vs. Sessions](#)
  - 10.1.1. [Cookies](#)
  - 10.1.2. [Sessions](#)
  - 10.1.3. [Choosing the appropriate option](#)
- 10.2. [Using cookies](#)
- 10.3. [Using sessions](#)
  - 10.3.1. [Starting a session](#)
  - 10.3.2. [Adding session data](#)
  - 10.3.3. [Reading session data](#)
  - 10.3.4. [Removing session data](#)
  - 10.3.5. [Ending a session](#)
  - 10.3.6. [Checking session data](#)
  - 10.3.7. [Files vs. Databases](#)
- 10.4. [Storing complex data types](#)
- 10.5. [Summary](#)
- 10.6. [Exercises](#)
- 10.7. [Further reading](#)
- 10.8. [Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Cookies vs. Sessions >>](#)

Previous chapter: [Next chapter](#)

Jump to: Cookies and Sessions

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Multimedia

If you have previously only used PHP to work with HTML content, it will come as a nice surprise that working with multimedia content uses almost exactly the same process. It seems that many people just don't realise that PHP is capable of handling much more than non-HTML data, and hopefully this chapter will help shed a great deal of light on this exciting and challenging topic.

Through its extensions, PHP is able to easily handle creating image data in a variety of formats, generating Flash movies, and even generating PDFs - each have their own advantages and disadvantages, and are all in popular use. Having the ability to create custom multimedia content on the fly is very powerful indeed - although outputting content to HTML can do a lot, a picture says a thousand words! For example, by generating multimedia content, we can create dynamic weather forecasts, or user-customised PDFs - the possibilities are endless.

Note that in order to work with multimedia, you must first understand that each type of content, whether it be images, Flash, or PDF, all have their own unique file formats that need to be absolutely precise - you must be careful not to use any text in your multimedia PHP scripts, not even extra blank lines before or after your <?php and ?> tags. The RTF format allows text formatting to be contained using special text encoding characters, and therefore can be manipulated using simple string routines.

Topics covered in this chapter are:

- The multimedia formats that are available and their advantages
- Creating basic image formats
- Working with the rich-text format (RTF)
- Creating portable document format (PDF) files
- Working with the Shockwave Flash (SWF) format

## Chapter contents

### 11.1. Brief history of web media

11.1.1. [GIF](#)

11.1.2. [PNG](#)

11.1.3. [JPEG](#)

11.1.4. [RTF](#)

11.1.5. [PDF](#)

11.1.6. [Flash](#)

11.1.7. [SVG](#)

### 11.2. Images

11.2.1. [Creating new images](#)

11.2.2. [Choosing a format](#)

11.2.3. [Getting arty](#)

11.2.4. [More shapes](#)

11.2.5. [Complex shapes](#)

11.2.6. [Outputting text](#)

11.2.7. [Loading existing images](#)

11.2.8. [Colour and image fills](#)

11.2.9. [Adding transparency](#)

11.2.10. [Using brushes](#)

11.2.11. [Basic image copying](#)

11.2.12. [Scaling and rotating](#)

11.2.13. [Points and lines](#)

11.2.14. [Updating the drawing script](#)

11.2.15. [Special effects using imagefilter\(\)](#)

11.2.16. [Introduction to special effects using simple algorithms](#)

11.2.17. [Special FX, Colour reduction](#)

11.2.18. [Special FX, Interlacing](#)

11.2.19. [Special FX, Screen](#)

11.2.20. [Special FX, Greyscale](#)

11.2.21. [Special FX, Duotone](#)

11.2.22. [Special FX, Noise](#)

11.2.23. [Special FX, Scatter](#)

11.2.24. [Special FX, Pixelate](#)

11.2.25. [Special FX, Blur](#)

- [11.2.26. Special FX, Other special effects](#)
- [11.2.27. Interlacing an image](#)
- [11.2.28. Getting an image's MIME type](#)
- [11.2.29. Keeping your files small](#)
- [11.2.30. Making graphs](#)
- [11.3. Working with RTF](#)
- [11.4. Creating PDF documents](#)
  - [11.4.1. There's more than one way to do it](#)
  - [11.4.2. Getting started](#)
  - [11.4.3. Adding more pages and more style](#)
  - [11.4.4. Adding imagery](#)
  - [11.4.5. PDF special effects](#)
  - [11.4.6. Adding document data](#)
  - [11.4.7. PDF Conclusion](#)
  - [11.4.8. Point sizes in real life](#)
  - [11.4.9. ClipPDF interoperability](#)
  - [11.4.10. PDF without a module?](#)
- [11.5. Creating Flash](#)
  - [11.5.1. A simple movie](#)
  - [11.5.2. Flashy text](#)
  - [11.5.3. Actions](#)
  - [11.5.4. Animation](#)
  - [11.5.5. Flash Conclusion](#)
- [11.6. Summary](#)
- [11.7. Exercises](#)
- [11.8. Further reading](#)
- [11.9. Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

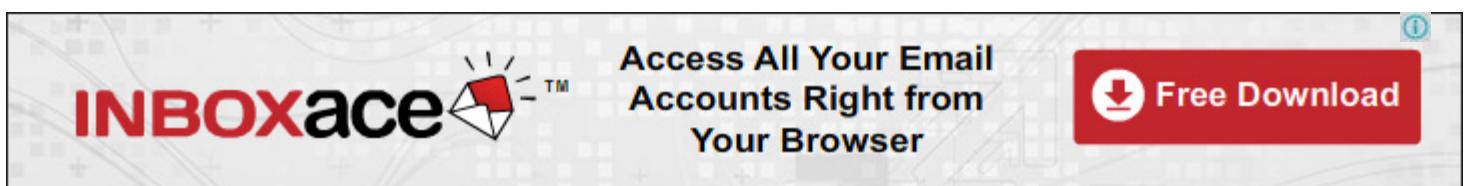
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Brief history of web media >>](#)

Previous chapter: [Next chapter](#)

Jump to: Multimedia

Home: [Table of Contents](#)



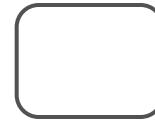
The image shows a promotional banner for INBOXace. On the left is the INBOXace logo, which consists of the word "INBOX" in red and "ace" in black, with a small envelope icon and a "TM" symbol. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". Below this text is a large red button with a white download icon and the text "Free Download". In the top right corner of the banner, there is a small blue circular icon with a white number "1".

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Get our free pH Brochure

Collection of methods, optimized to work with SevenExcellence.



# XML & XSLT

We all know that XML has great advantages going for it - it is human-readable, cross-platform, and easily converted into other forms. However, we all also know that it is not the fastest kid on the block when put up against binary formats.

So, how does a web developer harness the power of XML, whilst retaining as much performance as possible? Simple - use PHP's exceptionally fast XML module!

This chapter will teach XML parsing and manipulation using PHP, and requires that you are familiar with PHP as a language. Some familiarity with XML is required, although XML syntax and grammar is not mentioned in detail - the focus is PHP.

Important: if this is the first time you're working with XML, I strongly recommend you skip everything else and go straight to the SimpleXML module - it's the easiest way of working with XML, and it's the only thing I use nowadays.

Topics covered in this chapter are:

- Standard XML manipulation
- "SimpleXML" - the fast and easy way to use XML
- XSL and transforming XML

## Chapter contents

12.1. [Introduction to XML](#)

12.2. [Event-based parsing](#)

12.2.1. [Creating a parser](#)

- 12.2.2. [Getting to know callback functions](#)
- 12.2.3. [Callback function implementation](#)
- 12.2.4. [Event-based XML parsing, at last!](#)
- 12.2.5. [Bringing Everything Together](#)
- 12.3. [SimpleXML](#)
  - 12.3.1. [First steps](#)
  - 12.3.2. [Reading from a string](#)
  - 12.3.3. [Searching and filtering with XPath](#)
  - 12.3.4. [Outputting XML](#)
- 12.4. [Transforming XML using XSLT](#)
  - 12.4.1. [Adding PHP to the mix](#)
  - 12.4.2. [Handling the processed output](#)
  - 12.4.3. [Making XSL work for its money](#)
  - 12.4.4. [What else can XSL do?](#)
  - 12.4.5. [XSLT Conclusion](#)
- 12.5. [Summary](#)
- 12.6. [Exercises](#)
- 12.7. [Further reading](#)
- 12.8. [Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Introduction to XML >>](#)

Previous chapter: [Next chapter](#)

Jump to: [XML & XSLT](#)

Home: [Table of Contents](#)



**INBOXace**™

Access All Your Email  
Accounts Right from  
Your Browser

[Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



**INBOXace**™

Access All Your Email  
Accounts Right from  
Your Browser

 Free Download

# Output Buffering

Without output buffering, PHP sends data to your web server as soon as it is ready - this might be line by line or code block by code block. Not only is this slow because of the need to send lots of little bits of data, but it also means you are restricted in the order you can send data. Output buffering cures these ills by enabling you to store up your output and send to send it when you are ready to - or to not send it at all, if you so decide.

Topics covered in this chapter are:

- When to use output buffering
- Manipulating multiple buffers
- Incremental data flushing
- Output compression

## Chapter contents

- 13.1. [Advantages](#)
- 13.2. [Performance Considerations](#)
- 13.3. [Getting started](#)
- 13.4. [Reusing buffers](#)
- 13.5. [Stacking buffers](#)
- 13.6. [Flushing stacked buffers](#)
- 13.7. [Reading buffers](#)
- 13.8. [Other OB functions](#)
- 13.9. [Flushing output](#)

[13.10. Compressing output](#)

[13.11. URL rewriting](#)

[13.12. Summary](#)

[13.13. Exercises](#)

[13.14. Further reading](#)

[13.15. Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

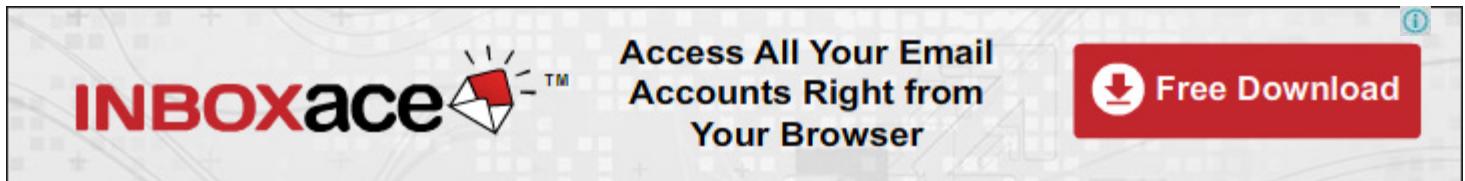
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Advantages >>](#)

Previous chapter: [Next chapter](#)

Jump to: [Output Buffering](#)

Home: [Table of Contents](#)



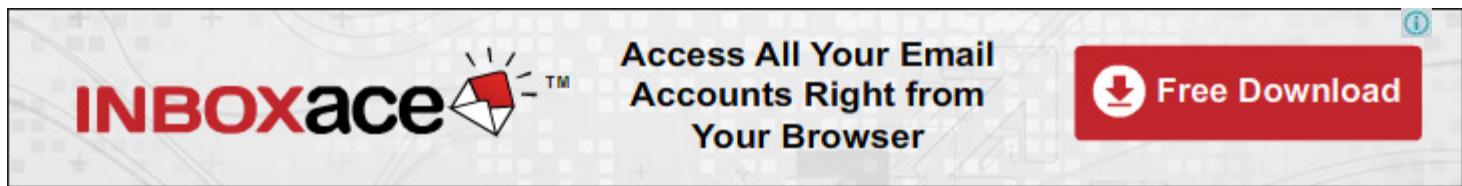
The banner features the INBOXace logo on the left, which includes the word "INBOXace" in a bold, black, sans-serif font next to a stylized red envelope icon with a "TM" symbol. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". On the far right is a red button with a white download icon and the text "Free Download". A small blue circular icon with a white question mark is located in the top right corner of the banner area.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).



- Writing PHP

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Writing PHP

While it is almost certain that you read this book to learn the PHP programming language, I believe that there is a lot more to learning a language than just memorising its syntax and functions. For example, learning to program to a fixed style means your code will be easier to maintain by yourself and by others, and learning to debug properly means your code will have fewer bugs and you will have fewer headaches.

This chapter is designed to fill you in on the practical aspects of being a PHP programmer, and should hopefully make your programming life easier, as well as help you find and solve problems with the minimum of fuss.

The topics covered in this chapter are:

- How to analyse your system requirements
- Using a development tool to help you code
- File layout schemes and group development
- Documentation and testing

- Distribution your code and licensing your work
- How to debug your scripts
- Troubleshooting hints and tips
- Where to get help if you still have a problem

## Chapter contents

### 19.1. The design process

- 19.1.1. [Analysing the requirements](#)
- 19.1.2. [Designing the solution](#)
- 19.1.3. [Developing the code](#)
- 19.1.4. [Implementing the application](#)
- 19.1.5. [Maintenance and support](#)

### 19.2. Which IDEs are good

- 19.2.1. [Line numbering](#)
- 19.2.2. [Syntax highlighting](#)
- 19.2.3. [Online help](#)
- 19.2.4. [Code insight](#)
- 19.2.5. [Interactive debugging](#)
- 19.2.6. [Profiling](#)
- 19.2.7. [Popular IDEs](#)

### 19.3. Laying out your files

- 19.3.1. [Directory structuring](#)

### 19.4. Group development

- 19.4.1. [How to develop code](#)
- 19.4.2. [Version control](#)

### 19.5. Documenting your project

### 19.6. Testing

### 19.7. Distributing your code

- 19.7.1. [Charging for your work](#)
- 19.7.2. [PHP Encoders](#)
- 19.7.3. [Cross-platform code 1: Loading extensions](#)
- 19.7.4. [Cross-platform code 2: Using extensions](#)
- 19.7.5. [Cross-platform code 3: Path and line separators](#)
- 19.7.6. [Cross-platform code 4: Coping with php.ini differences](#)

19.7.7. [Cross-platform code 5](#)

19.8. [Debugging](#)

19.8.1. [What is a bug?](#)

19.8.2. [The most basic debugging technique](#)

19.8.3. [Making assertions](#)

19.8.4. [Triggering your own errors](#)

19.8.5. [Source highlighting](#)

19.8.6. [Handling MySQL errors](#)

19.8.7. [Exception handling](#)

19.8.8. [Backtracing your code](#)

19.8.9. [Debuggers](#)

19.8.10. [Custom error handlers](#)

19.8.11. [Custom exception handlers](#)

19.8.12. [Using @ to disable errors](#)

19.8.13. [phpinfo\(\)](#)

19.9. [Debugging practice](#)

19.10. [Coding style](#)

19.10.1. [Comments and whitespace](#)

19.10.2. [Variable naming](#)

19.10.3. [Functions](#)

19.10.4. [Distinguishing code blocks](#)

19.11. [Output style](#)

19.11.1. [Options for Tidy](#)

19.12. [Troubleshooting](#)

19.12.1. [Error types](#)

19.12.2. [Choosing what types of errors you see](#)

19.12.3. [Common errors](#)

19.13. [Getting Help](#)

19.13.1. [The documentation](#)

19.13.2. [Mailing lists](#)

19.13.3. [Websites](#)

19.13.4. [IRC](#)

19.13.5. [Conferences](#)

19.13.6. [User groups](#)

19.13.7. [Submitting a bug](#)

19.13.8. [Contacting the author](#)

19.14. [Getting qualified](#)

[19.15. Summary](#)

[19.16. Exercises](#)

[19.17. Further reading](#)

[19.18. Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

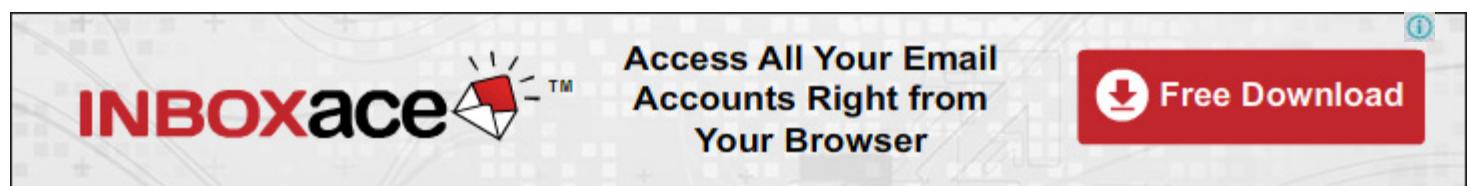
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [The design process >>](#)

Previous chapter: [Next chapter](#)

Jump to: [Writing PHP](#)

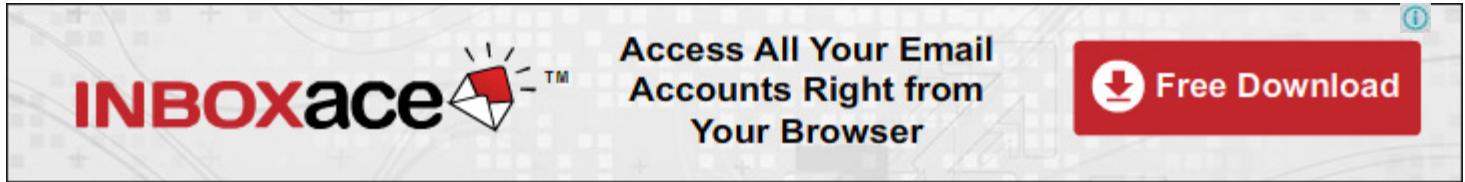
Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word "INBOX" in red and "ace" in black, with a small envelope icon and "TM" symbol. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". Below this text is a large red button with a white download icon and the text "Free Download". In the top right corner of the button is a small blue information icon.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Writing extensions

PHP is an incredibly rich language, as I hope you have come to see by reading this far - there are extensions available to perform all sorts of popular operations with external libraries that really help add power to your script without much effort.

So, why would anyone want to write their own extension? We'll be answering that question in this chapter, as well as designing and producing from scratch a new extension for PHP. Note that this entire chapter requires you have a compiler to hand so that you can compile your extension. You need to be able to compile PHP - usually this requires a Unix operating system, but if you have Windows and a Windows compiler, or Cygwin, that will do also. Otherwise, while the information here might be interesting, you will not be able to make use of it unless you can compile PHP.

Topics covered in this chapter:

- When to write a custom extension
- How to design, create, and test your extension

## Chapter contents

### 20.1. Why write your own extension?

#### 20.1.1. The C Perspective

### 20.2. Before we begin

### 20.3. First steps

### 20.4. Hello world - in C!

### 20.5. C Hello World v2

### 20.6. Factorials in C

### 20.7. Extensions Conclusion

[20.8. Summary](#)

[20.9. Further reading](#)

[20.10. Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

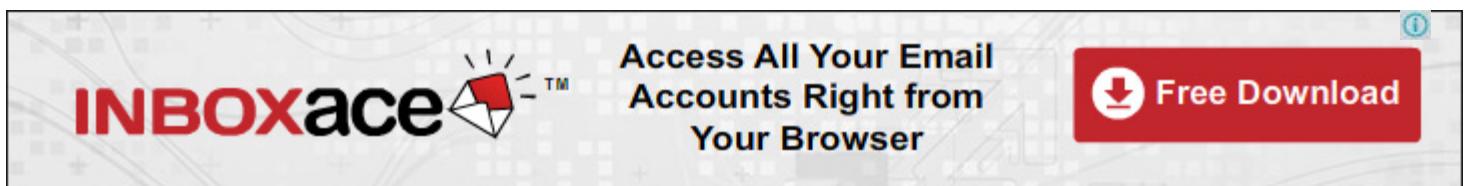
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Why write your own extension? >>](#)

Previous chapter: [Next chapter](#)

Jump to: [Writing extensions](#)

Home: [Table of Contents](#)



The image shows an advertisement for INBOXace. On the left is the INBOXace logo, which consists of the word "INBOX" in red and "ace" in black, with a small envelope icon and a trademark symbol. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". Below this text is a red button with a white download icon and the text "Free Download". In the top right corner of the ad, there is a small blue circle with a white number "1".

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Alternative PHP uses

As you probably know, PHP is a recursive acronym for "PHP: Hypertext Preprocessor", so why should a product designed for the web ever stray from its original goal? Quite simple, because PHP is cool, and if I could use PHP to do *everything*, I'd certainly give it a try, if only for the challenge!

In this chapter we're going to be looking at other uses you can put PHP to in order to extend its usefulness. One key thing to remember is that you can use what you have learnt so far wherever else you use PHP, but you will find as you read on that there is a lot of specific information to absorb about individual uses.

Hopefully this chapter will inspire you to try to transfer your PHP skills to other areas - it is a big world out there!

The topics covered in this chapter are:

- How to use PHP to write shell scripts
- How the CLI SAPI differs from "normal" PHP
- Interacting with the dialog program to create command-line user interfaces
- Using GTK+ to create graphical user interfaces
- Using Glade to automatically generate GTK+ GUIs
- Making text-based games with PHP
- Making graphical games with PHP and SDL
- Creating your own miniature language with PHP

## Chapter contents

21.1. [What else can be done with PHP?](#)

21.2. [Command-line scripting](#)

21.2.1. [Why use shell scripts?](#)

21.2.2. [CLI SAPI differences](#)

21.2.3. [Your first CLI script](#)

21.2.4. [Advanced command-line parsing](#)

21.2.5. [Getting down and dirty](#)

21.2.6. [Getting into the swing of things](#)

21.2.7. [Sending code direct to PHP](#)

21.2.8. [The grand finale](#)

21.2.9. [CLI Conclusion](#)

21.3. [Graphical user interfaces](#)

21.3.1. [Getting started](#)

21.3.2. [GUI toolkits](#)

21.3.3. [A Basic GUI](#)

21.3.4. [Multiple Windows](#)

21.3.5. [Handling popup menus](#)

21.3.6. [Advanced GUIs](#)

21.3.7. [Using Custom Parameters](#)

21.3.8. [GUI Themes](#)

21.3.9. [Distributing your apps](#)

21.3.10. [Graphical Interfaces Conclusion](#)

21.4. [Making games](#)

21.4.1. [Text-based world planning](#)

21.4.2. [Text game v1](#)

21.4.3. [Getting graphical](#)

21.4.4. [Getting it to work](#)

21.4.5. [First steps](#)

21.4.6. [Moving our sprite](#)

21.4.7. [Clearing the screen](#)

21.4.8. [Last tweaks](#)

21.4.9. [Games conclusion](#)

21.5. [Making your own language](#)

21.5.1. [Why make your own language?](#)

21.5.2. [The elements of a compiler](#)

21.5.3. [Analysis](#)

21.5.4. [Output](#)

- 21.5.5. [Planning it out](#)
- 21.5.6. [How to parse text into tokens](#)
- 21.5.7. [What is a token?](#)
- 21.5.8. [How parsing works](#)
- 21.5.9. [Finally, execution](#)
- 21.5.10. [If you have made it this far...](#)
- 21.5.11. [Operator precedence](#)
- 21.5.12. [The script in \(almost\) full](#)
- 21.5.13. [Mini-language conclusion](#)
- 21.6. [Summary](#)
- 21.7. [Further reading](#)
- 21.8. [Next chapter](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

[Tweet](#)

Next chapter: [What else can be done with PHP? >>](#)

Previous chapter: [Next chapter](#)

Jump to: [Alternative PHP uses](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

The BEST way to search  
product manuals online.  
**AtoZ Manuals**



**Click Here!**

# Practical PHP

While this book so far has by no means been theory only, we have not really looked at examples of complete PHP projects - applications taken from through all the stages of software development. There are a number of common PHP applications that you are likely to want to make during your programming career, and the code and ideas presented here is designed to be easily ported into your own work, even if your actual project is not covered exactly here.

The projects covered here are a poll, a guestbook, a messageboard, and image thumbnails - these terms are explained more in each individual section. Although there is not enough room in this book to cover much design brainstorming and the like, I have tried to give as much technical detail as possible in the hope that it will deepen your understanding of the language.

## Chapter contents

### 22.1. Creating a poll

- 22.1.1. Analysis: what makes a web poll?
- 22.1.2. Development: creating the simplest poll
- 22.1.3. Analysis: Poll v2
- 22.1.4. Putting Poll v2 into action
- 22.1.5. Analysis: Poll v3
- 22.1.6. Making the final poll
- 22.1.7. Building a better poll

### 22.2. Creating a guestbook

- 22.2.1. Analysis
- 22.2.2. Development
- 22.2.3. Problems in paradise: Guestbook v2

- 22.2.4. [Fixing the problems](#)
- 22.2.5. [Building a better guestbook](#)
- 22.3. [Creating a messageboard](#)
  - 22.3.1. [Analysis](#)
  - 22.3.2. [Development](#)
  - 22.3.3. [Analysis: Messageboard v1.1](#)
  - 22.3.4. [Making Messageboard v1.1](#)
  - 22.3.5. [Messageboard v2](#)
  - 22.3.6. [Making an object-oriented messageboard](#)
- 22.4. [Creating thumbnails](#)
  - 22.4.1. [Analysis](#)
  - 22.4.2. [Development](#)
- 22.5. [ASCII art](#)
  - 22.5.1. [Analysis](#)
  - 22.5.2. [Development](#)
  - 22.5.3. [Analysis: ASCII art in colour](#)
- 22.6. [Further Reading](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

[Next chapter: Creating a poll >>](#)

[Previous chapter: Next chapter](#)

Jump to: [Practical PHP](#)

Home: [Table of Contents](#)

# Owners Instruction Manual

Find PDF Product Manuals By Brands & Categories. Use AtoZManuals Now!



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

Hacking with PHP



# Object type information

```
bool is_subclass_of ( mixed object, string class_name)
```

As you can see, inheriting from class to class is an incredibly powerful way to build up functionality in your scripts. However, very often it is easy to get lost with your inheritance - how can you tell what class a given object is?

PHP comes to the rescue with a special keyword, "instanceof", which can be used like an operator. Instance of will return true if the object on the left-hand side is of the same class or a descendant of the class given on the right-hand side. For example, given the code \$poppy = new poodle::

```
if ($poppy instanceof poodle) { }
if ($poppy instanceof dog) { }
```

Both of those if statements would evaluate to be true, because `$poppy` is an object of the poodle class, and also is a descendant of the dog class.

**Author's Note:** Java programmers will be happy to know that instanceof is the same old friend they've grown used to over the years. It is a great keyword to keep to hand, as any Java veteran will tell you, and you will almost certainly find yourself using it quite often in your scripts.

If you only want to know whether an object is a descendant of a class, and not of that class itself, you can use the `is_subclass_of()` function. This takes an object as its first parameter (or a string

containing a class name), a class name string as its second parameter, and returns either true or false depending on whether the first parameter is descended from the class specified in the second parameter.

Understanding the difference between instanceof and *is\_subclass\_of()* is crucial - this script should make it clear:

```
<?php
    class dog { }
    class poodle extends dog { }
    $poppy = new poodle();
    print (int)($poppy instanceof poodle);
    print "\n";
    print (int)is_subclass_of($poppy, "poodle");
?>
```

That should output a 1 then a 0. Typecasting to int is used because boolean false is printed out as "" (blank), but by typecasting to an integer this becomes 0. As you can see, using instanceof reports true that **\$poppy** is either a poodle or a dog, whereas *is\_subclass\_of()* reports false because **\$poppy** is not descended from the class "poodle" - it *is* a poodle.

Author's Note: You can also use the instanceof keyword to see whether an object implements an interface.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Class type hints >>](#)

Previous chapter: [Iterating through object variables](#)

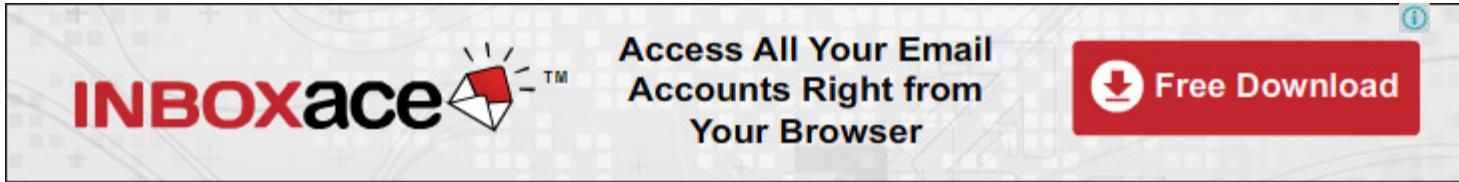
Jump to: [Object type information](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Saving objects

```
array get_object_vars ( object input)
```

Previously we covered how to save arrays in PHP using `serialize()`, `unserialize()`, `urlencode()`, and `urldecode()`. Saving objects works precisely the same way - you `serialize()` them into a string to make a format that can be saved, then `urlencode()` them to get a format that can be passed across the web without problem.

For example:

```
$poppy = new poodle('Poppy');
$safepoppy = urlencode(serialize($poppy));
```

There is one special feature with saving objects, though, and that is the fact that when `serialize()` and `unserialize()` are called, they will look for a `__sleep()` and `__wakeup()` function on the object they are working with respectively. These functions, which you have to provide yourself if you want them to do anything, allow you to properly keep an object working during its hibernation period (when it is just a string of data).

For example, when `__sleep()` is called, a logging object should save and close the file it was writing to, and when `__wakeup()` is called the object should reopen the file and carry on writing. Although `__wakeup()` need not return any value, `__sleep()` must return an array of the values you wish to have saved. If no `__sleep()` function is present, PHP will automatically save all variables, but you can mimic this behaviour in code by using the `get_object_vars()` function - more on that soon.

In code, our logger example would look like this:

```
class logger {
    private function __sleep()
```

```

    $this->saveAndExit();
    // return an empty array
    return array();
}

private function __wakeup() {
    $this->openAndStart();
}

private function saveAndExit() {
    // ...[snip]...
}

```

Any objects of this class that are serialized would have `__sleep()` called on them, which would in turn call `saveAndExit()` - a mythical clean up function that saves the file and such. When objects of this class are unserialized they would have their `__wakeup()` function called, which would in turn call `openAndStart()`.

To have PHP save all variables inside a `__sleep()` function, you need to use the `get_object_vars()` function. This takes an object as its only parameter, and returns an array of all the variables and their values in the object. You need to pass the variables to save back as the values in the array, so you should use the `array_keys()` function on the return value of `get_object_vars()`, like this:

```

private function __sleep() {
    // do stuff here
    return array_keys(get_object_vars($this));
}

```

If there is nothing in your `__sleep()` function other than returning the array, you should delete the function altogether as it is no different to what PHP will do by default.

If you find yourself needing to save objects, keep `__sleep()` and `__wakeup()` in mind - together they allow you to keep objects fully working across pages.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

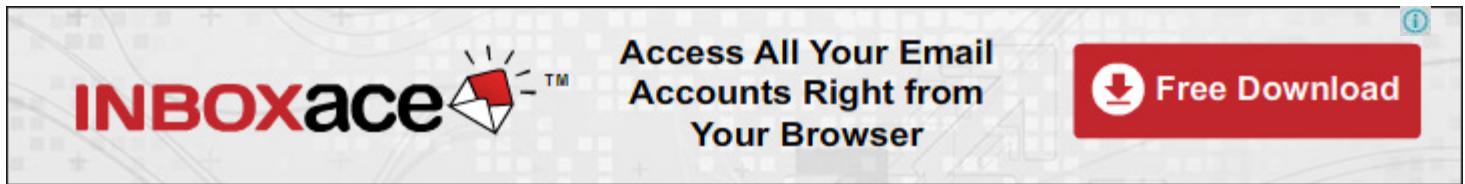
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Magic functions >>](#)

Previous chapter: [Comparing objects with == and ===](#)

Jump to: [Saving objects](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

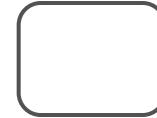
Hacking with PHP

Is this book for you?

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Free E-mail Organizer

Organize all Your E-Mail Accounts. InboxAce - Fast & Free! Get App.



# Is this book for you?

This book assumes no PHP programming skill at all - you'll be taught from scratch in that respect. However, I will be using terms like "variable", "function", and "loop" freely, so any prior programming experience you have will help enormously.

If you're coming from the world of HTML, PHP is a great step forward to make. HTML by itself, as you may have discovered, is a very simple language, which as a result leaves you with very simple pages. Think of using HTML like owning a car with no engine - the car might look great, but it will never actually do very much. PHP can take your sites into the fast lane by allowing you to turn your static pages into exciting, ever-changing pages with only a little thinking. In essence, it's like putting a V6 engine into your car shell.

Programmers coming from the world of Perl, you should prepare yourself for a surprise: Perl isn't the only good language out there! PHP is much easier on the eye than Perl, and often faster both in terms of development time and execution time. However, you have to make one big sacrifice: the major difference between PHP and Perl is that PHP has much less "one-line magic" than Perl. Regular contributors to the Obfuscated Perl Contest may find that PHP is probably not for them!

Those of you already programming in PHP and looking to extend your knowledge, then this book is ideal - you will find information on how to use the more advanced features (IMAP, XML, and Sockets), tips and tricks on how to program better PHP, and a great reference too.

If you're not a HTML programmer and not a Perl programmer, then relax - all the HTML and CGI information you need to know to use PHP is covered here.

*Please note: this book has been written to be OS-agnostic.* That is, samples are included for both Windows and Unix-like systems where possible. However, due to operating system constraints some parts that work on Windows may not work on Unix OSs, and vice versa. These sections are marked individually, along with reasoning as to why the information is not cross-platform.

Finally, if you're expecting that it is easy to learn PHP; that you can learn a new language without having to memorise hundreds of functions; that one book can take you from beginner to expert in web development - you're absolutely right, because you're reading it!

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Who this book is not for >>](#)

Previous chapter: [Preface](#)

Jump to: [Is this book for you?](#)

Home: [Table of Contents](#)

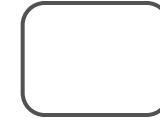
Hacking with PHP

Who this book is not for

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Repair Manual Online

Free Online Instruction Manuals For Popular Brands. Use AtoZManuals Now



# Who this book is not for

If you don't fit into one of these categories, you're doing well already:

- People who want more pictures than words
- People who don't trust open-source software
- People who aren't willing to commit a little time each day or every few days to try out code examples for themselves

I cite the last category particular because of Flip Wilson's law: "You can't expect to hit the jackpot if you don't put a few nickels in the machine".

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

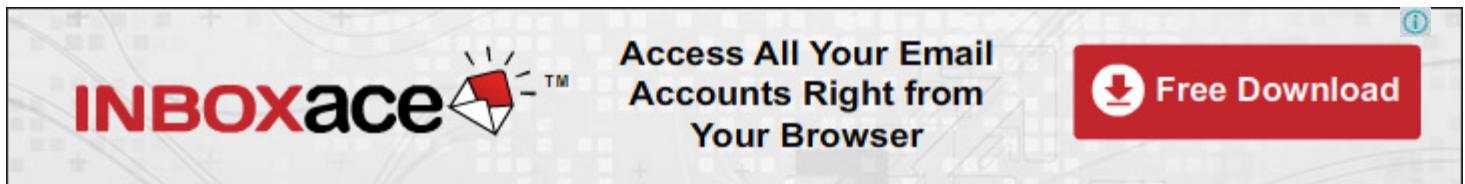
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [What you will get out of this book >>](#)

Previous chapter: [Is this book for you?](#)

Jump to: Who this book is not for

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# What you will get out of this book

Irrespective of what programming level you have before you start reading this book, there are a number of key things you should be able to take away once you are done:

- You'll have a thorough mastery of the ins and outs of PHP programming and development-related tasks, with particularly strong knowledge of forms, databases, and multimedia.
- You'll be able to design, develop, and deploy complex web-based solutions across several platforms
- You'll have a good understanding of the uses PHP can be put to above and beyond handling forms
- Finally, I really hope you will share some of the passion I feel for PHP - it's an exciting, challenging, and ever-expanding language and it's my wish that you'll have lots of fun working with it.

That said, I'm under no disillusionments that this is the "ultimate" book on PHP or programming in general. I've provided reading recommendations at the end of each chapter where I feel other books will contribute to your learning.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

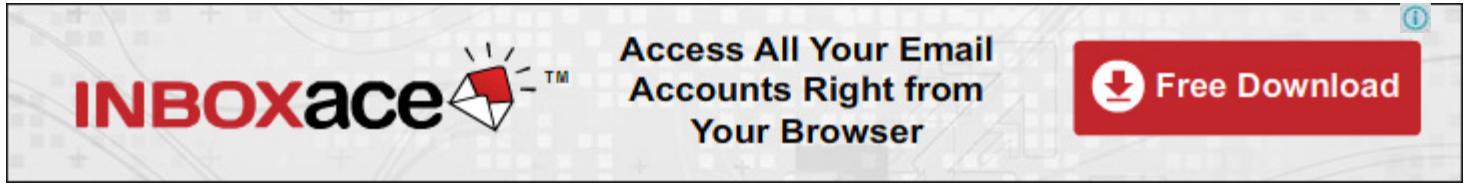
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [How to use this book >>](#)

Previous chapter: [Who this book is not for](#)

Jump to: [What you will get out of this book](#)

Home: [Table of Contents](#)

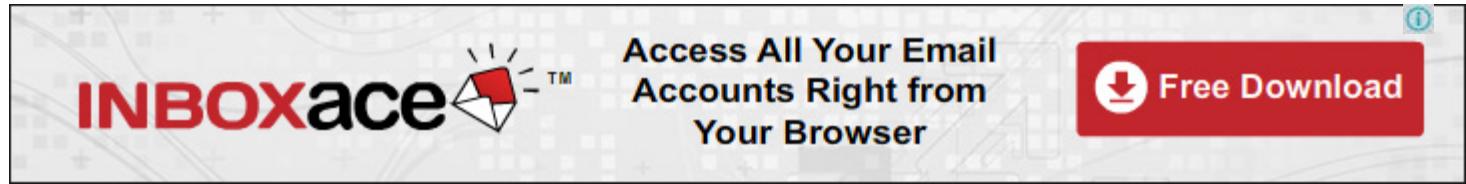


The image shows a promotional banner for INBOXace. On the left is the INBOXace logo, which consists of the word "INBOXace" in a bold, black, sans-serif font next to a red envelope icon with yellow sunburst-like lines emanating from it. A small "TM" symbol is to the right of the envelope. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". To the right of that is a red button with a white download icon and the text "Free Download". In the top right corner of the banner is a small blue circular icon with a white question mark inside.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

- Cross-platform PHP

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Cross-platform PHP

PHP works on many platforms, including Windows, Linux, BSD, Solaris, and OS X to name just a few - this book attempts to cover PHP from an OS-agnostic point of view, which means I have tried hard to make sure you can understand and follow as much code as possible irrespective of what platform you use.

To make things easier to read, I have lumped Linux, BSD, Solaris, OS X, and other Unix-like OSs into "Unix" - this saves me typing and saves you reading, so everyone is a winner.

Throughout the book I refer to "shell prompts" and "command lines". This is the text entry system for your OS - for Windows users this can be found by clicking Start, then Run, and entering "cmd". Linux/BSD users running from the console are already there, but if you are using a GUI such as KDE or GNOME, you will need to launch xterm, Konsole, Gnome Terminal or whatever you use to enter text commands. Mac OS X users should have "Terminal" available for launching. I encourage use of the command line because it lets you use PHP in "interactive mode" - to type things, and see them happen immediately.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

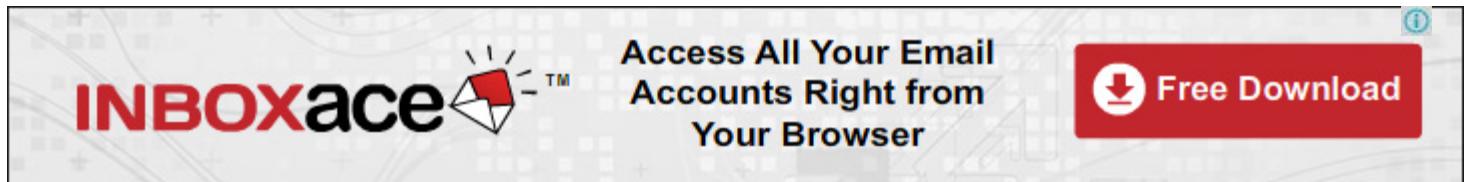
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Tips for success >>](#)

Previous chapter: [A note for programmers coming from C, C++, or Java](#)

Jump to: [Cross-platform PHP](#)

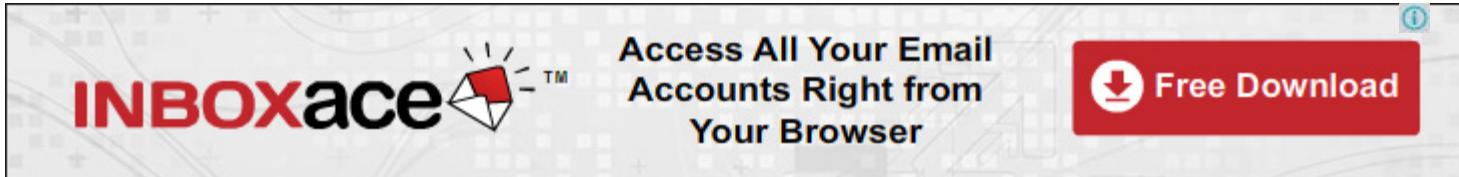
Home: [Table of Contents](#)



The image shows a promotional banner for INBOXace. On the left is the INBOXace logo, which consists of the word "INBOX" in red and "ace" in black, with a small graphic of an envelope and a gear-like symbol to the right. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". To the right of that is a red button with a white download icon and the text "Free Download". In the top right corner of the banner is a small blue circular icon with a white question mark.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Tips for success

Despite my best efforts to cut out as much fluff as possible, this is still quite a long book, and programming is largely a dry topic. Having said that, learning PHP need not be a case of reading a chapter a day and memorising things as you go, and there are a number of tips I want to share with you before you start reading that will help you get the most out of this book:

- Don't try to memorise things en masse. I will make it quite clear if there is something that is very important for you to remember, but to be frank that will be rare - trying to remember hundreds of functions or parameters or bits of jargon will only slow your learning.
- Don't rush yourself. Yes, it is fun to learn, and you are going to have a great time learning PHP I can promise you, but that does not mean you should try to hit the ground running. Take it easy, and you will remember things better, enjoy the book more, and become a better programmer as a result.
- Familiarise yourself with the language. Once you know a few bits and pieces, get started using them straight away. There is a surprising amount of functionality you can put together knowing just a few simple commands, and that's *before* you try hooking them all together into bigger and better things. So, get started using PHP as early as you feel comfortable.
- Be creative. If you want to try out something crazy, go for it - the only way you are going to learn is by trying, and, yes, failing too.
- Program in an environment that suits you. This might be your bedroom, in the dark, in the early hours of the morning, or it might be on a laptop in a park somewhere. The great thing about programming is that there are no restraints on how you do it - think of it like freestyle art.
- Don't over-do it. If you are a caffeine hog like most geeks, sure, go ahead and drink coffee all you want - I get my fix from Mountain Dew, which has kept me going on many all-nighters. Having said

that, you aren't helping yourself if you fall asleep reading this book night after night, and neither is it a good thing to program when you are half awake - there is nothing worse than waking up at a desk the next morning and not having the faintest clue how the code you wrote last night works.

- Finally, remember that programming is supposed to be *fun* . If you find yourself hitting a mental brick wall repeatedly, you are doing something wrong. This book is structured so that you keep getting newer and harder challenges to push your programming talents further, but feel free to stop at any point and take a walk in the park.

Enjoy the book!

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

[Tweet](#)

Next chapter: [About the Publisher >>](#)

Previous chapter: [Cross-platform PHP](#)

Jump to: [Tips for success](#)

Home: [Table of Contents](#)

**INBOXace** ™

Access All Your Email  
Accounts Right from  
Your Browser

 [Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

About the Publisher

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# About the Publisher

I wrote Hacking with PHP, and I'm publishing it myself here on this site, completely free, so I guess there's no traditional mega-corp book publisher behind me. I hope that the relatively clean and simple design of the site is able to offset the annoyance of having to carry adverts to pay for hosting costs!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Copyright and usage rights >>](#)

Previous chapter: [Tips for success](#)

Jump to: [About the Publisher](#)

Home: [Table of Contents](#)



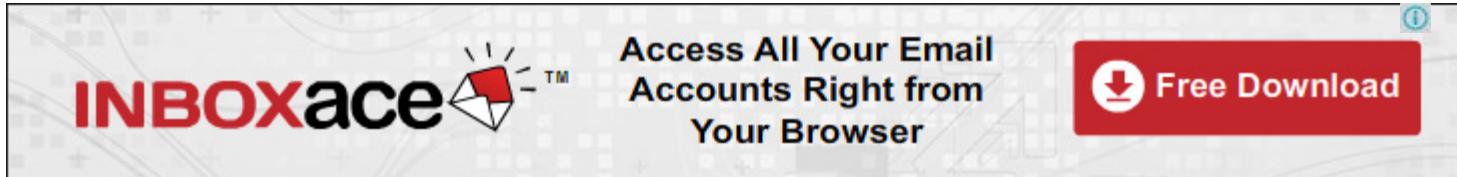
**INBOXace**™

Access All Your Email  
Accounts Right from  
Your Browser

 Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Copyright and usage rights

By using any part of the content you do so at your own risk - it is all offered with no warranty as to its suitability for any purpose, and may cause damage to your systems.

This book is copyright (c) 2003-2015 Paul Hudson. There may be trademarks and other copyrights in the content that are owned by others. You're welcome to take any PHP source code and use it in your own projects as if you wrote it yourself - feel free to use as much (or as little!) of it as you want.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

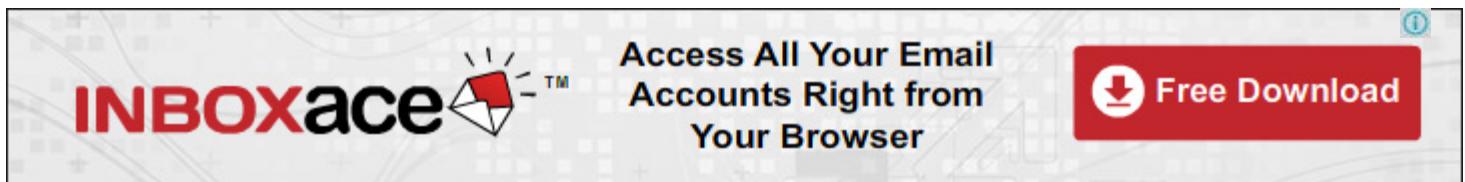
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Give something back! >>](#)

Previous chapter: [About the Publisher](#)

Jump to: Copyright and usage rights

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

Give something back!

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Give something back!

If you'd like to help me, the best thing you can do is help spread the word. Hacking with PHP (this site!) and [Hacking with Swift](#) both took me a massive amount of time to write, and the wider the audience they reach the more it encourages me to produce more content like it. So, please tell your friends and family! Post on social media! It all helps, and it's all very welcome.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

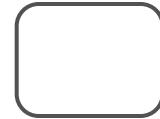
Next chapter: [About me >>](#)

Previous chapter: [Copyright and usage rights](#)

Jump to: Give something back!

# Editable AJAX TreeGrid

The fastest and most complex table, grid, chart or treegrid everywhere!

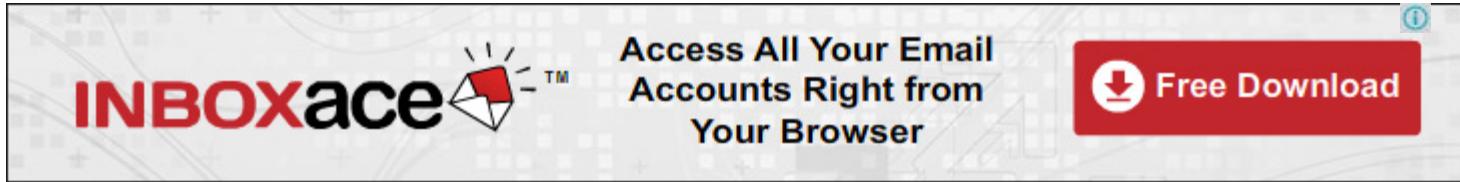


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

About me

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# About me

I'm a full-time developer working for Future Publishing in Bath, England. I love the free software movement, use Linux on all my servers, and enjoy programming anything that stays still long enough.

[You should follow me on Twitter.](#)

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Acknowledgements >>](#)

Previous chapter: [Give something back!](#)

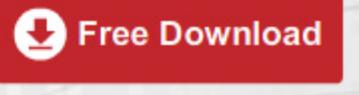
Jump to: About me

Home: [Table of Contents](#)



**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

 Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# Acknowledgements

This work would not have been possible without the love and support of my wife, Ildiko, and the skills and ideas granted to me by God. I supposed I'd better mention Nick Veitch, who taught me that journalism is as much of a game as we make it, as well as happily subjecting himself to regular thrashings at Crack Attack.

On that note, Daniel Nelson (the creator of Crack Attack) has many crimes to answer for: if only he knew how many hours I've lost playing that darn game...!

Furthermore, I owe my thanks to the kind readers who wrote in to correct errors in the content - without them you'd be reading some pretty clumsy and inept text! So, in alphabetical order, my thanks go out to: Rand Acs, Tim Ahpee, Jeremy Amos, Oliver Clarke, Michael Corcoran, John F. Douthat, J. Thomas Enders, Jack Gates, Berge Gazen, Nick Hall, Smith Hayward, Sven Hermans, Michael Hkam, Ammar Ibrahim, Shad Itschner, Erik Kneyber, Rob Lemieux, Elizabeth Leung, Neil McClean, Terry Peppers, PrzemysÅ,aw Pacholski, Timothy Pinkham, Gao Qing, Ronald Rhodes, Eric Sabo, Ulrich J Seyfferdt, Braxton Sherouse, Andrew Taylor, Artemy Tregoubenko, Russell Wiley, Nathan Witmer, John Wright, and Miloslav Zridkavesely for their valuable input.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Introducing PHP >>](#)

Previous chapter: [About me](#)

Jump to: [Acknowledgements](#)

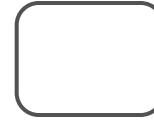
Home: [Table of Contents](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## A world without privacy

US debates a world without privacy, arguments of law agencies



# Advantages of PHP

I could cop out here and say that the advantages of PHP are too many to list in such a small book as this, but it seems that some people need convincing of PHP's inherent greatness, and I am very happy to oblige.

If you are not new to PHP, I still recommend you at least glance over the contents of this chapter. I find that programming in PHP is very often like playing my guitar. I can be sitting in my study strumming idly on my 12-string like I've done thousands of times before, only this time I play a different chord sequence, or strum the strings in a different manner than normal, and I think "Hey, that's cool... how come I never found that before?"

At various points in my PHP programming past, I have stumbled across functions or functionality in pretty much the same way, and responded with pretty much the same response! If you're the same, then definitely read on - it doesn't hurt to see what's on offer.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

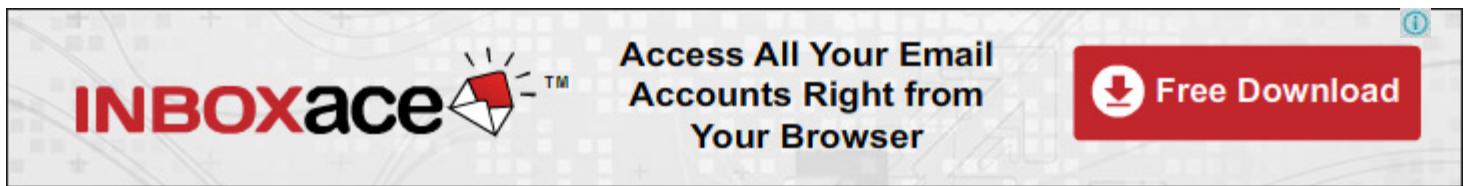
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [The HTML relationship >>](#)

Previous chapter: [The Zend Relationship](#)

Jump to: [Advantages of PHP](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# History

The key to understanding something fully is to know how it got to where it is currently, and PHP is no different in that respect - it has a long history full of key design decisions and changes to the language. The history of web development is even longer, and this section will cover both, albeit briefly.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Background >>](#)

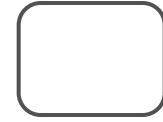
Previous chapter: [Introducing PHP](#)

Jump to: History

Home: [Table of Contents](#)

# Get our free BOD Brochure

How to setup your own Biochemical Oxygen Demand determination process

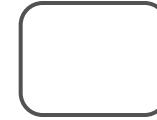


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Hyper-V Backup Software

Fast, easy, secure. Start from \$198 Backup unlimited VMs. Encrypted.



# Background

In older times, code to parse form input was usually written in C. While it executed very quickly, it was also tricky to program - a simple parsing script could easily take up fifty lines or more. The main reason for the length was that C was not designed specifically for the web so there was no pre-written code to make common tasks easier, so you had to do everything yourself.

For example, in a HTML form with two text boxes, "FirstName", and "LastName", where the user had entered "Joe", "Blow", the C program would receive "FirstName=Joe&LastName=Blow". It would then have to read the string in order to find out what variables are there and what they are set to. Of course, this was a great deal better than the situation before - HTTP is a stateless system, which means it saves no data at all across pages, and therefore even being able to use C to send data across pages was a big step forward.

This problem was solved somewhat by an easier language called Perl, which is a somewhat loose acronym for "Practical Extraction and Report Language", or "Pathologically Eclectic Rubbish Lister", depending on whom you speak to. Perl, originally invented as a generic text-processing language, enabled HTML form-parsing and other fancy tricks to be within reach of even novice programmers.

The Perl design process was simple: you had a Perl script performing all your functionality for a particular page on your web site, into which you embedded any HTML output that was necessary. Perl provided a large variety of functions designed to simplify any task you could dream up, which made it immensely popular with developers.

This was followed up by the CGI.pm module for Perl, which automatically converted values passed from a form into variables, eliminating the need to hand-parse the variables string. A Perl script to parse the same FirstName/LastName form as before could easily take under twenty lines to write, and would be much easier to read.

Even though it was a big step forward for web development, Perl was still far from perfect. Its "one language fits all" thinking meant that it wasn't truly designed for the web, and many Perl programmers placed a large

emphasis on "one-liners" - one line of probably confusing code that performed a certain task - as opposed to structured, easy to read programming.

Perhaps the largest drawback was that Perl was Perl-centric. That is, in order to output HTML content, you had to embed HTML inside the Perl.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Early versions of PHP >>](#)

Previous chapter: [History](#)

Jump to: [Background](#)

Home: [Table of Contents](#)

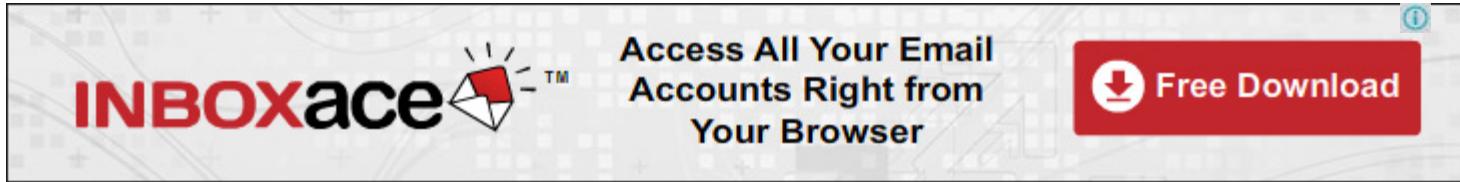


The banner features the INBOXace logo on the left, which includes the word 'INBOXace' in a bold red sans-serif font next to a stylized envelope icon with a red 'envelope' and yellow 'sunburst' design. To the right of the logo is the text 'Access All Your Email Accounts Right from Your Browser'. On the far right is a red button with a white download icon and the text 'Free Download'.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)



Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Early versions of PHP

The original release of PHP was created by Rasmus Lerdorf back in the middle of the 90s as a way of making common web tasks easier and less repetitive. Back then, the main goal was to have the minimum amount of logic as possible in order to achieve results, and this led to PHP being HTML-centric - that is, PHP code was embedded inside HTML.

The first popular version of PHP was called PHP/FI 2.0, for Personal Home Page / Form Interpreter, and, despite its parsing inconsistencies, managed to attract a fair few converts, including myself. The main issue with this version was that the PHP/FI parser was largely hand-written, and so users often encountered scripting errors that were technically not errors - they were just the PHP/FI parser screwing up. Furthermore, the parser was absolutely tied to the Apache web server, and was hardly renowned for its speed.

**Author's Note:** The parser is what takes your script and converts it to something the computer can understand. It comes from the word "parse", which means to break text up into components and analyse them.

Some of these issues were resolved in version 3, when Zeev Suraski and Andi Gutmans re-wrote PHP from the ground up using standard "compiler compiler" tools like Flex and Bison. This made the parser itself all but bulletproof, which in turn gave sanity back to many PHP users.

PHP 3 also finally made the language extensible - something that was seriously lacking from prior versions. Particularly keen developers were able to write their own modules for the language, adding functionality at the core level. The parser itself, though, was still tied to Apache, and, although speed was improved a great deal from PHP/FI, it still was not anything to shout about.

The only downside to upgrading to PHP 3 was that the language was a lot stricter - some code that worked

on PHP/FI would no longer work after upgrading. The language was still young, though, so not many were affected.

The all-round improvement brought about by the PHP/FI to PHP 3 upgrade brought in many new users eager to jump from the Perl ship to a system that was easier to use. At the time, there was no doubt at all that Perl was faster to execute, except perhaps among PHP zealots, however PHP still kept its lead in speed of development, and that was the key selling point.

With PHP 3, the language had gained limited object-oriented support, and this only added extra fuel to the fire of PHP's growth. By the time PHP 3 was replaced in the middle of 2000, it was installed on over 2,500,000 web-site domains, as compared to 250,000 just 18 months before.

In the middle of 2000, PHP 4 was released to the world, containing major differences to PHP 3 in all aspects. Extensive work had been done to ensure that backwards compatibility with older PHP scripts would remain - upgrading from PHP 3 to PHP 4 was much smoother than the PHP/FI to PHP 3 upgrade.

Perhaps the most important change made for PHP 4 was the switch to what is called the Zend Engine. The Zend Engine, created by Zend, a company founded by Zeev Suraski and Andi Gutmans (the name Zend is a contraction of ZEev and aNDi) to promote PHP in the corporate environment, allowed much more flexibility than had ever been seen before in PHP. The engine took over the core of PHP and introduced reference counting, whereby all resources used in scripts (database connections, files, etc) are tracked automatically by the engine, and freed when no longer used to minimise memory usage and ensure there were no memory leaks.

Also introduced with PHP 4 was complete web server abstraction, meaning that PHP now runs on Apache 2, Microsoft's IIS, Zeus and more. This opened use of the language up to the 50% of the world who don't use Apache for their web server.

Performance took a gigantic leap forward due to two main factors. Firstly, the execution paradigm was changed from prior versions. PHP 3 and before used an "execute while interpreting" paradigm which meant that PHP read a line of source code, interpreted it, executed it, read another, interpreted it, executed it, read another, etc. This meant that code was often re-read and re-interpreted twice or more, entirely unnecessarily.

PHP 4, with its new "compile first, execute later" paradigm read your entire script in and compiled it to byte code before execution, which produced a large speed increase - the average speed increase was about 100%, with some benchmarks showing up to a fifty-fold increase in speed when PHP 4 was pushed to its limits.

Author's Note: "Byte code" is the name for the internal representation of your script that PHP can understand easily - it is usually a lot longer than your script as a result of each PHP statement being

broken down into many simple byte code statements.

Furthermore, because PHP 4 compiled the entire script before executing it, it became possible to optimise and cache the compiled code before execution. We will be looking at how this works later on in the book.

Secondly, PHP 4 introduced multi-threading, which essentially allows particularly lengthy, but non-critical functions to be run independently from the main script process, further streamlining execution.

After years of loyal service, support for PHP 4 was discontinued at the end of 2007, and should not be used for any new coding.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Current release >>](#)

Previous chapter: [Background](#)

Jump to: Early versions of PHP

Home: [Table of Contents](#)

 INBOXace™

Access All Your Email Accounts Right from Your Browser

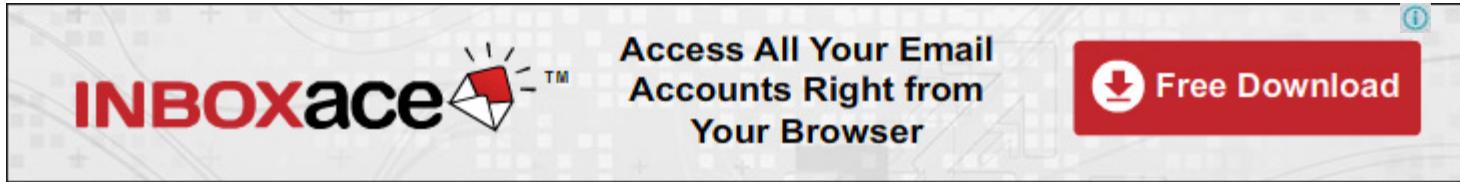
 Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

Current release

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Current release

PHP 5 was a big step forward for the language, although admittedly not as big as the jump from PHP 3 to PHP 4. The release is focused on language maturity, and offers a lot of new functionality that has simply been missing from previous versions simply because the language was a little too simple to properly support larger projects.

PHP 5 brought with it huge steps forward for object-oriented scripts - developers are now able to declare how their objects may be used, which makes it easier for one developer to work with another's code. Furthermore, there is a wide variety of functions available for objects that make them much more flexible and easy to work with - unified constructors are only available from v5 upwards.

PHP 5 also brought with it new error checking in the form of try/catch, which was something that programmers from other languages had been enjoying for a long time. Furthermore, objects are now always handled as references in order to help programmers who just do not understand how objects work.

The biggest improvements, though, were in the extensions: SimpleXML has landed, which is a fast and easy-to-learn way to interact with XML documents, and there's also the flat-file database API SQLite, a new SOAP extension, MySQL Improved, and a lot more.

Since the initial PHP 5.0 release, huge numbers of improvements have been introduced: 64-bit support, namespaces, an opcode cache, built-in debugging, generators, traits and more. Happily, development only seems to be getting faster, so there are even more huge improvements planned for future releases.

At the time of writing, PHP 5.6.8 is the latest release, and is recommended for all new PHP coding.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

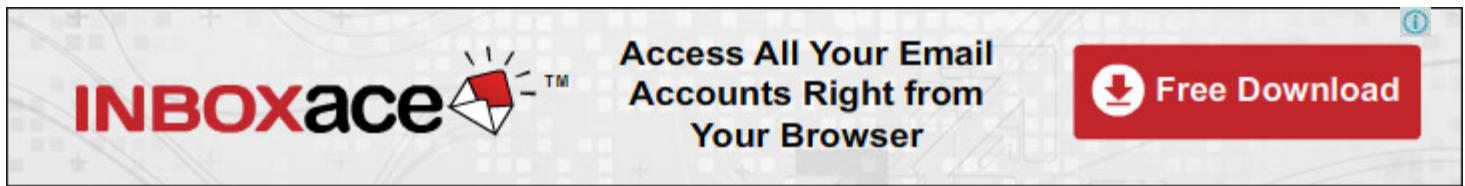
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Upgrading from PHP 3 >>](#)

Previous chapter: [Early versions of PHP](#)

Jump to:      Current release

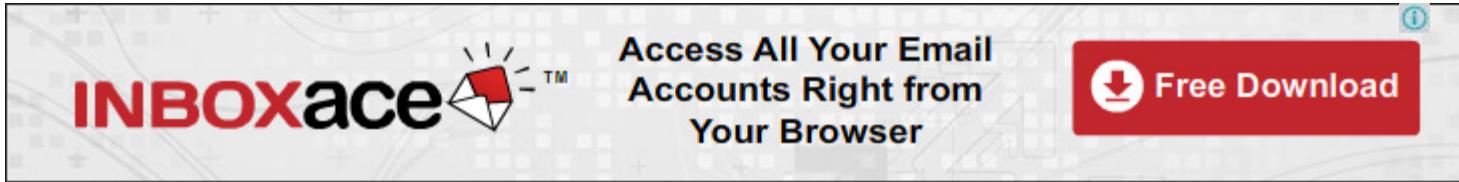
Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word "INBOXace" in red and black with a small envelope icon, and "TM" in a small circle. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". On the far right is a red button with a white download icon and the text "Free Download". A small blue circular icon with the number "1" is in the top right corner of the banner.

Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Upgrading from PHP 3

This chapter has been left in for historical reference. Surely no one is still running PHP 3...?

Even though PHP 4 has been out for years now, some people have yet to upgrade. If this is you, read on - I will be looking at how to take the sting out of upgrading to the latest release of PHP.

To start with, one of my favourite changes made in PHP 4 was the new variable substitution code within strings. All too often in PHP 3, I had found myself writing:

```
print "Array item 3 is " . $myarray[2] . " currently";
```

This is not particularly easy to read, and, thankfully, was improved in PHP 4. You can now put braces (the { and } symbols) around arrays and objects inside strings to have their variable correctly substituted. Thus:

```
print "Array item 3 is {$myarray[2]} currently";
```

More on how arrays work later. As mentioned already, the execution paradigm has changed between version 3 and 4. Owing to the fact that PHP 4 now compiles everything before executing anything, a new requirement was introduced which enforces syntactically complete files for source code.

It used to be the case in PHP 3 that you could start a while loop in one file and end it in another by using *include()*. This trick no longer works - all files you *include()* or *require()* must be syntactically correct on their own merits.

You will also find that no extensions written for PHP 3 will work with PHP 4, even if you recompile them. This is a side effect of PHP 4 having the Zend Engine at its core, and actually requires some re-writing of the extension. Don't worry, though - all the core extensions were re-written long ago.

Perhaps a very likely culprit for a code-breaking change was the behaviour of `setcookie()`. This was notorious to users of PHP 3, as, for one reason or another, PHP 3 would send the HTTP set-cookie headers in the reverse order of what you specified in your code. This has been rectified in PHP 4 - cookies are sent in the order you request them to be sent.

Finally, you need to be wary of the added functionality available in PHP 4. When I made the switch, many of my pages immediately stopped working simply because PHP 4 had a function `in_array()`, whereas PHP 3 did not. As `in_array()` is incredibly useful (it returns TRUE if it finds a given item in an array), I had written my own `in_array()` implementation for PHP 3, and PHP 4 complained that I was trying to redefine a function.

If you have the luxury of a test machine, I would recommend you make the switch to PHP 5 there, and see the results - it is quite likely the biggest incompatibilities will come about because of the change made in PHP 4.1. This change is discussed in the next chapter, and you will need to read it in order to upgrade smoothly from PHP 3.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Upgrading from PHP 4 >>](#)

Previous chapter: [Current release](#)

Jump to: [Upgrading from PHP 3](#)

Home: [Table of Contents](#)





**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

 Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# Upgrading from PHP 4

The difficulty you are likely to encounter when upgrading from PHP 4 will depend on what version of PHP 4 you are upgrading from, because in PHP 4.1 big changes were made regarding variable availability.

If you are upgrading from any version of PHP 4 from before 4.1, you need to be aware that PHP now uses special superglobal arrays to handle data coming from users. Register\_globals, the special php.ini setting that made PHP automatically create variables from user input, is now disabled by default as a security precaution, and you are advised to keep it disabled.

The majority of changes made in PHP 5 relate to object orientation - you now have a lot more control over how your classes are used. For example, you can declare variables as being private, protected, or public depending on how you want other programmers to be able to access them. Objects are now always passed and assigned by reference - when you say \$object\_a = \$object\_b in PHP 5, it is equivalent to \$object\_a =& \$object\_b in PHP 4.

If you have the time, I strongly recommend you read all of the chapter on Objects - there have been many new features added to the language to make object orientation easier and more flexible for programmers, and it is all well worth knowing.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [The creators of PHP >>](#)

Previous chapter: [Upgrading from PHP 3](#)

Jump to: [Upgrading from PHP 4](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

The creators of PHP

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# The creators of PHP

PHP would not be where it is today without the help and effort of thousands of people. A particular few deserve special mention and thanks - if you see them around at a conference, buy them a drink!

PHP was originally created by Rasmus Lerdorf, and he oversaw production of the first release and PHP/FI 2.0. PHP 3 was rewritten from the ground up by Zeev Suraski and Andi Gutmans, and the three of them are the "language architects" behind PHP.

In the PHP group, the developers who primarily work on development of the language core, there are: Thies C. Arntzen, Stig Bakken, Shane Caraveo, Andi Gutmans, Rasmus Lerdorf, Sam Ruby, Sascha Schumann, Zeev Suraski, Jim Winstead, and Andrei Zmievski. Each of them has worked exceptionally hard, along with the help of many others, to put PHP where it is today.

If you have a little time on your hands, visit <http://www.php.net/credits.php> and read the full credits online yourself - you will see that PHP really is a community effort built by the collective desire to build the best language possible.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [The Zend Relationship >>](#)

Previous chapter: [Upgrading from PHP 4](#)

Jump to:      [The creators of PHP](#)

Home: [Table of Contents](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

**Leather Shoes** 570,000vnd  
Chất liệu da bò tự nhiên



# The Zend Relationship

After having revolutionised PHP by creating PHP 3, both Andi Gutmans and Zeev Suraski went on to design and create PHP 4, another revolution for the language - no longer would each line of code need to be interpreted as it was read, because Andi and Zeev made the language compiled.

They did this by creating a new engine for the language, known as the Zend Engine - as well as making scripts compiled, the Zend Engine also offered much more flexibility to the programmers of PHP itself, making it easy to port across platforms and servers. The Zend Engine was the first product to come out from Zend, a company founded by Andi and Zeev to create commercial products for PHP. The Zend Engine itself, being an integral part of PHP, is completely free and always will be, however Zend also produce a variety of other products for PHP, including a development environment, a script encoder, and more.

Having two out of the three language architects working for them gives Zend a big advantage in the market place, and as such they produce some excellent products. We will be covering some of their products, as well as their competitors, later in the book.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Advantages of PHP >>](#)

Previous chapter: [The creators of PHP](#)

Jump to:      [The Zend Relationship](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# The HTML relationship

When used to output HTML content, PHP is embedded inside HTML in code islands, as opposed to in Perl, where HTML code is embedded inside the Perl script. The most common way to open and close PHP code blocks is by `<?php` and `?>`. Here is an example of a simple page, shown in Perl first then in PHP - don't worry about what the code means for now:

```
#!/usr/bin/perl
print "<html>\n";
print "<body>\n";
print "<p>Welcome, $Name</p>\n";
print "</body>\n";
print "</html>\n";
```

And now in PHP:

```
<html>
<body>
<p>Welcome, <?php print $Name; ?></p>
</body>
</html>
```

As you can see, the PHP version is only a line shorter, but infinitely much easier to read because the majority of the page is just HTML. Some modules for Perl (particularly CGI.pm) help, but PHP continues to have a big lead in terms of readability. If you really wanted, you could write your PHP script like the Perl script: switch to PHP mode and print everything out from there. However, it tends to get messy - as you can see!

Apart from editing legibility, another advantage to having most of the page in straight HTML is that it makes

editing with commercial IDEs possible, whereas products like Dreamweaver trash Perl's print statements.

One key advantage to using PHP as opposed to some other solutions is that PHP code is all executed at the server, with the client only receiving the results of the script. What this means is that users never see your PHP source code because they are never sent it - they only see what you want them to see.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

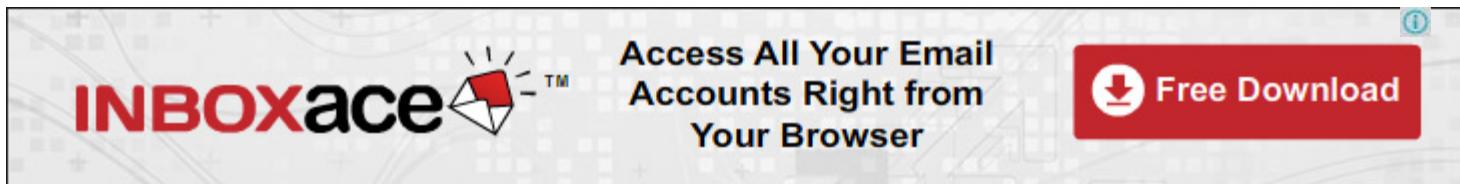
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Interpreting vs. Compiling >>](#)

Previous chapter: [Advantages of PHP](#)

Jump to:      [The HTML relationship](#)

Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word "INBOXace" in red and black with a stylized envelope icon, and the text "Access All Your Email Accounts Right from Your Browser" in the center. On the right, there is a red button with a download icon and the text "Free Download". A small "i" icon is in the top right corner of the banner area.

Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).



Hacking with PHP has been updated for PHP 7 - only \$20! >>

# Interpreting vs. Compiling

PHP uses a blend of interpretation and compilation in order to provide the best mix of performance and flexibility to programmers.

Behind the scenes, PHP compiles your script down to a series of instructions (called opcodes) whenever it is accessed. These instructions are then executed one by one until the script terminates. This is different from conventional compiled languages such as C++ where the code is compiled down to native executable code then that executable is run from then on. Instead, PHP re-compiles your script each time it is requested.

This constant recompilation may seem a waste of processor time, but it is actually not all that bad because you no longer need to worry about hand recompiling your scripts when you make any changes. On the flip side, many scripts take longer to compile than they do to execute!

Furthermore, it provides very quick feedback during development. If you have an error somewhere in your file, PHP will refuse to compile the page until you have fixed the problem, and you are able to step through execution of your code line by line until you find the problem.

The speed hit of regular compilation is nullified entirely by the use of PHP opcode caches.

One major advantage to having interpreted code is that all memory used by the script is managed by PHP, and the language automatically cleans up after every script has finished. This means that you do not need to worry about closing database links, freeing memory assigned to images, and so on, because PHP will do it for you. That is not to say you should be lazy and make PHP do all the work - good programmers clean up themselves, and let PHP work as backup in case something is missed.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Output Control >>](#)

Previous chapter: [The HTML relationship](#)

Jump to:      [Interpreting vs. Compiling](#)

Home: [Table of Contents](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# Output Control

PHP offers a great deal of flexibility as to how you want to output your content. In general use, PHP is embedded inside HTML in code islands started with `<?php` and ended with `?>`.

You can reverse this by writing your whole script as one big PHP code island and printing HTML as necessary. Going back to the example shown previously, we can make our PHP code look almost identical to the Perl code by printing the HTML from inside our PHP code:

```
<?php
    print "<html>\n";
    print "<body>\n";
    print "<p>Welcome, $Name</P>\n";
    print "</body>\n";
    print "</html>\n";
?>
```

"Print" is a simple function that outputs a chunk of text, enclosed in quotation marks, to the client. "\n" means "start new line in the source code", and it serves to lay the source code out nicely. For the longest time, a debate raged on messageboards and mailing lists as to whether it was faster to drop out of "PHP mode" to output large amounts of HTML, or whether it was just as fast to stay in PHP mode. The truth is that it is horses for courses - you will find little or no speed difference either way.

PHP purists like to point out that print is technically not a function, and, technically, they are correct. This is why print doesn't require brackets around the data you pass to it. Other language constructs that masquerade as functions (and are referred to as functions herein for the sake of sanity) include echo, include, require, and exit.

PHP also has great output buffering features that further increase your control over the flow of output. An output buffer can be thought of as a storage hole where you can queue up content for outputting. Once you start a buffer, any output you create is automatically put into that buffer, and is not seen unless the buffer is closed and flushed - that is, sent to your visitor.

The advantage to this output queuing is two-fold. First, it allows you to clean the buffer if you decide not to output the current output queue in the buffer. When a buffer is cleaned, all the output stored in there is deleted as if it were never there, and output for that buffer is started from scratch.

Secondly, output buffering allows you to break the traditional ordering of web pages - that of headers first and content later. Owing to the fact that you queue up all your output, you can send content first, then headers, then more content, then finally flush the buffer. PHP internally rearranges the buffer so that headers come before content.

Output buffering is covered in much more depth later.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Performance >>](#)

Previous chapter: [Interpreting vs. Compiling](#)

Jump to:      [Output Control](#)

Home: [Table of Contents](#)





Access All Your Email  
Accounts Right from  
Your Browser



Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Performance

Prior to the release of PHP 4, Perl mongers were more than happy to offer head-to-head tests between a PHP script and a similar Perl script, because they were safe in the knowledge that Perl outperformed PHP pretty much hands-down. However, since PHP 4, and particularly since PHP 5.1, PHP has really made big progress in terms of raw speed.

The change between v3 and v4 was largest, though - to give you an idea of the speed improvement, I wrote a simple test script that works in both PHP 3 and PHP 4. It simply creates an array of 1000 different random numbers, then sorts the array. When executing this script 1000 times, PHP 3 managed to achieve 19.51 requests per second on a 500MHz Linux box, compared to PHP 4's 43.08 requests per second - quite a difference, and the program was only four lines long!

So, PHP 4 performance was certainly nothing to be sniffed at. You will see a much bigger performance difference when using a more complicated script, and even higher numbers if you use the opcode cache that's built in as of PHP 5.5.

Later on in the book I will be discussing how to maximise your script performance by squeezing out every drop of performance available.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

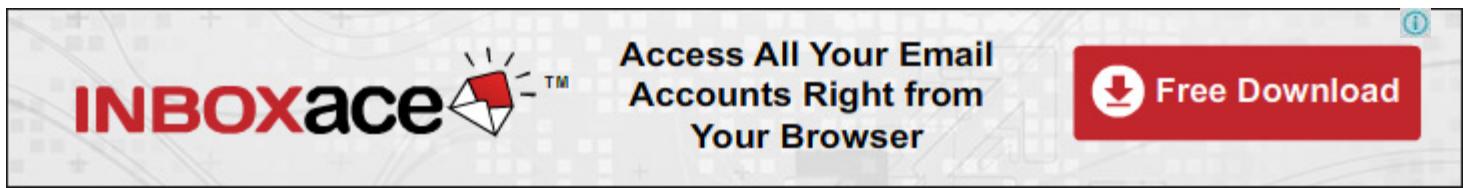
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Competing Languages >>](#)

Previous chapter: [Output Control](#)

Jump to:      [Performance](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

Vntrip.vn

Đặt khách sạn Online giá rẻ. Ứng dụng dành riêng cho người Việt



# Competing Languages

Much to the chagrin of a large group of developers, PHP is not the only option available to web developers, and, although I really am leaving myself open to flame-mail here, PHP *is not* always the best option, either.

In order to make sure your PHP education is rounded, I want to introduce you briefly to the other web development languages on offer, how they compare to PHP, and, where appropriate, when a language is a better choice than PHP for a certain task.

## 2.2.5.1 Perl

Perl is the most popular of the PHP alternatives out there, arguably because it is also the oldest. There is a large installed base of Perl out there; many open-source projects require Perl to be installed to work properly. It has the advantages of being very (very!) flexible, and also having a large collection of modules already written for it. However, it is let down by the fact that it is very easy to write obfuscated and confusing Perl without really realising you are doing so, and this has resulted in such marvels as the annual Obfuscated Perl Contest.

Well-written Perl scripts often look fairly like their PHP equivalent. The major cause for Perl's messy appearance is that many Perl programmers rely on "one-liners" - packing large amounts of functionality into just one line of code. Perl was once described very accurately by its creator, Larry Wall, when he argued that the front cover for his O'Reilly book on Perl should be a camel, saying that Perl was ugly but serviceable, and able to go long distances without much nourishment.

Perl is often a better choice when you want to take advantage of some of the pre-written libraries. CPAN, Perl's library repository, is very big, and there is a huge range of code for you to take, customise, and re-use. Perl also has a very active - and very cool - hacker community around it that's a whole lot of fun to be part of and is really a bedrock of support when you need it. Larry Wall and Damian Conway (both core Perl developers) are both rightfully revered as "alpha geeks" - people who really push the envelope of

programming by doing cool new things. They are both very friendly, and attend many conferences year round - go ahead and introduce yourself if you meet them, because they really are fascinating to talk to.

### 2.2.5.2 ASP.NET

ASP.NET is Microsoft's attempt to succeed in the web development market, and comes as standard with their web server, IIS. ASP.NET has been mauled by the open source community ever since it came out, and they gave a variety of reasons: it is proprietary, single platform (Windows), and slow.

I would like to say, "Yes, yes, and yes", but I'm not going to try to pull the wool over your eyes. The reality is that ASP.NET has been implemented on other platforms, and, when running on Windows and Microsoft Internet Information Services (IIS), is actually lightning-fast thanks to its .NET back-end.

That coupled with the fact that you can write back-end code for ASP.NET using C#, VB.NET or any other .NET language would make the whole solution very attractive indeed if it were not for the fact that ASP only really works well on IIS. On other platforms there are many fewer features, and it generally runs a great deal slower. When running on Windows, the licensing cost tends to be the most important thing, particularly when an all-Microsoft solution stack is being used.

ASP.NET is generally favoured when an all-Microsoft stack is in place.

**Author's Note:** If you're migrating from ASP to PHP, you might consider using ASP2PHP, a freeware converter between the languages. Although it isn't perfect, it can give you a big head start if you are trying to migrate a large project. Your best option is to use the tool, then go over the generated scripts by hand to make sure there are no bugs or performance issues. You can download ASP2PHP from <http://asp2php.naken.cc>.

### 2.2.5.4 Ruby on Rails

I'm not going to pretend that Ruby's syntax is nice, because it's not. But Rails manages to make it cool by doing a huge amount of the hard work for you - it leaves PHP in the dust in terms of usability. Rails development is quite closely associated with Apple, and the two share one big thing in common: once you switch to them, you rarely switch away.

In the case of Rails, it's sadly not because of its awesome performance - Rails is dog slow compared to PHP, which is why you rarely see it used on large websites. However, a common programming adage is "computer time is cheap; programmer time isn't" so if you have huge system resources at your disposal and don't mind using up a few thousand extra CPU cycles in exchange for faster development, Rails might be for you.

## 2.2.5.4 JSP

Java Servlet Pages has often been considered the "dark horse" in web scripting because at first many thought it would be overkill for the job and yet has managed to get quite a substantial community about it nonetheless. JSP has three key advantages over some of its competitors, which may be why it has done so well:

1. It uses Java, a language that already has a large skill set of developers and a massive amount of functionality available. Java is also conducive to scalability as it distributes across multiple computers well.
2. Sun, as well as other members of the community, has worked hard to promote the language and tools that support it, which means that JSP has a lot of backing inside larger enterprises.
3. It strongly encourages templating of pages for maximum code re-use. Templates for PHP are widely available, but they are a great deal more popular in JSP.

It is a common argument that because JSP is based on Java it scales better than PHP. This is not correct *per se* in the same way that most other over-generalisations are not correct (yes, I realise that is an over-generalisation too, and hence you are free to enjoy the irony!) - PHP scales perfectly well as long as you write your PHP scripts using the same design patterns you would have used writing your JSPs.

JSP is a popular choice when existing back-end business logic is written in Java also, as this keeps the development team language-homogenous.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

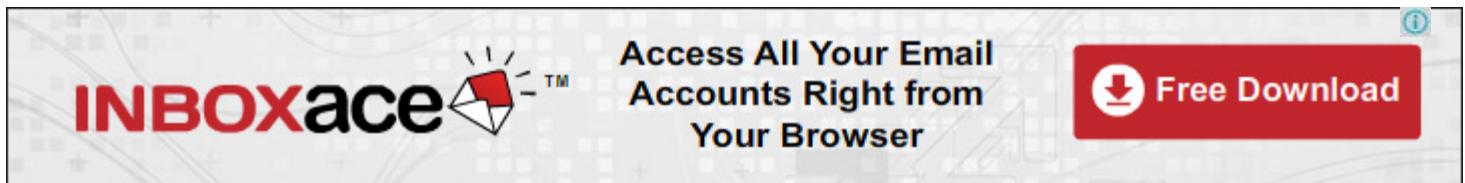
[Tweet](#)

Next chapter: [When to use PHP >>](#)

Previous chapter: [Performance](#)

Jump to: [Competing Languages](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

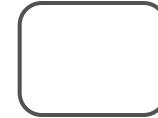
Hacking with PHP

When to use PHP

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Editable AJAX TreeGrid

The most complex tables, charts or grids on Internet. New version 12.0



# When to use PHP

Owing to its low learning curve, easy maintenance, and overall fast execution time, it is rare to find PHP is not the best choice for a web application. Homepages (big and small), database front-ends on the web, command line shell scripts where you want extra functionality, and even basic GUI development are all popular uses for PHP, and it excels at them all.

If you are doing anything that displays its text through a web browser, you are likely to find that PHP is your best choice. An increasing number of people are using Python for their web development, but it's still a tiny, tiny group compared even to Perl.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [When not to use PHP >>](#)

Previous chapter: [Competing Languages](#)

Jump to: When to use PHP

Home: [Table of Contents](#)

# Plsql Developer

Download the 30 day trial version for PL/SQL IDE!



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

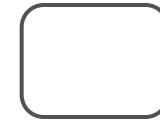
Hacking with PHP

When not to use PHP

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Editable AJAX TreeGrid

The most complex tables, charts or grids on Internet. New version 12.0



# When not to use PHP

There are two situations when I would not recommend PHP for use in a web application. Firstly, as already mentioned, if you desperately need all the performance you can get, you should be using C or C++. Both languages are unwieldy on the web, but they offer superior performance. Even in this situation you might want to consider simply writing your C code as extensions to PHP.

Secondly, if you are adding to a system which already has a lot of code already written in another language, then coding parts in PHP will only make your maintenance job more difficult, so you are likely to find it easiest in the long run to keep using the old language.

Author's Note: It's important for me to make the point that, although PHP is thread-safe, its libraries may not be. If you don't know what thread-safe means, you don't need to worry about it!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

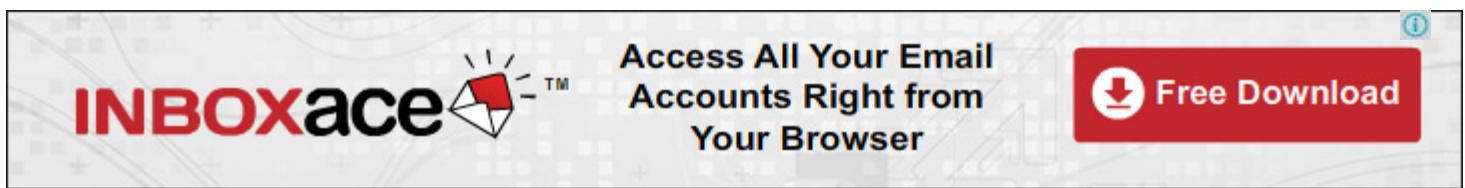
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Selling PHP to your boss >>](#)

Previous chapter: [When to use PHP](#)

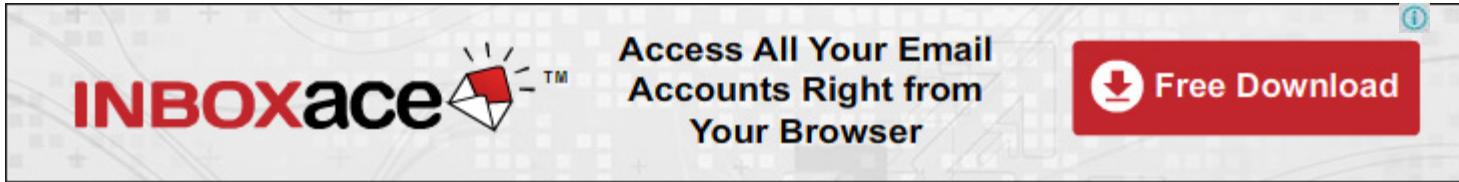
Jump to: [When not to use PHP](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Selling PHP to your boss

If you want to use PHP in your company and your manager favours another solution, or if you are trying to convince a potential client that PHP really is a superior choice for the web, you're going to need to have a clear-cut set of reasons why you believe PHP is the superior language. This short list should help you get started:

- PHP is cross-platform. It can be run on Windows, Linux, BSD, Mac OS X, and Solaris, as well as a variety of other platforms.
- PHP is free. You can download the source code, use it, and even make changes to it without ever having to pay any licensing costs. You can even give away your own modified version of PHP.  
Note to critics: just because PHP is free, it doesn't mean you need to give your scripts away for free.
- PHP is fast. In the majority of scripts beyond basic benchmarks, PHP will easily compete with both Perl and Python, and usually match Microsoft's ASP.NET. Add to that the fact that PHP code can be cached for execution, and PHP's performance is first-class.
- PHP is capable. There are thousands of pre-written functions to perform a wide variety of helpful tasks - handling databases of all sorts (MySQL, Oracle, MS SQL, PostgreSQL, and many others), file uploads, FTP, email, graphical interfaces, generating Flash movies, and more.
- PHP is extendable. Writing your own extension to PHP is a common and easy way to implement speed-critical functionality, and PHP's extension API is a particularly rich and flexible system.
- PHP is reliable. It already runs on millions of servers around the world, which means it is mature enough for even the most demanding of situations.
- PHP is easy to debug. There are a number of debuggers, both commercial and freeware, that make debugging PHP a snap.

- PHP is supported commercially. Two of the main contributors to PHP founded a company, Zend, to sell supporting products and technical support for the language, so there is no need to worry about PHP not being supported by a big company.
- PHP is supported by the community. There are several very popular PHP web-sites such as [www.phpbuilder.com](http://www.phpbuilder.com) that provide user-run technical support for PHP, as well as a variety of official mailing lists to help you get answers when you need them.
- PHP is advancing. With the release of PHP 5, PHP has introduced features that have long been waited for, including more comprehensive error handling, better object orientation, and, of course, more speed.
- PHP is fun! As I am sure you will agree while using this book, PHP is an enjoyable language to use - very little time is spent debugging code, and there is a large selection of pre-written functions available. OK, so your boss might not care too much about this one...

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Extending PHP >>](#)

Previous chapter: [When not to use PHP](#)

Jump to:      [Selling PHP to your boss](#)

Home: [Table of Contents](#)





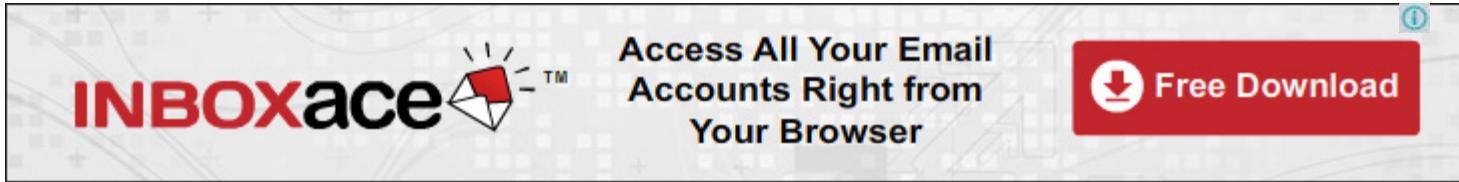
**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

 Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Extending PHP

The base of the PHP language is very simple, having just enough to set and retrieve variables, work with loops, and check whether a statement is true or not. The real power behind PHP comes with its extensions - add-ons to the base language that give it more flexibility. There are hundreds of extensions to PHP, and they can be broken down into five distinct types: core, bundled, PECL, third party, and DIY.

- Core extensions are extensions that are bundled with PHP itself, and enabled by default. For all intents and purposes they are part of the base language, because, unless you explicitly disable them (few people do, and sometimes you cannot), they are available inside PHP. For example, the mechanism to handle reading and saving files in PHP is actually handled by an extension that is automatically compiled into PHP.
- Bundled extensions are extensions that are bundled with PHP, but not enabled by default. These are commonly used, which is why they are bundled, but they are not available to you unless you specifically enable them. For example, the mechanism to handle graphics creation and editing is handled by an extension that is bundled with PHP, but not enabled by default.
- PECL stands for "PHP Extension Community Library", and is as a subset of the PHP Extension and Application Repository, PEAR. PECL (pronounced "pickle") was originally created as a place where rarely used or dormant bundled extensions could be moved if they were no longer considered relevant. PECL has grown a lot since its founding, and is now the home of many interesting and experimental extensions that are not quite important enough to be bundled directly with PHP.
- Third-party extensions are written by programmers like you who wanted to solve a particular problem that was unsolvable without them creating a new extension. There is a variety of third-party extensions available out there, with the sole difference between a third-party extension and a PECL extension is that there are various rules about having code being submitted to PECL. Third-

party extensions are frequently unstable, and often just downright crazy. That is not to say they are bad - give them a shot and see what you can do.

- Finally, Do-It-Yourself (DIY) extensions are simply extensions you created yourself. PHP has a remarkably rich extension creation system that makes it quite simple to add your own code as long as you know C. Later on we'll be going through the task of creating your first extension from start to finish. Note that creating your own extension requires that you have the ability to compile PHP - this is not a problem if you use Unix, but is very difficult for Windows users without the correct software.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [PEAR >>](#)

Previous chapter: [Selling PHP to your boss](#)

Jump to: [Extending PHP](#)

Home: [Table of Contents](#)

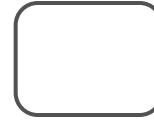


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Php Email

Organize all Your E-Mail Accounts. InboxAce - Fast & Free! Get App.



# PEAR

The PHP Extension and Application Repository, or PEAR for short, contains code written by other programmers to achieve various goals. The concept is simple: other programmers, usually specialists in their areas, write very complicated code, and present it to you in an easy-to-use form, thereby enabling you to create very powerful scripts using just a few simple commands.

PEAR contains two types of pre-written code: PECL code, and PHP code. PECL code, as mentioned already, forms full extensions written in C that interact with external libraries. Extensions reside in PECL when they are considered useful, but not popular or much used. Most of PEAR, however, is PHP code that performs all varieties of tasks for you, with the primary advantage being that you can use PEAR code on any PHP server without needing to enable any extensions or re-compile PHP.

The most famous package in PEAR is called PEAR::DB, and provides an object-oriented, database-independent framework for reading and writing to your database. PEAR::DB is covered in depth later on.

PHP comes with "go-pear", an easy way to configure PEAR for use on your computer. To use it, simply run go-pear from the command line and follow the on-screen instructions. Windows users will need to change directory to where PHP is, e.g. c:\php.

Once you have PEAR installed on your system, you will see the "pear" command - this allows you to search for and download new PEAR modules for your PHP installation.

Author's Note: I've had people email me asking why my code examples rarely make use of PEAR - sometimes I write things from scratch rather than using the pre-built PEAR code. Well, there are two reasons for this: first, if I teach you how to use PEAR, you won't learn how to do it yourself. This is not necessarily a bad thing; certainly I wouldn't want to try re-implementing parts of C++'s Standard Template Library! However I think there's a lot to be learnt by writing things yourself.

The second - and important - reason is that PEAR isn't installed on that many web servers, or if it is installed it may not be kept up to date. So if I had used PEAR in my code examples, many thousands of you wouldn't be able to use them because your host provider doesn't support it! That, in a nutshell, is why I haven't used PEAR very much - sorry!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Running PHP scripts >>](#)

Previous chapter: [Extending PHP](#)

Jump to: [PEAR](#)

Home: [Table of Contents](#)

## Php Email

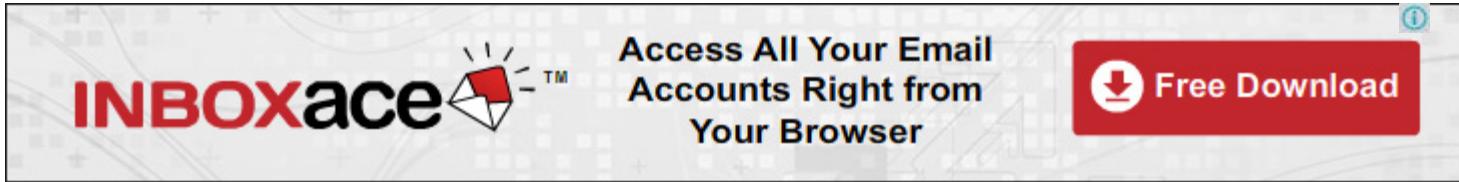
Organize all Your E-Mail Accounts. InboxAce - Fast & Free! Get App.



Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).



Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Running PHP scripts

You can execute your scripts in one of two ways: through a web server where the output is a web browser, or through the command line where the output is the command line itself. Of the two, the former is much more popular, but the latter is steadily growing in popularity.

We cover both methods of script execution in this book, with the bias towards web server usage. Most scripts will work smoothly in either environment, with the exception of scripts covered in the chapter "Alternative PHP uses", which require specific environments. You're encouraged to try using both when reading this book - it will give you more familiarity with how PHP works, and will also give you good experience using the CLI SAPI to debug your scripts.

The primary difference between outputting text to the command line and to a web browser is the format of new lines - through the CLI you need to use `\n` for a new line, whereas for web browsers you need to use the HTML line break, `<br />`. If you want to take a script designed for CLI and make it work through the web, swap `\n` for `<br />`, and vice versa for converting web scripts to command line scripts.

Running scripts through your web server is as simple as putting the PHP script into your web server's public directory, then navigating to the appropriate URL with your browser. Hopefully you will already have followed the official PHP installation instructions for your platform!

Running scripts through the command line is done using the CLI SAPI, which, if you are using Windows, can be found in the directory of your PHP installation, which is probably `c:\Program Files\PHP`. If you are using Unix, you may find the CLI SAPI is available in a special package called something like `php5-cli` or just `php-cli`.

Later on we'll look at more detailed information on how using the CLI SAPI is different from normal PHP scripting, so don't rush off and try it out just yet!

If you are not sure about whether PHP is set up on your machine correctly, the best way forward is to run

the following script:

```
<?php  
    phpinfo();  
?>
```

That calls a special function, *phpinfo()*, which outputs all the information PHP currently has - how it was configured, what server it is running on, what modules are available, and more. It is very handy to keep around when you are developing, as it will answer most questions you have about configuration.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [How PHP is written >>](#)

Previous chapter: [PEAR](#)

Jump to: [Running PHP scripts](#)

Home: [Table of Contents](#)

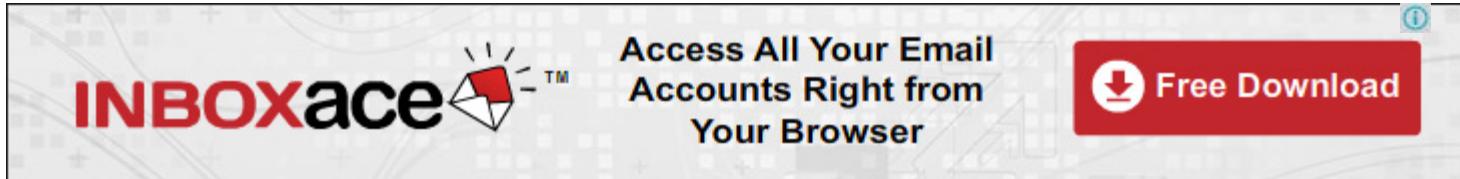
 INBOXace™

Access All Your Email Accounts Right from Your Browser

 Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# How PHP is written

As explained already PHP code is embedded inside HTML, making a complete PHP script. By default, PHP operates with PHP mode turned off - all text read in a script is considered to be HTML unless PHP mode has been enabled.

This method of parsing means that the PHP elements of a script are "code islands"; standalone chunks of code that can work independently of the HTML "sea" around them. That is not to say that the PHP code cannot affect the HTML - far from it! We will get there soon.

PHP scripts are generally saved with the file extension .php to signify their type. Whenever your web server is asked to send a file ending with .php, it first passes it to the PHP interpreter, which executes any PHP code in the script before returning a generated file to the end user. Each line of PHP code is called a statement, and ends with a semi-colon to signify it is a complete statement.

Variables in PHP, that is, objects that store data, all begin with \$ followed by a letter or an underscore, then any combination of letters, numbers, and the underscore character. This means you may not start a variable with a number, so be careful. Here is a list of valid and invalid variable names:

\$myvar	Correct
\$Name	Correct
\$_Age	Correct
\$__AGE__	Correct
\$91	Incorrect - starts with a number
\$1Name	Incorrect - starts with a number
\$Name91	Correct; numbers are fine at the end and after the first character
\$_Name91	Correct
\$Name's	Incorrect no symbols other than _ are allowed, so apostrophes ' are bad.

Variables are case sensitive, which means that **\$Foo** is not the same variable as **\$foo**, **\$FOO**, or **\$fOO**.

Assigning variables is as simple as using the equals operator (=) on a variable, followed by the value you want to assign. Here is a basic script showing assigning and outputting data - note the semi-colons used to end each statement:

```
<?php
    $name = "Paul";
    print "Your name is $name\n";
    $name2 = $name;
    $age = 20;
    print "Your name is $name2, and your age is $age\n";
    print 'Goodbye, $name!\n';
?>
```

As you can see, we set the `$name` variable to be the string "Paul", and PHP lets us print out that variable alongside "Your name is". Therefore, the output of the first print statement is "Your name is Paul", because PHP will substitute `$name` for its value whenever it finds it by itself, or inside a double-quoted string (that is, one starting and ending with "").

**Author's Note:** The technical term for "variable substitution" is "variable interpolation".

We then set `$name2` to be `$name`, which effectively copies `$name`'s information into `$name2` entirely. `$name2`, therefore, is now also set to "Paul". We also set up the `$age` variable to be the integer 20. Our second print statement outputs both variables at once, as again PHP will substitute them inside the string.

The last print statement will not do what was expected, however. It will print:

Goodbye, \$name!\n

The reason for this is that PHP will not perform variable substitution for variables inside single-quoted strings, and won't even replace the escape characters. Whereas in double-quoted strings, PHP will replace the variable `$name` with its value, in a single-quoted string, PHP will consider `$name` to mean that you actually want it to output the word "\$name" just like that.

One final note before we go onto other topics. Occasionally you may run into the situation where you want to tack a value onto your variable inside your strings, but you find that PHP considers these characters to be part of the variable. Take a look at this code:

```
<?php
    $food = "grapefruit";
```

```

    print "These $foods aren't ripe yet.";
?>

```

Ideally what we wanted to be printed was "These grapefruits aren't ripe yet", but because we have added the S to the end of the variable name, we have changed it from trying to read `$food` to trying to read `$foods`. The variable `$foods` does not exist, so PHP will leave the space blank. How to solve this? Well, we need to make it clear to PHP where the variable starts and ends, and there are two ways to do this:

```

<?php
$food = "grapefruit";
print "These ${food}s aren't ripe yet.";
print "These {$food}s aren't ripe yet.";
?>

```

Both of those are valid and should work as expected. The braces, { and }, tell PHP where the variable ends. Note that code like this will work without modification:

```

<?php
$food = "grapefruit";
print "This $food's flavour is bad.";
?>

```

That will work fine because you are not allowed to use an apostrophe as part of your variable names. If you want to make your life easy you can just avoid the problem entirely by not getting PHP to substitute the variable:

```

<?php
$food = "grapefruit";
print "These " . $food . "s aren't ripe yet.";
?>

```

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

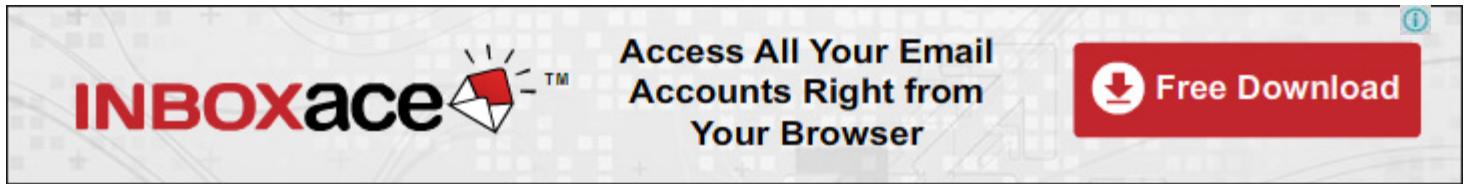
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Whitespace >>](#)

Previous chapter: [Running PHP scripts](#)

Jump to: [How PHP is written](#)

Home: [Table of Contents](#)

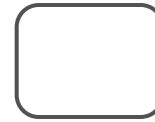


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Get our free pH Brochure

Collection of methods, optimized to work with SevenExcellence.



# Whitespace

Spaces, tabs, and new lines in between statements have no effect on how the code is executed. For example, this next script is the same as the previous script, even though it is laid out quite different:

```
<?php  
    $name = "Paul"; print "Your name is $name\n";  
    $name2 = $name; $age = 20;  
  
    print "Your name is $name2, and your age is $age\n";  
  
    print 'Goodbye, $name!\n';  
?>
```

It is generally recommended that you use whitespace to separate your code into clear blocks, so that it can almost be understood simply by visually inspecting the layout.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Escape sequences >>](#)

Previous chapter: [How PHP is written](#)

Jump to:      Whitespace

Home: [Table of Contents](#)

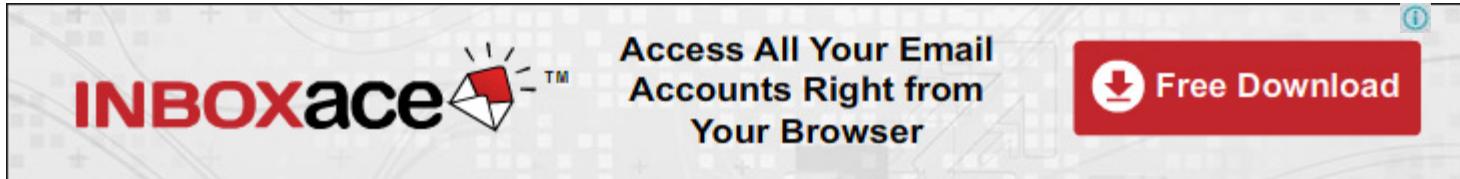
# Plsql Developer

Download PL/SQL Developer lots of features, plug-ins & more



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Escape sequences

You can achieve the same effect in double-quoted strings by using the escape character, which, in PHP, is a backslash \.

Escape sequences, the combination of the escape character \ and a letter, are used to signify that the character after the escape character should be treated specially. For example, if you wanted to have the string "And then he said, "That is amazing!", which was true", you would need escape characters because you have double quotes inside double quotes. Here is a list of the escape sequences in PHP:

\"	Print the next character as a double quote, not a string closer
\'	Print the next character as a single quote, not a string closer
\n	Print a new line character (remember our print statements?)
\t	Print a tab character
\r	Print a carriage return (not used very often)
\\$	Print the next character as a dollar, not as part of a variable
\\\	Print the next character as a backslash, not an escape character

Here is a code example of these escape sequences in action:

```
<?php
$MyString = "This is an \"escaped\" string";
$MySingleString = 'This \'will\' work';
```

```
$MyNonVariable = "I have \$zilch in my pocket";
$MyNewline = "This ends with a line return\n";
$MyFile = "c:\\windows\\system32\\myfile.txt";
?>
```

It is particularly common to forget to escape Windows-style file system paths properly, but as you can see, it is simply a matter of adding more backslashes in there. If you were to print `$MyFile`, you would get this:

```
c:\\windows\\system32\\myfile.txt
```

This is because the escape characters are just to make sure PHP can read the string correctly - once the data has been read, it is converted into the original format.

Along the same lines, escape sequences only work in double-quoted strings - if you type 'Hello!\n\n\n', PHP will actually print out the characters \n\n\n rather than converting them to new lines. It is important to note that escape characters are just one character in files themselves, however they are represented as two in PHP because they cannot physically be typed using your keyboard.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

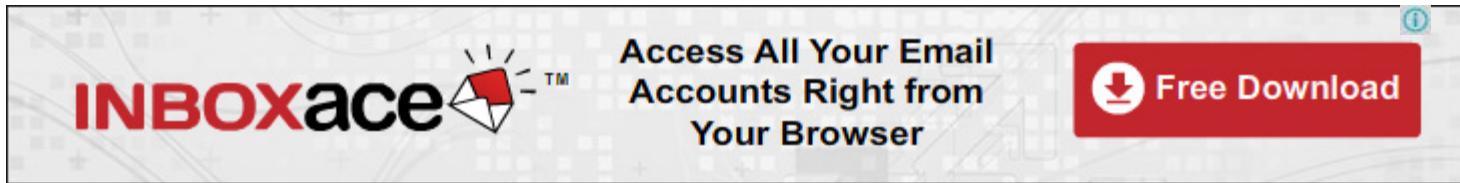
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Heredoc >>](#)

Previous chapter: [Whitespace](#)

Jump to:      [Escape sequences](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Heredoc

In order to allow people to easily write large amounts of text from within PHP, but without the need to constantly escape things, heredoc syntax was developed. Heredoc might be a little tricky to understand at first, but it's actually a big help. Put simply, it allows you to define your own string limiter so that you can make it something other than a double or single quote. So, for example, we could use the string "EOT" (end of text) for our delimiter, meaning that we can use double quotes and single quotes freely within the body of the text - the string only ends when we type EOT.

It is a little more complicated than that in practice, but not much - the string delimiter needs to be by itself on a line, in the very first column. That is, you cannot add spacing or tabs around it.

Take a look at this example:

```
<?php
$mystring = <<<EOT      This is some PHP text.
    It is completely free
    I can use "double quotes"
    and 'single quotes',
    plus $variables too, which will
    be properly converted to their values,
    you can even type EOT, as long as it
    is not alone on a line, like this: EOT;
?>
```

There are several key things to note about heredoc, and the example above:

- You can use anything you like; "EOT" is just an example
- You need to use <<< before the delimiter to tell PHP you want to enter heredoc mode
- Variable substitution is used in PHP, which means you do need to escape dollar symbols - if you do

not, PHP will attempt variable replacement.

- You can use your delimiter anywhere in the text, but not in the first column of a new line
- At the end of the string, just type the delimiter with no spaces around it, followed by a semi-colon to end the statement

Without heredoc syntax, complicated string assignments can quickly become very messy. Heredoc is not used all that often in the wild - very often you will wish it were used more, because too many scripts you will come across have messy code as a result of not using heredoc!

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Brief introduction to variable types >>](#)

Previous chapter: [Escape sequences](#)

Jump to:      [Heredoc](#)

Home: [Table of Contents](#)



**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

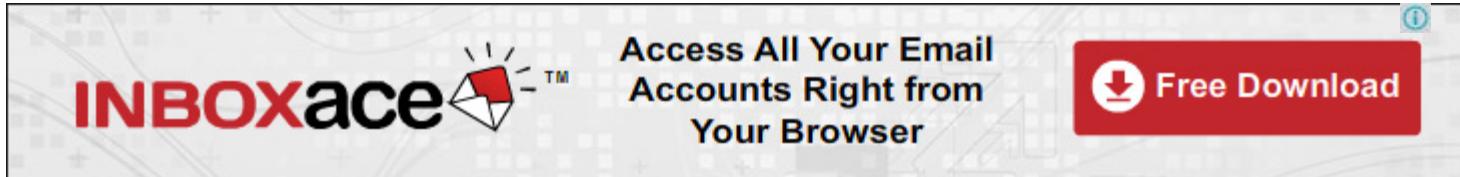
 **Free Download**

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

Brief introduction to variable types

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Brief introduction to variable types

Variables in PHP can be of type integer (a whole number), floating-point (usually called "float"; a fractional number), string (a set of characters), array (a group of data), object (a complex mix of data and functionality), or a resource (any outside information, such as picture data). We will be looking at data types in more depth later on; for now, you only need to know what variables are and how they work.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Code blocks >>](#)

Previous chapter: [Heredoc](#)

Jump to: Brief introduction to variable types

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>

# Code blocks

One key thing to note is that PHP makes extensive use of code blocks - chunks of PHP code that are separate from the rest of the script. As you read the following sections in this chapter, you will notice that PHP uses braces, { and }, to open and close code blocks - I will be explaining it more when you see it.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Opening and closing code islands >>](#)

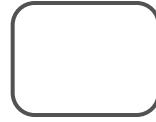
Previous chapter: [Brief introduction to variable types](#)

Jump to:      [Code blocks](#)

Home: [Table of Contents](#)

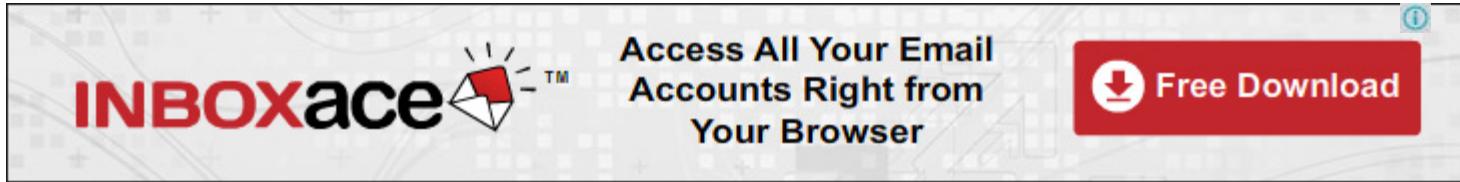
# Get our pH Guide for free

The pH Life Science guide gives you a 360° view with practical tips.



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Opening and closing code islands

There are two main ways to open a PHP code island (to enter PHP parsing mode), and you are welcome to choose which you prefer. The most common (and recommended) manner is to use `<?php` to enter PHP mode, and `?>` to leave PHP mode, however you can also use the short tags version, `<? and ?>`.

The short version has one big advantage and one big disadvantage. The advantage is that you can output information from your script by using a special short tags hack, `<?=`. Here is how that looks:

```
<?="Hello, world!"  
?>
```

Here is the equivalent written using the standard open and closing tags:

```
<?php  
    print "Hello, world!";  
?>
```

As you can see, the short tags version is much shorter, if a little harder to read. However, the big downside to the short version is that it clashes with XML, which also uses `<?` to open code blocks. This means that if you try to use XML and short-tagged PHP together, you'll almost certainly encounter problems - this is the primary reason people recommend using the normal open and close tags.

You can switch into and out of PHP mode by using `<?php` and `?>` whenever you want to and as often as you want to.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

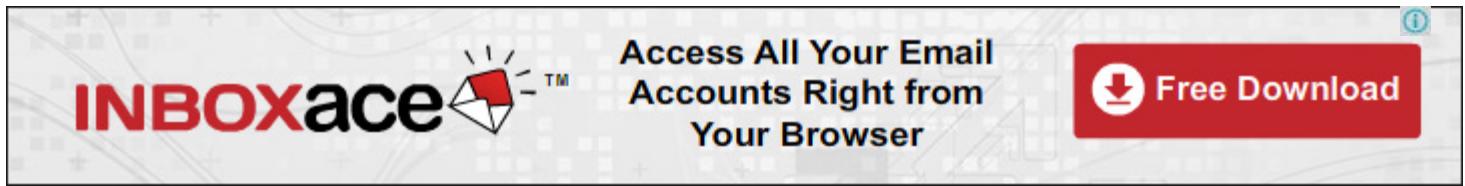
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Comments >>](#)

Previous chapter: [Code blocks](#)

Jump to:      [Opening and closing code islands](#)

Home: [Table of Contents](#)



The image shows a promotional banner for INBOXace. On the left is the INBOXace logo, which consists of the word "INBOX" in red and "ace" in black, with a small envelope icon and a trademark symbol. To the right of the logo is the text "Access All Your Email Accounts Right from Your Browser". Below this text is a red button with a white download icon and the text "Free Download". In the top right corner of the banner is a small blue information icon.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

	<b># of Files:</b> <b>Files Types:</b>	30+ PDF, DOC, TXT, RTF, XLS PPT, BMP, JPG, TIFF, AZW + more	<small>compatible with:</small>   	<a href="#" style="color: red; border-radius: 10px; padding: 5px 10px;">Click to Install</a>
	<b>Runs On:</b>	Windows ®7, 8, 10, Vista™, XP™ Operating Systems		<small>Convert PDF Files for FREE</small>

# Comments

While in PHP mode, you can mark certain parts of your code as a comment that should not be executed. PHP has three ways of doing this: `//`, `/* */`, and `#`. `//` and `#` mean, "ignore the rest of this line", whereas `/* */` means "ignore everything until you see `*/`". Some complications exist with `/*` and `*/`, though, which make them less desirable to use.

```
<?php
  print "This is printed\n";
  // print "This is not printed\n";
  # print "This is not printed\n";
  print "This is printed\n";
  /* print "This is not printed\n";
  print "This is not printed\n"; */
?
>
```

That chunk of code shows all three types of comments in action, but does not demonstrate the problem with the `/* */` form of commenting. If you were to start a `/*` comment on line one, and end it on the line near the bottom where the other `/*` comment is started, you would find that the script would fail to work. The reason for this is that you cannot stack up, or "nest" `/* */` comments, and attempting to do so will usually fail spectacularly.

It is generally best to stick to `//` for your commenting purposes, simply because it is easier to spot, easy to read, and easy to keep control of.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a

downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

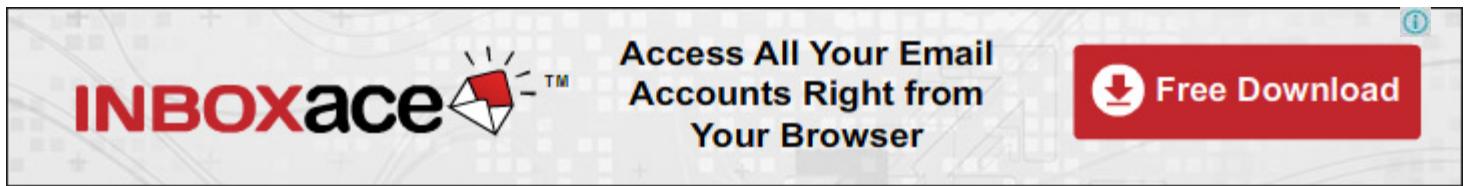
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Conditional statements >>](#)

Previous chapter: [Opening and closing code islands](#)

Jump to:      [Comments](#)

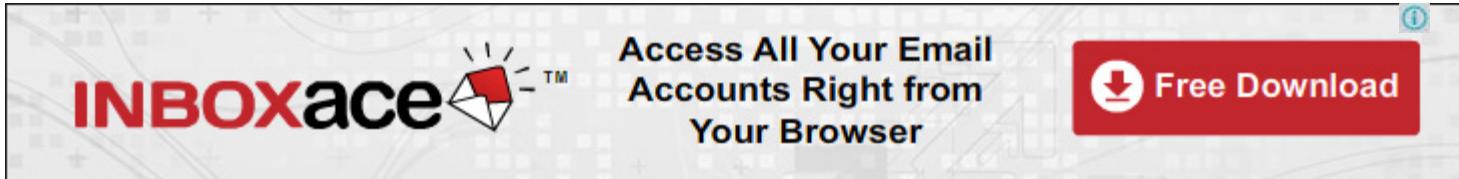
Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word 'INBOX' in red and 'ace' in black, with a small envelope icon and a trademark symbol. To the right of the logo is the text 'Access All Your Email Accounts Right from Your Browser'. On the far right is a red button with a white download icon and the text 'Free Download'.

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Conditional statements

PHP allows you to choose what action to take based upon the result of a condition. This condition can be anything you choose, and you can combine conditions together to make for actions that are more complicated. Here is a working example:

```
<?php
$Age = 20;
if ($Age < 18) {
    print "You're young - enjoy it!\n";
} else {
    print "You're not under 18\n";
}

if ($Age >= 18 && $Age < 50) {
    print "You're in the prime of your life\n";
} else {
    print "You're not in the prime of your life\n";
}

if ($Age >= 50) {
    print "You can retire soon - hurrah!\n";
} else {
    print "You cannot retire soon :( ";
}
?>
```

PHP evaluates if statements left to right, meaning that it first checks whether `$Age` is greater or equal to 18, then checks whether `$Age` is less than 50. The double ampersand, `&&`, means that both statements must be true if the `print "You're in the prime of your life\n"` code is to be executed - if either one of the statements is not true for some reason, "You're not in the prime of your life" is printed out instead.

As well as `&&`, there is also `||` (the pipe symbol printed twice) which means "OR". In this situation, if any of

the conditions being checked in an if statement is true, the whole statement is considered true.

You have a vast array of ways to check statements, of which we have just looked at < (less than) and <= (less than or equal to), and >= (greater than or equal to). We will be looking at the complete list later, but first I want to mention one key check: ==, or two equals signs put together. That means, "is equal to". Therefore 1 == 1 is true, and 1 == 2 is false.

Did you notice that if statements use the code blocks I mentioned earlier? The code to be executed if the statement is true is in its own block (remember a block starts with { and finishes with }), and the code to be executed otherwise is in an else block. This stops PHP from trying to execute both the true and false actions.

One key thing to note is that PHP practises "if statement short-circuiting" - this is where PHP will try to do as little conditional work as possible, so it stops checking conditional statements as soon as it knows the result for sure. For example:

```
if ($Age > 10 && $Age < 20)
```

If \$Age is 8, the first check (\$Age > 10) will fail, so PHP will not even bother checking it against twenty. This means you can, for example, check whether a variable is set and whether it is set to a certain value - if the variable is not set, PHP will short-circuit the if statement and not check its value, which is good because if you check the value of an unset variable, PHP will flag an error!

A helpful addition to if statements is the elseif statement, which allows you to chain commands together in a slightly more intelligent way. Here is an example script:

```
<?php
if ($Age < 10) {
    print "You're under 10";
} elseif ($Age < 20) {
    print "You're under 20";
} elseif ($Age < 30) {
    print "You're under 30";
} elseif ($Age < 40) {
    print "You're under 40";
} else {
    print "You're over 40";
}
?>
```

You could reach the same effect by having lots of if statements, however this is a slightly neater way of doing things. The downside of this system is that the \$Age variable needs to be checked repeatedly.

If you only have one line of code to execute, you can do without the braces entirely. Given that there is no speed difference between using braces and not using braces, it's usually down to a readability issue.

So, these two code chunks are the same:

```
if ($foo) {  
    echo $bar;  
}  
  
if ($foo) echo $bar;
```

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Case switching >>](#)

Previous chapter: [Comments](#)

Jump to:      Conditional statements

Home: [Table of Contents](#)



Access All Your Email  
Accounts Right from  
Your Browser

Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# Case switching

Consider this piece of code:

```
<?php
$Name = "Bob";
if ($Name == "Jim") {
    print "Your name is Jim\n";
} else {
    if ($Name == "Linda") {
        print "Your name is Linda\n";
    } else {
        if ($Name == "Bob") {
            print "Your name is Bob\n";
        } else {
            if ($Name == "Sally") {
                print "Your name is Sally\n";
            } else {
                print "I do not know your name!\n";
            }
        }
    }
}
?>
```

As you can see, here we're just trying to find out which piece of code we should execute, however it requires a lot of code because of the way if statements work. PHP has a solution to this, and it is called switch/case statements. In a switch/case block you specify what you are checking against, then give a list of possible values you want to handle. Using switch/case statements, we can rewrite the previous mess of if statements like this:

```
<?php
$Name = 'Bob';
```

```

switch($Name) {
    case "Jim": print "Your name is Jim\n"; break;
    case "Linda": print "Your name is Linda\n"; break;
    case "Bob": print "Your name is Bob\n"; break;
    case "Sally": print "Your name is Sally\n"; break;
    default: print "I do not know your name!\n";
}
?>

```

Switch/case statements are used to check all sorts of data, and as you can see they take up much less room than equivalent if statements.

There are two important things to note in the PHP switch/case statement code. Firstly, there is no word "case" before "default" - that's just how the language works.

Secondly, each of our case actions end with "break;". This is because once PHP finds a match in its case list, it will execute the action of that match as well as the actions of all matches beneath it (further down on your screen). This might not seem very helpful at first, but there are many situations where it comes in useful - not the least of which is trying to program a script to print out the song "The 12 Days of Christmas"!

The keyword "break" means "get out of the switch/case statement", and has the effect of stopping PHP from executing the actions of all subsequent cases after its match. Without the break, our test script would print out this:

```

Your name is Bob
Your name is Sally
I do not know your name

```

Once PHP matches Bob, it will execute Bob's action, then Sally's, then the default also. It is generally best to use "break" unless you specifically want to have the fall-through behaviour.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Loops >>](#)

Previous chapter: [Conditional statements](#)

Jump to: Case switching

Home: [Table of Contents](#)

Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



**INTERACTION PROTOTYPING**  
RUN PROTOTYPES ON ANY HTML5 ENABLED DEVICE, **INCLUDES IOS PACK**

**FREE DOWNLOAD** 

# Loops

There are four ways to make code loop in PHP so that you can perform repetitive actions, and each work in different ways. In practice, only three of the four are used regularly, but you need to at least be aware of all four.

The first loop type is called a while loop, and can be thought of as an if statement that is evaluated repeatedly until it fails. In pseudocode it looks like this:

```
while (condition is true) {
    do code
}
```

Notice that again PHP uses code blocks to represent the extent of our loop - while loops start with an opening brace { and finish with a close brace } to tell PHP clearly which lines of code should be looped through.

Like if statements, you can put whatever conditions you like into while loops, however it's crucial that you change the value of the condition with each loop. Consider this piece of code:

```
<?php
    while (1 == 1) {
        // do stuff
    }
?>
```

1 is always, *always* going to be equal to 1, which means this while loop will just whizz around in circles perpetually. This is called an infinite loop - a loop that will not exit because its condition will always be met - and they tend to drag your computer to a halt very quickly. Having said that, infinite loops can be useful -

we'll look at those situations soon.

"While" loops are most often used to iterate over a list where there is no known limit to the number of iterations of the loop. For example:

```
<?php
    while(there are still rows to read from a database) {
        read in row;
        move to the next to row;
    }
?>
```

A more common form of loop is the "for" loop, and is slightly more complicated. A for loop has three parts - a declaration, a condition, and an action - as well as a loop counter variable that tracks the number of times the loop code has been executed (the number of iterations). The declaration is where the loop counter variable is declared and set to a starting value, the condition is where the loop variable counter is checked against a value, and the action is the action to take at the end of each iteration to change the loop counter.

Here is how a for loop looks in PHP:

```
<?php
    for ($i = 1; $i < 10; $i = $i + 1) {
        print "Number $i\n";
    }
?>
```

As you can see, the for loop has the three distinct parts separated by a semi-colon each. Firstly, for our declaration, we set the variable `$i` to 1. Our condition, the second of the three parts in the for loop, is that the loop will execute if `$i` is less than 10. Finally, the action is that we add 1 to the value of `$i` every loop iteration - that is, every time the loop code is executed. So, a for loop is made up of a declaration, a condition, and an action.

Therefore, what this script does is to count from 1 to 10, outputting text along the way. Note that it will not actually output "Number 10" because we specify that `$i` must be *less* than 10, not less than or equal to it.

Here is the output:

```
Number 1
Number 2
Number 3
Number 4
Number 5
```

```
Number 6
Number 7
Number 8
Number 9
```

The \n part at the end of the string is what causes the new lines, so if you're using Windows you'll need to use \r\n. This is all explained later - don't worry about it too much for now.

The third and least important loop type takes the form do...while, and is similar to a while loop with the difference that it is always executed at least once. Consider the following piece of code:

```
<?php
$foo = 12;
while ($foo < 10) {
    do_bar();
}
?>
```

In that situation, the `do_bar()` function will never be called, because `$foo` will not be less than 10. However, in a do...while loop, we have this:

```
<?php
$foo = 12;
do {
    do_bar();
} while ($foo < 10);
?>
```

In the latter case, `do_bar()` will be executed before `$foo` is compared against 10. If `$foo` is less than 10 when checked, the loop executes again. As you can see, the only difference between a while loop and a do...while loop is that the latter is always executed a minimum of once.

The last type of loop is the foreach loop, and is used to loop through an array of data. As arrays are quite unique in how they work, this particular type of loop is covered specifically in the Arrays chapter.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a

downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

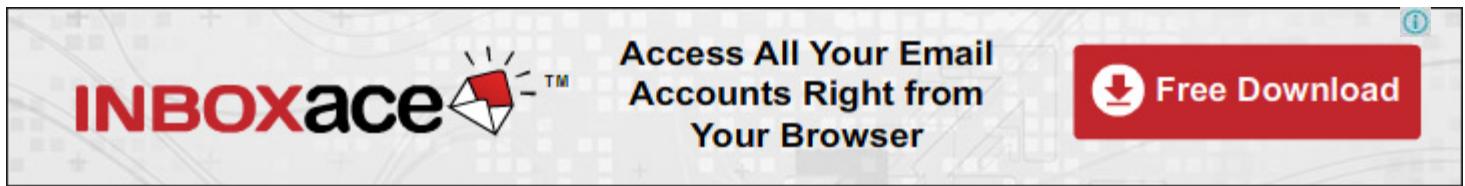
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Infinite loops >>](#)

Previous chapter: [Case switching](#)

Jump to:      [Loops](#)

Home: [Table of Contents](#)

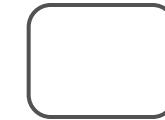


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

## Viêm Amidan Nguy Hiểm Ko?

Bạn Biết Nhũng Biển Chứng Amidan?. Gặp Chuyên Gia Tư Vấn Miễn Phí



# Infinite loops

Perhaps surprisingly, infinite loops can sometimes be helpful in your scripts. As infinite loops never terminate without outside influence, the most popular way to use them is to break out of the loop and/or exit the script entirely from within the loop whenever a condition is matched. You can also rely on user input to terminate the loop - for example, if you are writing a program to accept people typing in data for as long as they want, it just would not work to have the script loop 30,000 times or even 300,000,000 times. Instead, the code should loop forever, constantly accepting user input until the user ends the program by pressing Ctrl-C.

While you are learning PHP, you should steer clear of infinite loops as they can cause problems in the first few weeks. After that, try playing around with them to see how they can actually help you make your scripts better. Having said that, you will almost certainly make a few loops carry on forever simply by accident!

If you want to try them out, here are the most common examples of infinite loops:

```
<?php
    while(1) {
        print "In loop!\n";
    }
?>
```

As "1" also evaluates to true, that loop will continue on forever. Many people also like to write their infinite loops like this:

```
<?php
    for (;;) {
        print "In loop!\n";
    }
?>
```

In that example, the for loop is missing the declaration, condition, and action parts, meaning that it will always loop.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Special loop keywords >>](#)

Previous chapter: [Loops](#)

Jump to: [Infinite loops](#)

Home: [Table of Contents](#)



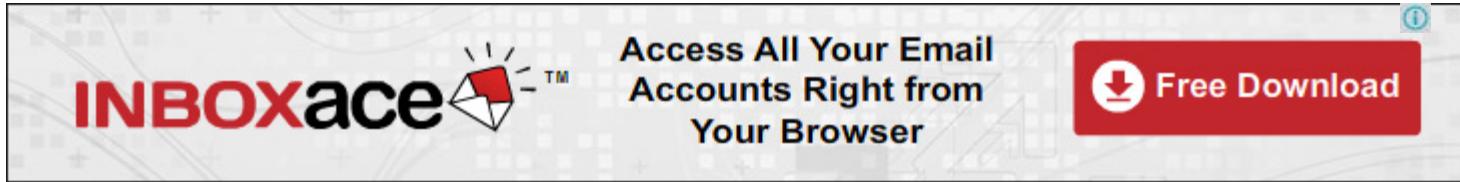
**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

[!\[\]\(0d726af61dd3aaf779535a25c58a0c5f\_img.jpg\) Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Special loop keywords

There are two special keywords you can use with loops, and they are "break" and "continue". We already used break previously when we looked at case switching - it is used there to exit a switch/case block, and it has the same effect with loops. When used inside loops in order to manipulate the loop behaviour: break causes PHP to exit the loop and carry on immediately after it, and continue causes PHP to skip the rest of the current loop iteration and go onto the next.

For example:

```
<?php
for ($i = 1; $i < 10; $i = $i + 1) {
    if ($i == 3) continue;
    if ($i == 7) break;
    print "Number $i\n";
}
?>
```

That is a modified version of our original for loop script. This time the output is this:

```
Number 1
Number 2
Number 4
Number 5
Number 6
```

Note that number 3 is missed out entirely, and the script exits after number 6. The reason for this is because of the two if statements - if the current number is 3, "continue" is used to skip the rest of that iteration and go

on to number 4. Also, if the number is 7, "break" is used to exit the loop altogether.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

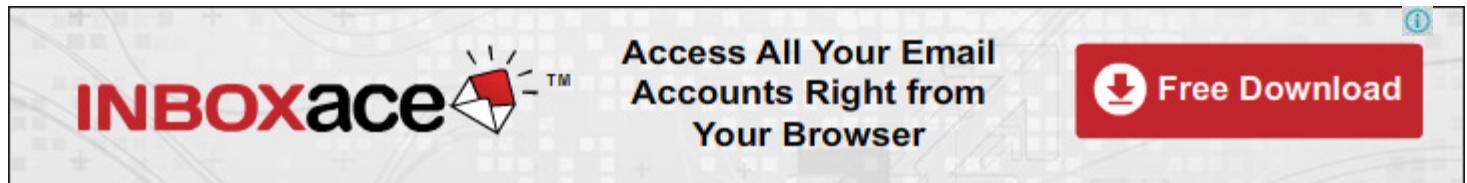
[Tweet](#)

Next chapter: [Loops within loops >>](#)

Previous chapter: [Infinite loops](#)

Jump to: [Special loop keywords](#)

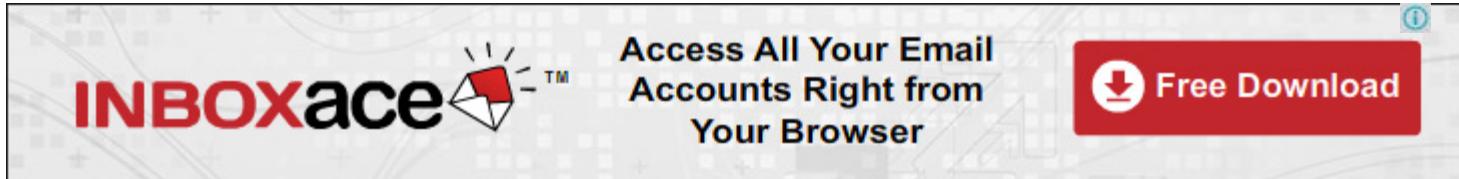
Home: [Table of Contents](#)



The banner features the INBOXace logo with a red envelope icon and the text "Access All Your Email Accounts Right from Your Browser". A red button on the right says "Free Download" with a download icon. There is also a small "i" icon in the top right corner of the banner area.

Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Loops within loops

You can stack loops up as you see fit simply by having one loop inside the body of another, like this:

```
for ($i = 1; $i < 3; $i = $i + 1) {
    for ($j = 1; $j < 4; $j = $j + 1) {
        for ($k = 1; $k < 3; $k = $k + 1) {
            print "I: $i, J: $j, K: $k\n";
        }
    }
}
```

The output of that script is:

```
I: 1, J: 1, K: 1
I: 1, J: 1, K: 2
I: 1, J: 2, K: 1
I: 1, J: 2, K: 2
I: 2, J: 1, K: 1
I: 2, J: 1, K: 2
I: 2, J: 2, K: 1
I: 2, J: 2, K: 2
```

In this situation, using break is a little more complicated, as it only exits the closest loop. For example:

```
for ($i = 1; $i < 3; $i = $i + 1) {
    for ($j = 1; $j < 3; $j = $j + 1) {
        for ($k = 1; $k < 3; $k = $k + 1) {
            print "I: $i, J: $j, K: $k\n";
        }
    }
}
```

```
        break;  
    }  
}
```

This time the script will print out the following

I: 1, J: 1, K: 1  
I: 1, J: 2, K: 1  
I: 2, J: 1, K: 1  
I: 2, J: 2, K: 1

As you can see, the `$k` loop only loops once because of the break call. However, the other loops execute several times. You can exercise even more control by specifying a number after break, such as "break 2" to break out of two loops or switch/case statements. For example:

```
for ($i = 1; $i < 3; $i = $i + 1) {  
    for ($j = 1; $j < 4; $j = $j + 1) {  
        for ($k = 1; $k < 3; $k = $k + 1) {  
            print "I: $i, J: $j, K: $k\n";  
            break 2;  
        }  
    }  
}
```

That outputs the following:

I: 2, J: 1, K: 1

This time the loop only executes twice, because the `$k` loop calls "break 2", which breaks out of the `$k` loop and out of the `$j` loop, so only the `$i` loop will go around again. This could even be "break 3", meaning break out of all three loops and continue normally.

It is important to remember that "break" works both on loops and switch/case statements. For example:

```
for ($i = 1; $i < 3; $i = $i + 1) {  
    for ($j = 1; $j < 4; $j = $j + 1) {  
        for ($k = 1; $k < 3; $k = $k + 1) {  
            switch($k) {
```

```
    case 1:
        print "I: $i, J: $j, K: $k\n";
        break 2;
    case 2:
        print "I: $i, J: $j, K: $k\n";
        break 3;
    }
}
}
```

The "break 2" line will break out of the switch/case block and also out of the `$k` loop, whereas the "break 3" line will break out of those two and also the `$j` loop. To break out of the loops entirely from within the switch/case statement, "break 4" is required.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Mixed-mode processing >>](#)

Previous chapter: [Special loop keywords](#)

Jump to: Loops within loops

Home: [Table of Contents](#)





Access All Your Email  
Accounts Right from  
Your Browser



Free Download

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Mixed-mode processing

A key concept in PHP is that you can toggle PHP parsing mode whenever you want and as often as you want, and you can even do it inside a code block. Here is a basic PHP script:

```
<?php
    if ($foo == $bar) {
        print "Lots of stuff here";
        print "Lots of stuff here";
        print "Lots of stuff here";
        ...[snip]...
        print "Lots of stuff here";
        print "Lots of stuff here";
    }
?>
```

As you can see, there are a lot of print statements that will only be executed if the variable `$foo` is the same as variable `$bar`. All the output is encapsulated into print statements, but PHP allows you to exit the PHP code island while still keeping the if statement code block open - here's how that looks.

```
<?php
    if ($foo == $bar) {
?>
    Lots of stuff here
    Lots of stuff here
    Lots of stuff here
    ...[snip]...
    Lots of stuff here
    Lots of stuff here
<?php
    }
?>
```

The "Lots of stuff here" lines are still only sent to output if `$foo` is equal to `$bar`, but we exit PHP mode to print it out. We then re-enter PHP mode to close the if statement, and continue - it makes the whole script easier to read.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

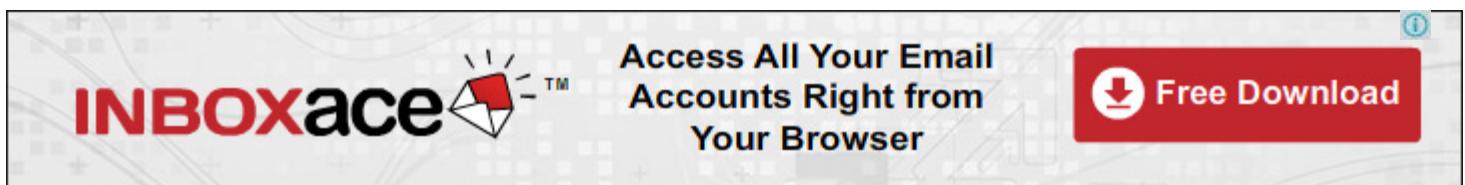
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Including other files >>](#)

Previous chapter: [Loops within loops](#)

Jump to:      Mixed-mode processing

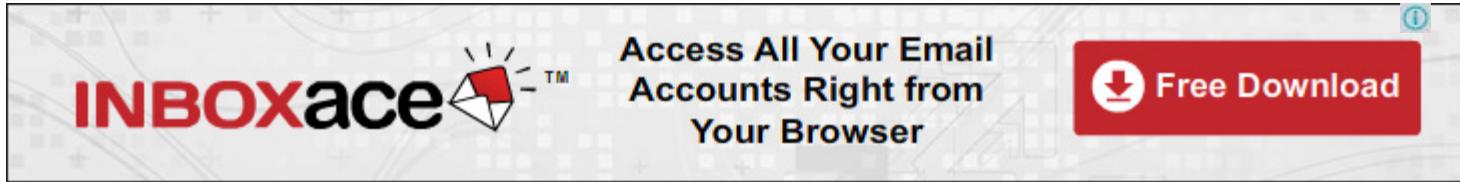
Home: [Table of Contents](#)



The banner features the INBOXace logo on the left, which includes the word 'INBOX' in red and 'ace' in black with a small envelope icon. To the right of the logo is the text 'Access All Your Email Accounts Right from Your Browser'. On the far right is a red button with a white download icon and the text 'Free Download'.

Copyright ©2015 Paul Hudson. Follow me: [@twostraws](#).

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Including other files

One of the most basic operations in PHP is including one script from another, thereby sharing functionality. This is done by using the include keyword, and specifying the filename you want to include.

For example, consider the following file, foo.php:

```
<?php
    print 'Starting foo\n';
    include 'bar.php';
    print 'Finishing foo\n';
?>
```

And also the file bar.php:

```
<?php
    print 'In bar\n';
?>
```

PHP would load the file bar.php, read in its contents, then put it into foo.php in place of the "include 'bar.php'" line. Therefore, foo.php would look like this:

```
<?php
    print 'Starting foo\n';
    print 'In bar\n';
    print 'Finishing foo\n';
?>
```

If you were wondering why it only writes in the "In bar" line and not the opening and closing tags, it's

because whenever PHP includes another file, it drops out of PHP mode, then re-enters PHP mode as soon as it comes back from the file. Therefore, foo.php, once merged with bar.php, will actually look like this:

```
<?php
    print 'Starting foo\n';
?>
<?php
    print 'In bar\n';
?>
<?php
    print 'Finishing foo\n';
?>
```

It has the same effect, but it is a little harder to read!

It is important to note that PHP only includes a file if the include line is actually executed. Therefore, the following code would never include bar.php:

```
<?php
if (53 > 99) {
    include 'bar.php';
}
?>
```

If you attempt to include a file that does not exist, PHP will generate a warning message. If you find that you write a script that absolutely needs a particular file to be included, PHP also has the require keyword, which, if called on a file that does not exist, will halt script execution with a fatal error. However, the chances of you telling PHP to include a script and it not being a problem if that script cannot be located are slim, so favour require.

The most common way to use these include files is as storage for common functions, object definitions, and layout code. For example, if your site always has the same header HTML to it, you can start each of your pages with this:

```
<?php
include 'header.php';
...[snip]...
```

That way, whenever you want to change the header of your site, you just need to edit one file. Two more keywords that are likely to be of use are include\_once and require\_once, which operate in the same way as include and require respectively, with the difference that they will only include a file once, even if you try to

include it several times. `include_once` and `require_once` share the same list of "already included" files, but it is important to note that operating systems that are case sensitive, such as Unix, are able to `include_once/require_once` a file more than once if the programmer uses varying cases for their filenames. For example:

```
<?php  
    include_once 'bar.php';  
    include_once 'BAR.php';  
    include_once 'Bar.php';  
?>
```

On Unix, that will attempt to include three entirely different files, because Unix is case sensitive. The solution is simple: use lower case filenames for everything!

Note that if PHP cannot find a file you try to `include()` or `require()` in the same directory as the script that is running, it will also look in its standard include path. This is defined in your `php.ini` file as the `include_path` directive.

**Author's Note:** Each time you include a file using `include()` or `require()`, PHP needs to compile it. If you're using a code cache this problem is avoided, but if not PHP really does compile the same file several times. That is, if you have various includes for the same file, it will need to be compiled and processed each time - it's best to use `include_once()`. Failing that, there are two functions called `get_included_files()` and `get_required_files()` that will tell you the names of the files you have already included - they are actually the same function internally, so you can use either one.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

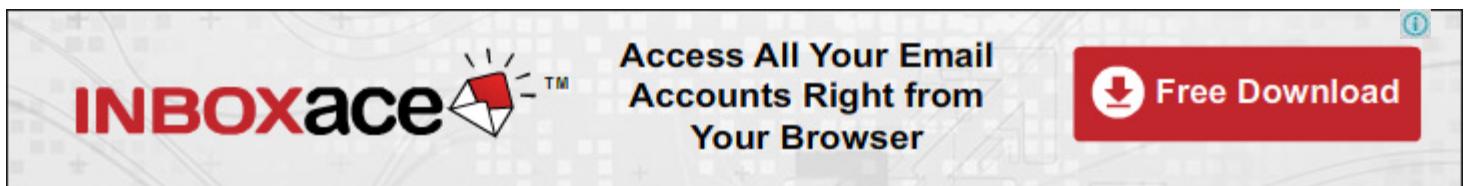
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Abnormal script termination >>](#)

Previous chapter: [Mixed-mode processing](#)

Jump to:     [Including other files](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Abnormal script termination

Although you would obviously rather that your PHP scripts always execute correctly from the start of the script to the end of the script, there are a variety of reasons why they might not. Put succinctly, these reasons are:

1. You've screwed up somewhere, and PHP cannot execute your code.
2. PHP has screwed up somewhere due to a bug, and cannot continue
3. Your script has taken too long to execute, and gets killed by PHP
4. Your script has requested more memory than PHP can allocate, and gets killed by PHP

To be brutally honest, the first situation is unequivocally the most common. This will change a little as your skill with PHP improves, but the first situation is still the most common even amongst the most veteran programmers! There is a large chunk of this book dedicated to debugging PHP code - feel free to skip ahead to start reading there if you're already fluent with the PHP basics.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Editing your PHP configuration >>](#)

Previous chapter: [Including other files](#)

Jump to: Abnormal script termination

Home: [Table of Contents](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Editing your PHP configuration

PHP gets its options from a file `php.ini` on your hard drive. If this does not exist, PHP assumes a default set of values, so it is best to set up a `php.ini` file to make sure you have PHP working as you want.

If you do not have access to your `php.ini` file, perhaps because you host your site elsewhere, you will need to hand set your PHP settings in each script, although there are limitations on how much you can change.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

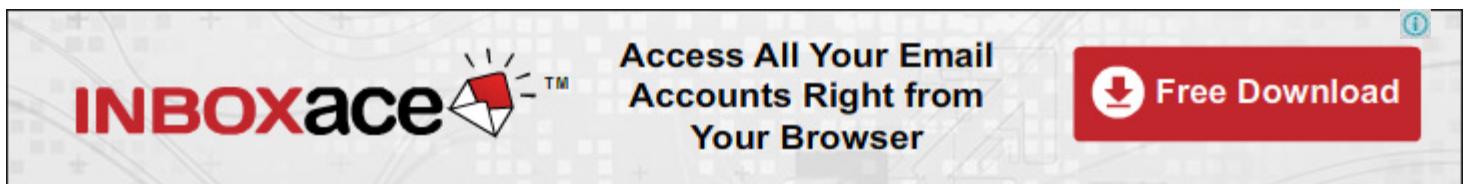
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Summary >>](#)

Previous chapter: [Abnormal script termination](#)

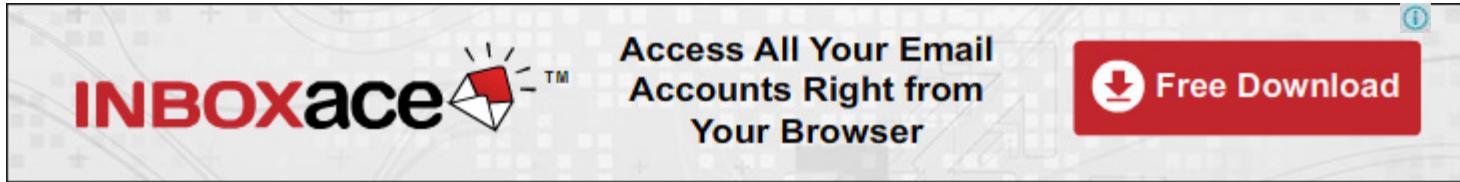
Jump to: Editing your PHP configuration

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Summary

- PHP is a powerful, modern, and flexible programming language well-suited to internet tasks and a variety of other environments
- But that does not always make it the best choice - you should always carefully consider your choice of language before starting a project
- There are a variety of real business benefits to using PHP - don't let others fob you off with poor excuses such as "it's free - it cannot be good!"

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Exercises >>](#)

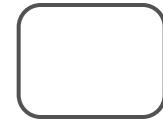
Previous chapter: [Editing your PHP configuration](#)

Jump to: Summary

Home: [Table of Contents](#)

# Get our free BOD Brochure

How to setup your own Biochemical Oxygen Demand determination process



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>

# Exercises

1. Which of these statements is true about PHP 5?

- a) Require and include are the same
- b) Require includes files whether or not the line is executed
- c) Require exits the script if it cannot find the file to include
- d) None of the above

2. Which of the following is not a valid way to start a PHP comment?

- a) //
- b) ##
- c) /\*
- d) None of the above

3. PEAR is...

- a) A tasty fruit
- b) The PHP Extended Assignment Repository
- c) The PHP Extension and Application Repository

4. Heredoc syntax is...

- a) A way to define strings without worrying about quotation marks
- b) An automatic documentation tool for PHP
- c) The opposite of the heredoc syntax

5. Mixed-mode processing is...

- a) The act of outputting HTML and pictures at the same time
- b) The act of entering and exiting PHP processing mode several times
- c) Impossible in PHP

6. Using switch/case rather than multiple if statements is...

- a) Faster
- b) Easier to read
- c) Both a) and b)
- d) None of the above

7. Escape sequences are the fastest way to exit your script: true or false?

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Further reading >>](#)

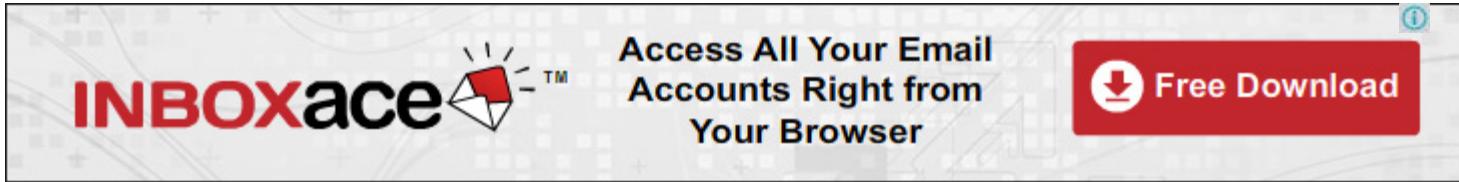
Previous chapter: [Summary](#)

Jump to: [Exercises](#)

Home: [Table of Contents](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



The banner features the INBOXace logo with a red envelope icon and the text "Access All Your Email Accounts Right from Your Browser". A red button on the right says "Free Download" with a download icon.

# Further reading

- Microsoft has quite a lot of literature geared towards getting people to move from PHP to its proprietary ASP.NET system. Unlike some of the Microsoft literature to its more hated enemies, such as Linux, the ASP.NET migration information is actually fairly well thought out and makes interesting reading.

If you are currently an ASP.NET developer and are considering PHP, give this document a read - it will help you fully understand the differences between PHP and ASP.NET. Having said that, remember it is written by Microsoft about a language they obviously want to promote - take the facts presented with a pinch of salt unless you can verify them elsewhere.

Due to the ever-shifting nature of MSDN articles, it is hard for me to provide a URL that will work for more than a month. Running a Google search for "php msdn" should turn it all up.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

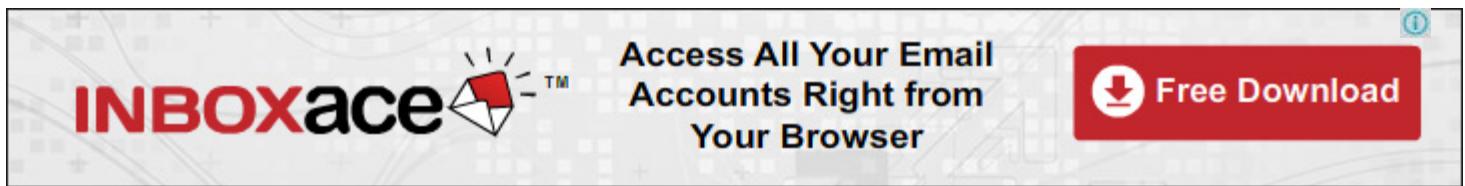
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Next chapter >>](#)

Previous chapter: [Exercises](#)

Jump to: [Further reading](#)

Home: [Table of Contents](#)

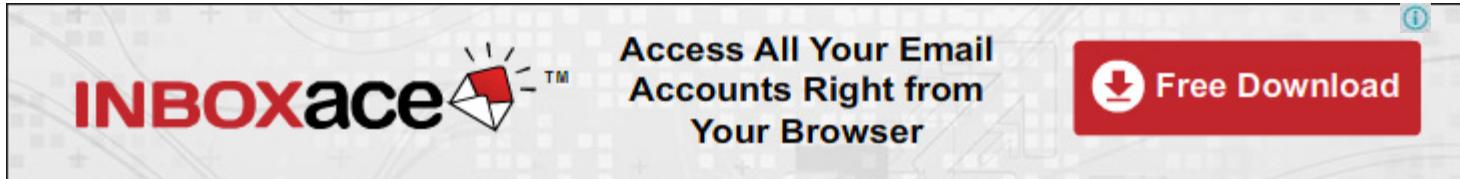


Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP

Next chapter

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Next chapter

Now that you have a basic grasp of what makes PHP tick and why you should want to use it in the first place, it is time to get down to the serious task of learning how to program in the language. The next chapter will teach you much of what you need to know about variables and operators, and most of it is pretty crucial information for you to remember - take a break now if you need to!

Want to learn PHP 7?

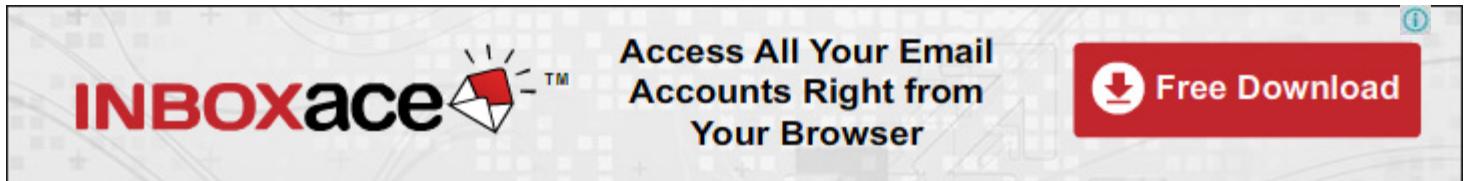
Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Simple variables and operators >>](#)

Previous chapter: [Further reading](#)

Jump to: [Next chapter](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been updated for PHP 7 - only \$20! >>



# Types of Data

PHP has seven types of variables, and all but one hold a specific class of information. The seven types are: strings, integers, floats, booleans, arrays, objects, and resources. You'll be using them all throughout this book, so it is worthwhile remembering what each do.

Strings hold characters (literally: a string of characters) such as "a", "abc", "Jack and Jill went up the hill to fetch a pail of water", etc. Strings can be as short or as long as you want - there's no limit to size.

Integers hold whole numbers, either positive or negative, such as 1, -20, 55028932, etc. There is a maximum limit to the size of integers - any numbers lower than -2147483647 and any numbers higher than 2147483647 are automatically converted to floats, which can hold a much larger range of values.

Floats hold fractional numbers as well as very high integer numbers, such as 4.2, 1.00000001, and 2147483647000.

Booleans simply hold true or false. Booleans are, in fact, just integers behind the scenes - PHP considers the number 0 to be false, and everything else to be true.

Arrays are a special variable type in that they hold multiple values. Arrays can be quite complicated, and so are covered in detail in their own chapter.

Objects are complex variables that have multiple values, but also their own functions. Objects are also very complicated, and, like arrays, are covered in their own chapter.

Resources are anything that is not PHP data - this might be picture data you have loaded from a file, the result of an SQL query, etc. Resources are used like any other variable, with the key difference that you should generally remember to free up your resources when you are finished with them.

Most data types are freely convertible to most other data types, which makes PHP *loosely typed*. Here's a piece of code that should illustrate the point nicely:

```
<?php
$mystring = "12";
$myinteger = 20;
print $mystring + $myinteger;
?>
```

That script will output 32, despite the fact that the first variable, `$mystring`, is a string, whereas the other is an integer. PHP will automatically attempt to convert the non-integer operand, `$mystring`, into an integer, and will find that it is in fact an integer inside a string. If it tries to convert a string like "wombat" to an integer, PHP will simply return the value 0.

**Author's Note:** "Operand" is one of those terms you learn at school and forget the minute you leave. An operand is something, in this case `$mystring`, upon which a mathematical function, in this case `+`, is being performed. So in `$mystring + $myinteger`, `+` is the operator, `$mystring` and `$myinteger` are the operands.

Strings have one special usage that set them apart a little bit from the other variable types, and that is `{x}` notation. As a string is just a collection of characters, it is sometimes possible that you may want to read or write just one character. Take a look at this example:

```
<?php
$mystr = "Jello, world?";
$mystr{0} = "H";
$mystr{12} = "!";
print $mystr;
?>
```

Starting off with a broken string, we change the first character (letter 0) to H, then the twelfth character to an exclamation mark, forming "Hello, world!". Later on we'll look at a function, `strlen()`, that returns the size of a string - you can use this instead of numbers so that rather than 12 we could use this:

```
$mystr{strlen($mystr)-1} = "!";
```

The `-1` is necessary because `strlen()` returns the length of the string, which is always one more than the position of the last digit of the string in PHP because the first character starts at 0. That is, a string of length 13, as above, will have its last character at position 12. More on that later.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Checking a variable is set >>](#)

Previous chapter: [Simple variables and operators](#)

Jump to: [Types of Data](#)

Home: [Table of Contents](#)



**INBOXace** TM

Access All Your Email  
Accounts Right from  
Your Browser

[!\[\]\(766a34838bef958d7c81883ac7ce7fb3\_img.jpg\) Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>

# Checking a variable is set

Although the majority of functions are covered in the Functions chapter, you really need to get to grips with the *isset()* function (literally "is a variable set?") in order to make the most of this chapter. To make it work, you pass a variable in as the only parameter to *isset()*, and it will return true or false depending on whether the variable is set. For example:

```
<?php
$foo = 1;
if (isset($foo)) {
    echo "Foo is set\n";
} else {
    echo "Foo is not set\n";
}
if (isset($bar)) {
    echo "Bar is set\n";
} else {
    echo "Bar is not set\n";
}
?>
```

That will output "Foo is set" and "Bar is not set". Usually if you try to access a variable that isn't set, like *\$bar* above, PHP will issue a warning that you are trying to use an unknown variable. This does not happen with *isset()*, which makes it a safe function to use.

As I said, flip ahead to the Functions chapter to read more about this function; for now, you just need to know this one.

Note that throughout this book (and throughout the programming world), "nonsense" variable names are used. Names like *\$foo*, *\$bar*, and *\$baz* are the most common, but I often add *\$wom* and *\$bat*. Some

argue against the use of these names, but the reality is that their use is so widespread - and that they can help liven up otherwise dull texts - and so I've used them liberally. That said, avoid them in your own code unless it's for short-lived variables.

### Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Automatic type conversion >>](#)

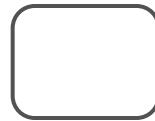
Previous chapter: [Types of Data](#)

Jump to: [Checking a variable is set](#)

Home: [Table of Contents](#)

## A world without privacy

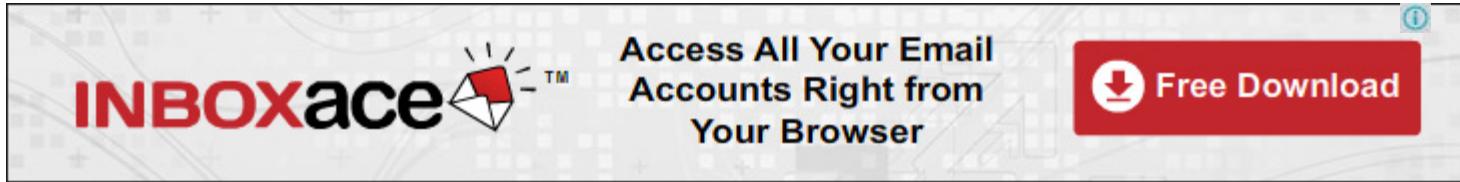
US debates a world without privacy, arguments of law agencies



Copyright ©2015 Paul Hudson. [Follow me: @twostraws](#).



Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Automatic type conversion

As PHP is loosely typed, it will automatically convert one type to another whenever possible. Problems with automatic conversion occur when either no meaningful conversion is possible, or when conversion yields unexpected results. For example, calling "print" on an array makes PHP print out "Array"; it doesn't automatically convert the array to a string of all its elements. Treating an object like a string has its own unique behaviour, but that's not too important right now.

The unexpected results occur when PHP converts values and produces unhelpful results. This is not PHP being badly written, more that your code needs to be more explicit. For example, converting from a boolean to a string will produce a 1 if the boolean is set to true, or an empty string if false. Consider this script:

```
<?php
$bool = true;
print "Bool is set to $bool\n";
$bool = false;
print "Bool is set to $bool\n";
?>
```

That will output the following:

```
Bool is set to 1
Bool is set to
```

As you can see, it hasn't printed out a 0 for false. Instead, nothing is printed, which makes the output look incorrect. To solve this problem, and others like it, tell PHP how you want the value converted by typecasting. The above script should be rewritten to typecast the boolean to an integer, as this will force boolean true to be 1 and boolean false to be 0.

```
<?php  
    $bool = true;  
    print "Bool is set to $bool\n";  
    $bool = false;  
    print "Bool is set to ";  
    print (int)$bool;  
?>
```

This time the script outputs 1 and 0 as we wanted.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Forcing a type with type casting >>](#)

Previous chapter: [Checking a variable is set](#)

Jump to: [Automatic type conversion](#)

Home: [Table of Contents](#)



Access All Your Email  
Accounts Right from  
Your Browser

 [Free Download](#)

Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)

Hacking with PHP has been [updated for PHP 7](#) - only \$20! >>



# Forcing a type with type casting

PHP will automatically convert data types as necessary across the board - you need not worry about it happening. If you specifically wish to override PHP's type conversion, you can perform what is called a *type cast* - you forcibly convert a variable of type A to type B. In PHP, type casting looks like this:

```
<?php
  $mystring = "wombat";
  $myinteger = (integer)$mystring
?>
```

At first, `$mystring` contains a string. However, we type cast it to be an integer, so PHP will convert it to an integer, and place the result into `$myinteger`. You can type cast as boolean using (bool), string using (string), and floating-point using (float).

Type casting is most often used to specifically enforce a type in order to provide extra security or just to make sure a set type of data is being used. For example, if your script absolutely requires an integer number, it's a smart move to typecast your variable with (integer) so that PHP will convert any other type to integer or do nothing if the type is already integer. Converting a float to an integer will automatically round the number down, and is actually faster than using the equivalent function.

Want to learn PHP 7?

Hacking with PHP has been fully updated for PHP 7, and is now available as a

downloadable PDF. Get over 1200 pages of hands-on PHP learning today!

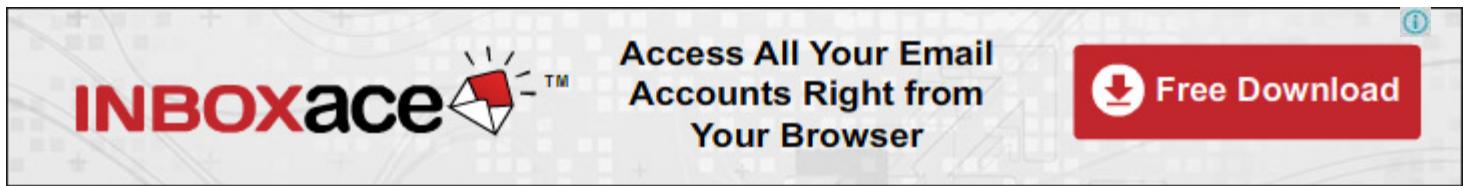
If this was helpful, please take a moment to tell others about Hacking with PHP by tweeting about it!

Next chapter: [Non-decimal number systems >>](#)

Previous chapter: [Automatic type conversion](#)

Jump to: [Forcing a type with type casting](#)

Home: [Table of Contents](#)



Copyright ©2015 Paul Hudson. [Follow me: @twostraws.](#)