The No Bullshit Bible : Creating Web 2.0 Startups & Programming

Written by James Gillmore
Edited by Holly Welch

FaceySpacey
www.faceyspacey.com

PHP

TABLE OF CONTENTS

Technical

Chapter 1

1-1 PHP | DYNAMIC CODE PATHS & VARIABLES

This tutorial on PHP should be treated as a tutorial about learning the basics of all programming. It's goal is to be an eye-opener into what programming is about.

The reason we think PHP is the language to start learning programming through for the web developers is two-fold:

1) it's syntatically similar to Javascript, which all web developers will have to learn, since it's the language of the browser (note: HTML works in conjunction with javascript in the browser, where HTML deals with simple display & presentation, while javascript does the heavy lifting necessary for animations, and ajax calls to your server that don't require a page refresh). PHP is also syntactically similar to C/C++ which is basically the grandfather of all modern procedural & object oriented programming.

2) it's the easiest language to setup on a server and get productive with, as it's the most prevalent web development server side language. That also means it has the most resources to learn it and all its tools.

So back to the tutorial. I like to start be introducing if/else statements since it does a very good job at highlighting how programming is really all about basic logic you already know. Code example time:

```
$variable = 2;

if ($variable == 2) {
    echo 'the variable equals 2';
} else {
    echo 'this code path will not get executed currently';

}
```

So that code if executed will always echo out "the variable equals 2" to the browser. That's fine. Quick note: a variable in PHP is any word that starts with a *$* symbol.

Simply a variable is a placeholder that at different times can hold different values. If you don't fully understand variables yet, just look out for any*$words* like that--you'll understand soon. So lets break down a few important things going on here:

1) The following is the assignment of the value 2 to a variable called $variable:

```
$variable = 2;
```

2) The following tests whether $variable equals 2 (the double equal symbol here is the difference between this comparison and the assignment in #1 above):

```
if ($variable == 2)
```

3) The following code between the brackets delineates 2 separate code paths. The code within the brackets is obviously what gets executed depending on which path is taken. What path is taken in this example is the first path since the comparison resolves to true. Otherwise the "else" path would be taken:

```
} else {
   echo 'this code path will not get executed currently';

}
```

So let's make this a bit more "dynamic," which as you may recall means changing. Let's try to make it so it's possible for both of the 2 code paths to be taken:

```
$dayOfMonth = date('j');

if ($dayOfMonth  < 15 ) {
    echo "it's the first half of the month";
} else {
    echo "it's the second half of the month";

}
```

So clearly this will execute one code path the first half of the month and the second code the second half of the month. Don't worry about *date('j')* just yet. That's obviously finding the day of the month. It's called a function. We'll get to it later in detail. They're actually quite simple to grasp. Simply you provide it some input in the parentheses and it gives you a response, which in this case is assigned to the variable *$dayOfMonth*. Basically the date() function is connected to your server's clock and knows the date and time, and in this case is returning the number for the current day of the month.

So to wrap up your first PHP and programming in general tutorial, i'd like to simply point out the most important thing that you learned: languages like PHP allow you to dynamically determine to different code paths, i.e. different results to display to the end user. That's the main goal. That's the main birds-eye-view concept to really get a hold of before moving on.

1-2 PHP | FUNCTIONS

So if you read the PHP 3 - Functions tutorial, you got a hint at what a function is. If you took algebra in school, you've already done a lot of work with functions. A function is all about input and output, which you learned about in the Introduction to Programming tutorial. So it allows you to provide some input, and depending on the input allows you to get returned a different output. In order to not introduce more things than necessary, let's go back to the *date()* function.

As you may recall, we provided *date()* with the *'j'* parameter like this: *date('j')*. In this example, 'j' is called a parameter, sometimes referred to as an argument. There is a slight difference between the terms "parameter" and "argument," but for brevity let's just stick with "parameter." Basically in this example, the*'j'* means to take the current date and return just the day of the month. If we supplied an *'m'* like this*date('m')*, it

would return the number for the day of the month. Simple as that. Different input led to different output being supplied.

The next thing to learn is that programming languages have built-in functions such as date() which as mentioned before is internally connected to the system clock of your server/computer. There are also user defined functions. To explain this we're going to go back to the code example from the PHP 1 tutorial and turn it into a function like so:

```
function halveMonth($dayOfMonth)
{
    if ($dayOfMonth < 15 ) {
       return "it's the first half of the month";
   } else {
       return "it's the second half of the month";
   }

}

$dom = date('j');
$theDayOfMonth = halveMonth($dom);

echo $theDayOfMonth;
```

So the first thing to notice is that we got the current day of the month outside of the function. Then we passed it as a parameter to the function. The function then took this input and instead of "echoing" the result to the browser, it returned the text string (i.e. "it's the first half of the month"). And then it assigned it to the new variable *$theDayOfMonth*, which subsequently was echoed to the browser.

What this allowed us to do is make a piece of re-usable code in the form of this function. Now we can call *halveMonth($variable)* anywhere we want and not have to re-write all the corresponding if/else code. bingo. Functions allow you to tuck code away for reuse.

The last thing I'd like to point out is how we have 3 different variables that all hold similar data:

```
1) $dom (the variable passed as input to the function)
2) $dayOfMonth (the variable the function deals with)
3) $theDayOfMonth (the variable the output of the function is assigned
to)
```

The names basically all could have been the same, but I made them different so you could simply see that we were dealing with separate areas of the example. What this boils down to is a term called “scope.” Variables in the definition of the function (i.e. the code block that started with “*function halveMonth($dayOfMonth)*”) only pertain to that code block where the function is defined. Think about it like this: code within the function definition is very separate from code outside the function. By outside the function, i mean this code here:

```
$dom = date('j');
$theDayOfMonth = halveMonth($dom);
echo $theDayOfMonth;
```

That typically would be called “client code” in terms of its relationship to *halveMonth()*. Just like in business a client hires a contractor to serve up some results, here the “client code” hires *halveMonth()* to serve up some results. So here in the world of the “client code” the variables are all in the same “scope.” That means if we re-wrote the above code like this:

```
$dom = date('j');
$dom = halveMonth($dom);
echo $dom;
```

$dom will go from having an assigned value of something like “15” to having an assigned value of ‘it’s the second half of the month’. In other words, we overwrote the value of that variable. If you needed to keep the original numerical value and use the output text string at the same time, you’d use 2 different variables like I originally had.

Either way, the point is that here outside the function, variables of the same name will overwrite each other when assigned new values. This is called “scope.” Continue to PHP 3 - Scope to learn more.

1-3 PHP | SCOPE

"Scope" is one of the first gotchas in learning programming. It has to do with when variables you've defined are available. If we re-wrote the function definition (from PHP 3 - Functions) like this:

```
function halveMonth($dom)
{
   if ($dom < 15 ) {...
```

nothing would change. The name of the variables within the function is totally unrelated from the names outside the function, and can be the same names or different names or whatever. They are not related. THE POINT IS THIS: if for example in the function definition, we re-assigned the value of *$dom*, that would not change the value of *$dom* outside the function (unless of course the returned output was re-assigned to *$dom*). So that means this (let’s go back to the original code snippet and modify it a bit to use *$dom* inside and outside the function):

```
function halveMonth($dom)
{
   $dom = 17;
   if ($dom < 15 ) {
       return ‘it’s the first half of the month’’;
   } else {
       return ‘it’s the second half of the month’;
   }

}

$dom = date('j');
$theDayOfMonth = halveMonth($dom);
echo $theDayOfMonth;
```

will lead to *$dom* within the function definition having a value assigned of 17, and outside the function here *$dom = date('j');* the value of dom would be 14 on the fourteenth of the month. So by assigning it 17 within the function definition, we're not overwriting the value of $dom outside it in the "client code," and in this case, "it's the second half of the month" that will be returned and assigned to the *$theDayOfMonth*variable. That illustrates the definition of "scope."

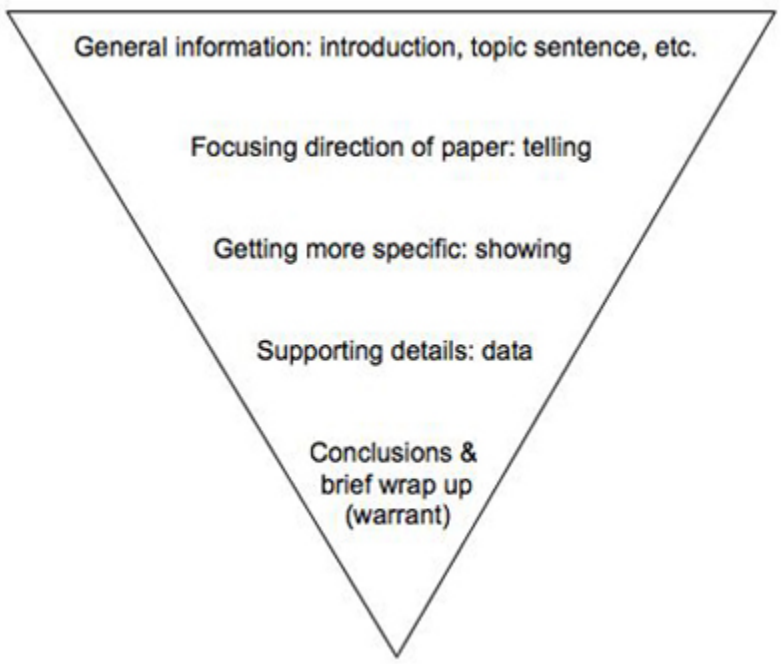
If it's not fully clear yet, it will become clear after a little more study. The important thing to implant in your head and think back to when attempting to grasp this in the future is that scope means variables at different levels of the code can have the same names and are not the same variable and not related at all--though similar names can be used to help programmers reading the code figure out what's going on.

1-4 PHP | HOW TO LEARN PROGRAMMING

If you read PHP 4 - SCOPE you may have noticed i left you with what might seem like a vague description of scope. Accordingly I would like to point out something very important about how I'm writing these tutorials: my main goal in writing these tutorials is to come at an angle of teaching (and learning) that is very different from everything else out there. Specifically, my goal is to teach you the techniques I've learned that help me learn better. So in other words, my goal is not just to teach you programming, but to teach you how to learn programming (and how to learn in general). That will help you learn programming way faster. That said, what I wrote in PHP 3 - FUNCTIONS about "implant in your head..." is a very important technique I have while learning a subject.

Basically the technique goes like this: if there's a concept I don't fully understand, but I know is central to learning the subject matter, I file it away in my brain as

something noteworthy that i'm inevitably going to have to understand, and look out for more lessons and explanations that will further explain it and help me grasp it.



The next bit is I'm teaching programming accordion to the "general to specific" formula of learning. Think about it like a well written college thesis--you introduce your readers to the subject at hand, providing an outline of what will be explained, and gradually take the reader deeper. So rather than hit you with specific after specific whose connection might not be clear, I skim over the subject generally to give you the bigger picture concept you can easily understand. What this does is both inspire you to go deeper, and make it easier to understand the specifics you dive deeper into. After learning the birds-eye-view terrain like this, you'll be hungering to learn more. Teaching specific after specific is what how they teach you in school. They force-feed specific after specific because the assumption is you don't want to learn, and don't really care about the bigger picture, let alone any of it. So traditional teachers will just inject the raw skills, e.g. Calculus, into you, but you barely know what you need it for or how it fits into the bigger picture of math as a whole. So while kids may have no interest in the

bigger picture, the assumption here is that you're hungering for what I'm teaching you, and can take the time and be patient to first learn the birds-eye-view general explanation of the terrain you're about to cover in more detail.

The final bit about this is that I actually gave you that "gotcha" you should file away. Most of the time when reading these tutorials on the web or in books--though written by very capable of authors--they do not give you that nugget, or rather that seed which will grow. And often that means you're left with an unconfident feeling. I find it better to give you something that will make you feel confident about the true baby steps necessary to grasp the subject matter. That's what will lead to you not giving up, and the neurons in your brain forming the proper conclusions on your own, so next time you come back to the subject matter you're stronger. The point is it's a lot easier to understand these general overviews, than the exact specifics. However, once you understand the general overview of the terrain, the specifics become easier. It's just like as if you're working out and work out too hard and pull a muscle, and can't come back for a long time to train again, but if you took the proper baby steps you'd be back at the gym the next day still growing your muscles.

So I'll reiterate, here the seed that has a high likelihood of growing into a tree/muscle is:

"Scope means variables at different levels of the code can have the same names and are not the same variable."

Again, that's not the complete definition, and it's put in very layman's terms, but should give you an idea of how different pockets of code are connected and how they're not. In the case of functions, they're specifically connected in 2 places:

1) on input via the parameter passed in between the parentheses halveMonth*($parameter);*

2) and the output when the result is assigned through an equal sign back to a variable in the current client code scope:

$response = halveMonth($parameter);

1-5 PHP | ARRAYS

So now we're going to get into what I consider the most important thing in order to be productive with programming: loops & arrays. I remember when I was first learning programming, I basically got the concept of loops, but I really didn't get their importance. I certainly didn't understand precisely how important it would be. It wasn't until I got to object oriented programming where I really fell in love with the value of loops and arrays, specifically when looping through objects of the same type, and performing similar actions on the objects. Talking about objects is a little bit too advanced for what I'm trying to convey right now. Just put it into the back of your mind that we will in a later tutorial deal specifically with looping through arrays of objects, and the results will be so extraordinarily useful, and appear in almost every web page you make.

Ok, so before we can get to loops you need to understand arrays. I've always wondered why every explanation I've read of what an array is didn't start by describing the definition of an "array" in plain English unrelated to programming. Either way, that's how I'll start. Without even looking up the precise definition of "array" in a dictionary, I'm going to describe it in a way I think everyone thinks about it, and I'm going to do so with an example: you buy a box of Crayola crayons and dump them all out on a table, you now have an array of crayons to choose from. You have more than one or many crayons. The connotation I usually think of when thinking of "array" is a bunch of items, specifically where they're all not the same.

In programming an array is a list of items. So simply we have a variable called $anything. The idea is that this variable holds a laundry list of items. So rather than *$anything* equaling *1*, it equals a list of values, such as *7, 4, 8, 9, 10*. That only leaves one final thing to understand, which is simply the syntax with which these values are stored under the heading of this single *$anything* variable. In order to demonstrate how multiple values are stored under a single heading, i.e. within a variable of a single name such as *$anything*, I'll provide an example:

```
$anything[0] = 7;
$anything[1] = 4;
$anything[2] = 8;
$anything[3] = 9;
$anything[4] = 10;
```

So that means you can get the value of one of these items like this:

```
echo $anything[3];
```

The result is that 9 is displayed in the web browser. The number *3* in this case is called the "key" and *9*is the "value." So basically you can access any values in the list by using the key. Just while we're at it, I'll point out that this array could have been built like this:

```
$anything[ ] = 7;
$anything[ ] = 4;
$anything[ ] = 8;
$anything[ ] = 9;
$anything[ ] = 10;
```

As a feature of the PHP language/application, it will automatically increment the keys when you build the array like that.

Final note about arrays: the keys can be text strings such as:

```
$anything[‘some_name’] = 5;
echo $anything[‘some_name’];
```

And I’m sure you know what that will do.

1-6 PHP | LOOPS

Understanding loops is the next big hurdle in learning to program. The reason you use them and why they are so prevalent in programming can be a little obscure at first. Let's dive into a quick intro about loops, and a few examples will make all clear.

Firstly, keep in mind the list of values stored in the $anything array from the PHP 6 - ARRAY tutorial. We’ll start with some code:

```
foreach($anything as $k => $v) {
   echo 'the key is ' . $k . ' and the value is ' . $v . '<br>';
}
```

The result in the web browser will be:

```
the key is 0 and the value is 7
the key is 1 and the value is 4
the key is 2 and the value is 8
the key is 3 and the value is 9
the key is 4 and the value is 10
the key is some_name and the value is 5
```

Quick note: the periods around the variables above are how you separate text strings (“literal strings”) from variables, and the *
* string at the end is simply HTML that says put the text strings echoed out to the browser on new lines.

And most importantly, the language structure

```
foreach($anything as $k => $v) {...
```

simply says take the array and iterate through it, aka “walk through it,” aka loop through it, etc. And then simply in the block between the parentheses that follow, *$k* and *$v* will be the variables used to refer to the keys and values of the array. That’s it. That’s the most important thing you need to know about loops in PHP.

Another quite note: if your code went like this:

```
foreach($anything as $v) {...
```

Then you have access to only the values in the parentheses. It’s an extremely common shorthand, used when you don’t need the values of the keys, which usually is the case when the keys are incrementing numbers from 0 up.

The next thing to understand is the *for* loop. In languages like C, there is no concept of a foreach loop. All there is a “for” loop, which is a lot less automated. It doesn’t have intelligence of the array it’s iterating through. In programming terms, they’d say foreach loops are “coupled” with the concept of arrays. That’s not the case with for loops.

```
for ($i = 1; $i <= 10; $i++) {
   echo $i . ‘<br>’;
}
```

So what do you think that will do?

```
1
2
3
4
5
6
7
8
9
10
```

Take a second before I explain what's going on to guess what it's doing...It's simply echoing out the value of the *$i* variable to the browser, which starts as *1*, since *$i=1* is the initial assignment of the *$i* variable as set before the first semicolon. It ends at *10*, because the second statement before the second semicolon says to stop when *$i* reaches *10*. And the 3rd statement says to increment by one after each loop. That's it.

Now let's imagine iterating through our previous array using a for loop:

```
for ($i = 0; $i <= 4; $i++) {
   echo $anything[$i]';
}
```

That will simply alter the keys of the *$anything* array before echoing out the value for that key. It just so happens that the keys of the *$anything* array are ordered numbers from *0* to *4*. If we tried to echo out the value of the *$anything[5]*, nothing would be output since there is no value there. This loop would not be able to echo out the value of the *'some_name'* key like the foreach loop would.

1-7 PHP | CONCLUSION & MORE LEARNING TECHNIQUES

That concludes the basic PHP part of our tutorials. You have just had a birds-eye overview of what is called "procedural programming" in php. Later we'll get to "object oriented programming" in php.

I'd like to point out one last thing regarding for vs. foreach loops, you'll use foreach loops way more often in PHP than you will for loops. The learning technique I would apply now is again to tuck this away for later, but do so like this: keep a look out for a practical problem where a foreach loop won't solve your needs, and when that happens, re-read about for loops, and see if they provide the solution;

try to imagine a for loop at some point in the future providing you with the solution you need. Otherwise, keep it moving, and don't waste much more mind on this.

That's another learning technique I apply: weigh the importance of certain subjects and keep it moving. Often, you'll get your answers in later things you learn, and you'll inefficiently waste time and energy trying to force the understanding of something you may never need, or something that will inevitably become clearer when you learn other aspects. So be OK with not necessarily fully understanding something or being able to see it's practical use.

Some people may say it's important to understand one thing completely before moving to the next. Being determined to learn something is very important. I agree, but with a big BUT. The but is that I usually re-read everything I study at least twice. So you're going to be back here anyway. I'm viewing the real learning process in way more long-term approach, not just a wam bam thank you mam, "read it once and understood it completely," which rarely happens.

Right now, if this is your first time through, you're building a general understanding of the landscape you're tackling. This is what they call the "general to specific" way of learning, as discussed in the PHP 5 - How to Learn Programming tutorial. What I'm trying to do here is get you through the entire landscape of programming as quickly as possible to give you a birds-eye-view. Most of the time people give up because the teacher focuses too hard on each step prematurely, and just bores the hell out of the student, or confuse him/her because pieces/clues to aid the student are missing since they come from other aspects of the subject matter at hand. Usually those aspects can be picked up while getting the general overview of each aspect of the topic, as I'm doing here.

So the reason I'm teaching like this is because once you have that birds-eye-view framework, you can so much more easily dive deeper and learn the specifics. The

reason this is true is because it allows you to see how things piece together. It allows you to see the practical application of things. It allows the neurons in your brain to start building a fuzzy somewhat distorted picture of the entire puzzle, but at least some sort of picture of how it all relates. That makes it so much easier to find and learn what you're looking for later, and ultimately put those puzzle pieces in place. In essence, you'll now be on the hunt for specific puzzle pieces. That's really the entire premise of my entire teaching/learning technique. Learn general to specific, and then dive into the sub-topics more, then sub-sub-topics, and so on. It gives you a sense of purpose when you decide to go back of your own volition and figure out a specific aspect of the whole subject matter. Your passion and enthusiasm--birthed from quickly getting an overview of the entire subject matter--make it so much easier for you to understand something. If you care to learn something you will. If you're force-fed it (prematurely usually), you are more likely to be confused and waste your time.

The other approach is specific-after-specific. Here you'll spend hours on one point when you're not quite ready to learn it. With general-to-specific, easy to understand pieces of information from other areas of the top level subject matter will serve as clues to aid better understanding when you go back and dive into each specific sub-topic, and you'll simply learn quicker. But don't forget, the key ingredient is truly wanting to learn the subject-matter. Hopefully after getting a birds-eye-view of the entire subject, like I'm providing now, you'll cultivate your own interest to learn more ;).

FINAL THOUGHT: it's very important to learn an entire subject matter. As a programmer-in-the-making, you may be inclined to take what you've learned and start coding immediately, but before you've learned enough of the topic at large. In reality this will lead to you coding away, and then eventually spending hours just to research how to solve one little aspect you need. If you learned the whole subject matter of, say, procedural PHP, you most likely will have naturally encountered that solution, and in a faction of the time! Often it can take very many hours to research one small solution that results in one line of code. Whereas learning that trick might have been a natural thing to learn as part of learning the whole subject, and only requires 5 minutes of study in between learning many other tricks. In other words, a developer that has to constantly seek out solutions to things he/she doesn't know will waste a lot more time in aggregate than someone that mastered the subject as a whole, and then went to apply it. You're just more relaxed and in a better learning mode when you're focused on nothing but learning. But when you're focused on producing results, and have to learn at the same time, it can be stressful and waste you tons of time going back and forth from testing each of the tens of wrong solutions you're trying out and googling until you find the right one.

1-8 PHP OOP PHP | INTRO TO OBJECT ORIENTED PROGRAMMING IN PHP

We’re going to start by going directly to a code example that will highlight the power of object oriented programming. I wish I saw this as the first thing I learned when learning OOP. Usually books/tutorials on OOP explain all the theory and individual features of an OOP language, but you’re left having no clue on the magic you can do with it. Once you see and understand the magic, neurons in your brain will get to work figuring out what aspects of OOP you need to learn next to get productive, and it will quickly all make sense. Here we go:

```
$article1 = new Entry;
$article1->title = ‘The Bible’;
$article1->pageCount = 100;
$article2 = new Entry;
$article2->title = ‘OOP PHP Programming’;
$article2->pageCount = 200;
$articles = array(); //this is the creation of an array
$articles[ ] = $article1;
$articles[ ] = $article2;

foreach($articles as $article) {
   echo ‘The book, ‘ . $article->title
   . ‘ has ‘ . $article->pageCount . ‘ pages. <br>';
}
```

When output to the browser from your web server, the result is:

```
The book, The Bible, has 100 pages. The book, OOP PHP Programming has
200 pages.
```

Ok so what’s happening? The first thing I want to point out is that I used an array. I used an array to store multiple objects. The 2 objects were created with these lines:

```
$article1 = new Entry;
$article2 = new Entry;
```

I used an array to store 2 objects. Then I looped through the array, echoing values from the two objects. I was able to access the data stored in the object in identical ways, which is why the foreach loop worked. The idea is that the object is a package of structured data. It’s like a variable in that it starts with a $ symbol, i.e. *$article1*. However it contains more data, similar to how an array contains a list of data. Syntactically it’s different in that you access the sub-data like this: *$article1->title*; . I.e. the “keys” are strings (never numbers) that come after -> .

Next the idea is that each of the 2 objects is generated from the same template, specifically with the “new Entry” statement. There’s a term you will hear non-stop as you dive deeper into OOP for this template--it’s called a “class.” For now just think of the Entry class as a template, from which we can generate similar objects. Elsewhere this template/class is defined, similar to how a function is defined. The main thing here is that each objects generated from this template/class all have the same “keys,” i.e. again the part that comes after ->. In objects, these “keys” are actually called “properties”--so that’s how we’ll refer to them from here on out, now that you get the correlation.

In this example, the “Entry” class has 2 properties: “title” and “pageCount”. And each object generated from this class/template can assign their own unique values to these properties. This allows for us to iterate through an array containing these 2 objects and request data from these 2 properties and know that something will be there. Imagine you had 10 objects stored in an array, and each object corresponded to a blog article, i.e. it carried with it all the content necessary to display a blog article. So it would have the title, body of the article, author name, list of tags, etc. That’s where the power lies. You can write very simple code that will extract data from a similar place in a list of many objects like we did here:

```
foreach($articles as $article) {
   echo ‘The book, ‘ . $article->title
   . ‘ has ‘ . $article->pageCount . ‘ pages. <br>';
}
```

Capiche. Because the data is structured in a similar format, we can be sure it will always be there.

Right now, you should be asking yourself what the fundamental differences are between arrays and objects. Both contain basically lists of data, right? Well, technically you could achieve similar results with arrays, but you’d have to individually define the

keys of each array to be things like “title” and “pageCount”. With object oriented programming, you can define a class once like this:

```
class Entry
{
   public $title;
   public $pageCount;
}
```

You do that once, and you can create objects created from that template/class any time you want without writing the same code. Note: to more experienced OOP developers, don’t worry I’ll soon start referring to classes as templates--I just feel that correlation is extremely important to make for newbies, so I’ll keep combining the two with a slash for a little while longer before just calling them “classes” always.

So the key principle here is that each of these objects have the same “interface.” Interfaces are the key to good programming. If you’re new to this, which most of you are reading this, then the term “interface” is sure to get you thinking, curious, and possibly confused. My favorite analogy here is that of a Microwave. You know how it has all the buttons to control it, and the numbers 1-9 to determine how long you want your food to cook for, etc, right? And you can cook your meal with these buttons without having to know the internals of the microwave, i.e. you don’t have to know the slightest thing about how micro “waves” vibrate your food to cook it, etc. Well, that’s an interface. An interface provides a simple way to control something without having to know anything about its internals.

In object oriented programming, these object *$variables* are accessed by these interfaces, i.e. the properties “title” and “pageCount”. The takeaway here is that the interface is CONSISTENT!!! From object to object (generated from the Entry class), the interface is the same. And that’s what allows you to loop through objects (such as

those in an array) and echo out the unique data within each object, but without having to code any different code to access each object. They are all accessed the same.

1-9 PHP OOP PHP | OBJECT METHODS (AKA FUNCTIONS)

Ok so if you get the bit about objects being similar to arrays in that they hold lists of data but are accessed through “properties” rather than “keys,” and you’re asking yourself: “that’s all?” then prepare yourself for the cool stuff. Here we go--lets lead with an example:

```
class Entry
{
   public $title;
   public $pageCount;

   public function getInfo()
   {
      echo ‘The book, ‘ . $this>title . ‘ has ‘
      . $this>pageCount . ‘ pages . '<br>';
   }
}
```

Now let’s loop through those objects previously created in the OOP PHP 1 tutorial:

```
foreach($articles as $article) {
   echo $article->getInfo();
}
```

The most important thing to notice--the practical goal accomplished here--is that in our “client code” we can echo the result of this code out to the browser:

```
echo ‘The book, ‘ . $this>title . ‘ has ‘
. $this>pageCount . ‘ pages . '<br>';
```

and do so without having to write that code. What I mean is that we wrote it one central place. If in another part of your web application, you need to echo this info again, well boom, just type *echo $article->getInfo();*. Capiche!

What has happened here is that we've given our object not just properties, but methods, and we did so with this line in the class code:

```
public function getInfo()
{
   echo 'The book, ' . $this>title
   . ' has ' . $this>pageCount . ' pages . '<br>';
}
```

In essence, objects derived from the Entry class have a function called *getInfo()* . To understand the difference between methods and the functions you previously learned about, let's compare this to regular functions like *date()* or *halveMonth()*. Notice those do not come after *$objectVariableName->* . Basically those sort of functions are global functions, where as the *getInfo()* function (called "method" in this case) applies only to a given object. What makes it special is that it innately has access to the other data stored in the object, specifically the data stored in its properties.

Now this is where you're likely to have trouble understanding what I'm about to teach, but let's give it a try. Go back to the code in your *getInfo()* method:

```
echo 'The book, '. $this>title . ' has ' .$this>pageCount . ' pages';
```

See the "this" in bold. That's a special php feature to refer to the current object. Remember, this method exists within a class, and a class is a template. It's not the real thing. It's not a real object. It's a blueprint of what a real object will look like. So it needs a way to refer to itself. Later in client code objects generated from this class (technically called "instantiated") will not use "*$this*" but rather the $variable name you assigned the object too.

Really, the best way to understand this is straightup by testing these examples and making a few modifications yourself, i.e. actually doing a little coding. It's extremely simple to do, but hard to first get in its abstract written form. When you see it in action,

it will instantly make sense--or at least a lot quicker. My suggestion is to flip to the Setting up Your Server tutorial, and get your server working and executing php, and immediately try these out by copy/pasting the code into one file. Typically, if reading a book about this, you'd go on to read many many more chapters all about the theory of OOP and all its features. But that's stupid. If you can truly grasp this, all the rest will be fluff that just adds few other cherries on top.

Here's the complete code snippet to copy/paste and execute:

```
class Entry
{
   public $title;
   public $pageCount;

   public function getInfo()
   {
      echo 'The book, ' . $this>title . ' has '
   . $this>pageCount . ' pages . '<br>';
   }
}

$article1 = new Entry;
$article1->title = 'The Bible';
$article1->pageCount = 100;
$article2 = new Entry;
$article2->title = 'OOP PHP Programming';
$article2->pageCount = 200;
$articles = array();

$articles[ ] = $article1;
$articles[ ] = $article2;

foreach($articles as $article) {
   echo $article->getInfo();
}
```

I'm not going to get into the other features of OOP, because in reality what you just learned here taught you the features of OOP php you'll be using 80% of the time. Coders reading this may be freaking out, but the reality is you need to get your startup executed asap. You need to cut the bullshit and get productive asap. So this approach is very practical. Understand this, then refer to my further reading section to get a list of other books you should read to acquire true mastery.

1-10 PHP OOP PHP | INHERITANCE IN OOP

And actually there is one last thing worth learning now about OOP. You can make new classes that inherit from another class. Check this code:

```
class BlogEntry extends Entry
{
   public function exageratePageCount()
   {
      echo $this->pageCount + 20;
   }

}
```

Notice how the "pageCount" property is not defined in the *BlogEntry* class. It's because it's inherited from the *Entry* class. Now let's rebuild $article 1, but from the BlogEntry class/template instead, and execute this new method:

```
$article1 = new BlogEntry;
$article1->title = 'The Bible';
$article1->pageCount = 100;

echo $article1->exageratePageCount();
```

Now the browser reads that it has 120 pages. The idea is so damn simple: objects instantiated/generated from the BlogEntry class have access to all the properties and methods from the parent class, Entry. Therefore you can write a reduced amount of code when you want to create similar classes. point blank period. Again, don't worry too much about inheritance yet. It's a feature you'll use a lot, but a very simple one to code up and make use of. You'll know when you need to use it to not have to write redundant code.

Super final note on OOP: visualize all this OOP stuff as simulating a world in programming. The idea is to make coding more human-like, more like the real world. I.e. classes can have children and parent classes, just like humans can have real

children and real parents. Later on you'll learn even more ways objects can work together, just as objects and organisms in the real world can. But I'm not covering that in this book. Look out for the terms "aggregation" and "composition"--they are techniques to make objects work together to create re-usable code and efficient solutions. For now inheritance is the code reuse technique we'll leave you on. Don't plan on abusing it ;).

Conclusion & Further Reading

We at FaceySpacey hope you've enjoyed our FaceySpacey Bible, and are coming away many times more ready to succeed at your next software startup. At the very least, you should have a birds-eye-view of what you need to do to get your startup, and have quelled a lot of insecurities you may have had regarding how you should execute it. That said, I will point to you to what you should do next.

As promised, the following is a list of the precise books I read to master web development using HTML/CSS, Javascript, PHP & MySQL. They are presented in the best order to most efficiently learn the subject at hand. It's similar to the order I read them in, but enhanced based on what I learned and the order I wish I read them in. Good luck:

HTML/CSS:

CSS Mastery: Advanced Web Standards Solutions

http://www.amazon.com/CSS-Mastery-Advanced-Standards-Solutions/dp/1430223979/

Before you start coding PHP, Javascript, etc, understand how HTML works. This is where you start. HTML is easy. Read this book in combination with studying the HTML & CSS tutorials on w3schools.

PHP & MySQL:

PHP & MySQL For Dummies, 4th Edition

http://www.amazon.com/PHP-MySQL-Dummies-Janet-Valade/dp/0470527587

This book--well an older edition--I read a year before I got serious about learning to code. I read it and didn't actually code anything i learned, but what it did was plant seeds in my head with regards to what programming is all about and what databases are all about, and how to connect the two. It assumes very little in what you may

already know, and is an excellent start in your journey to becoming a master programmer.

PHP Object-Oriented Solutions

http://www.amazon.com/PHP-Object-Oriented-Solutions-David-Powers/dp/1430210117

This book is where I learned what OOP is. I didn't get the hang of it until reading the following book. Don't worry if you read this and have a hard time with it. This book and the next each have introductory chapters that go over how OOP works. It took me reading basically this book and the next book about the same stuff to get it. This book is a lot less complicated than the following and dives into practical examples & problems, whereas the next is a lot more theoretical.

PHP Objects, Patterns and Practice

http://www.amazon.com/Objects-Patterns-Practice-Experts-Source/dp/143022925X

After reading this book, I basically mastered OOP. It's a very hard book to get through if you're new to OOP, and goes into some very advanced stuff, specifically tons and tons of "design patterns." The design patterns are presented in as basic of a form as possible, but they weren't very practical like examples from the previous book in that you probably will never actually need any of the code used in the book. Either way, this is my favorite Programming of all time because it taught me how to think like a coder and how to solve complex problems with concise refactored solutions. It really showed me what is possible with OOP. You don't know PHP unless you've read this book.

Pro PHP: Patterns, Frameworks, Testing and More

http://www.amazon.com/Pro-PHP-Patterns-Frameworks-Testing/dp/1590598199

I read this book too just to solidify my experience with PHP, and cover all my bases. Check it out. It's optional.

PHP Functions Essential Reference

http://www.amazon.com/Functions-Essential-Reference-Torben-Wilson/dp/073570970X

At some point during my study of PHP I found this book and decided just to learn every PHP function available so that I could better understand the examples in the above books. Start reading this early on, and complete the whole thing. You'll quickly learn patterns in how PHP functions are named, and as a result be able to guess what a function does within the context of the examples in the above books--even if you don't remember precisely what it does.

Agile Web Application Development with Yii 1.1 and PHP5

http://www.amazon.com/Agile-Web-Application-Development-PHP5/dp/1847199585

I read this book in combination with reading the Blog Tutorial and Definitive Guide on YiiFramework.com. When you're done studying all these materials, you'll be amazed with how much power you have. This book isn't hard to read either. You'll love it if you reach this stage!

Javascript & jQuery:

Learning jQuery, Third Edition

http://www.amazon.com/Learning-jQuery-Third-Jonathan-Chaffer/dp/1849516545

jQuery is a framework built on top of the native browser language of Javascript. Usually one would recommend you learn the base language--Javascript--before learning an abstracted framework on top of it--jQuery. However because of the nature of jQuery and how comprehensive it is and because of how quirky Javascript is coming from PHP, I found it best to jump to jQuery and immediately start accomplishing the DOM manipulation tasks I needed. And ultimately because of syntax similarities between PHP and Javascript I was able to get productive in Javascript without studying a single book just on Javascript. One thing I did different

when studying this book from the PHP books is I did every single tutorial as I read it. The reason is because when I learned PHP, I was learning my first real programming language--so it took me a lot of time to just digest things before I could code a single line, which is why I just read PHP book after PHP book before I got started until it all made sense. However, by the time I got to Javascript & jQuery, I understood how programming in general works and found it helpful for memorization purposes to immediately start doing the tutorials.

jQuery 1.3 with PHP

http://www.amazon.com/jQuery-1-3-PHP-Kae-Verens/dp/1847196985

With this book I didn't do all the tutorials like I did with *Learning jQuery*, but what reading this book did for me is taught me precisely how Ajax works and what it's all about. After reading it, coding features that required Ajax using Yii and PHP was obvious and a no-brainer.

JavaScript: The Good Parts

http://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742

This book gave me a deep understanding of the Javascript language and what it's truly all about. After reading it, many hours of debugging and head-scratching when coding Javascript & jQuery were removed from my schedule--because I finally learned the quirks of the Javascript language I needed to know.

Pro JavaScript Design Patterns

http://www.amazon.com/JavaScript-Design-Patterns-Recipes-Problem-Solution/dp/159059908X

Now this book took my Javascript game to the next level, gave me an idea of how jQuery was built, taught me how to do things similar to how you would in a "classical" OOP language like PHP, and completely ended any remaining head-scratching I was having with Javascript, particularly with how "scope" works in Javascript.

Linux:

The Official Ubuntu Server Book, 2nd Edition

http://www.amazon.com/Official-Ubuntu-Server-Book-2nd/dp/0137081332

Note: by the time I read this book I had already learned Linux through blogs on the internet. The best thing I can recommend you do is install Linux on your computer from the Ubuntu website, and start navigating around the command line, practicing Linux commands you learn off the web. Just google "linux tutorials" and you'll be off to a running start. That said, by the time I got proficient in Linux and after I read this book, I felt confident that I really knew what I was doing and had practical solutions for the most common problems you'll face at the command line.

Apache Cookbook: Solutions and Examples for Apache Administrators

http://www.amazon.com/Apache-Cookbook-Solutions-Examples-Administrators/dp/0596529945

This book I treat like a pocket reference and still refer to it often since it's impossible to remember all the different Apache configurations, given how comprehensive this web server application is. I did read it through when I first got it. I kinda skimmed it though--just to get an idea of what is possible. Getting an idea of what is possible without mastering a subject matter is so important in programming because you'll know where to look when you face a challenge that the subject matter can solve.

Pro Bash Programming

http://www.amazon.com/Bash-Programming-Experts-Voice-Linux/dp/1430219971

This is a little advanced for readers of the *FaceySpacey Bible*, but I'm putting it here because it really took my Linux skills to the next level.

NON-TECHNICAL BOOKS:

Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent

http://www.amazon.com/Smart-Gets-Things-Done-Technical/dp/1590598385

If you plan to grow a small application into a large company, this book is a must. It's short. Read it.

SEO Book.com

http://www.seobook.com

This obviously isn't a book, but I read their entire site like a book, and its creator, Aaron Wall, expects you to read it like a book. When I was done reading it, I felt I was completely up to speed regarding what SEO is, how search engines work, and practical techniques to get better rankings in search engines.

For daily Startup Wisdom, checkout FaceySpacey.com/blog daily. And don't forget to download the entire No Bullshit FaceySpacey Bible or more individual chapters here: http://www.faceyspacey.com/resources?section=book .

*Thanks again from FaceySpacey and be sure to check out **FaceySpacey.com** often for further knowledge we kick to take your Startup to the Stars!*