Ghacks.net Web Development Articles January 09

How does PHP work?

When someone starts learning web development they usually start with HTML and CSS, and many people get stuck there without ever experiencing the wonderful work of server side scripting like PHP or ASP. I'll be talking about PHP here, but the basic rules for ASP and others are the same though.

The most important thing to understand is the difference between HTML and PHP. In HTML you write your code, upload it, and the user's will subsequently download that page along with <strong>all the code</strong>. The user's browser interprets this code and shows the user the page as you intended it (hopefully). In other words HTML is sort of what you see is what you get, in the sense that all the code goes to the user and is interpreted by the browser.

With PHP it works a bit differently because you don't actually download the code the author wrote. What happens is that if you want to download a php page the code in that file is first processed by the server, and you download the <strong>output of the code</strong>, as opposed to the whole code as is. This in turn will be HTML just as before, this is why you never see PHP code in the source of a webpage. So what happens in processing? Turn the page to find out!

With PHP the goal is to use the processing powers of the server to build (usually) dynamic webpages. A very basic example is showing the correct greeting for the time of day on a webpage. In human terms you write a script with the directions to show "Good Morning" if it is before 10am but after 5am, "Good Afternoon" if it is after 10am but before 6pm and "Good Night" after 6pm, before 5am. Instead of receiving all the code for this and processing it in your browser, this all gets processed before you download it, and you only get the result of the process, the text "Good Night" for example if it is 9pm.

This is much quicker, since if you think of bigger sites, instead of having to download 300kb (or much more) of code, it is quickly processed on the server and you might get as little as 10kb or less. Obviously your PC could process the code quickly, but downloading and handling can take a while. In addition, the code may also have database queries which can not execute if processed on your PC, they have to be processed on the server which has the database.

If you would like a more real-life example, take a look at gHacks, which has almost 5,000 posts now. In HTML world we would have to have 5,000 posts which all have the whole site code, from header to footer with the article in between. PHP makes it possible to "compress" those 5,000 files into only 1!

When you view any gHacks post on a single post page you are actually viewing a file called single.php. This file also has some additional info in the url which will tell the script which post to show, so the file you are viewing would be single.php?p=234. This tells the script that the post with the ID of 234 needs to be shown. The script queries the database for the relevant post and pulls its details (like title and post body) from the database. So in the end all you are shown is one post. Wordpress has some other stuff built in to make nicer URL's and so on, but under the hood this is what is happening.

Likewise for the front page we don't always go and modify the code when posting something. Martin would be coding all day, removing the last post on the page and pasting the code of the new one. Instead, in the php file you are viewing the code gets the latest 10 posts and puts their data on the page.

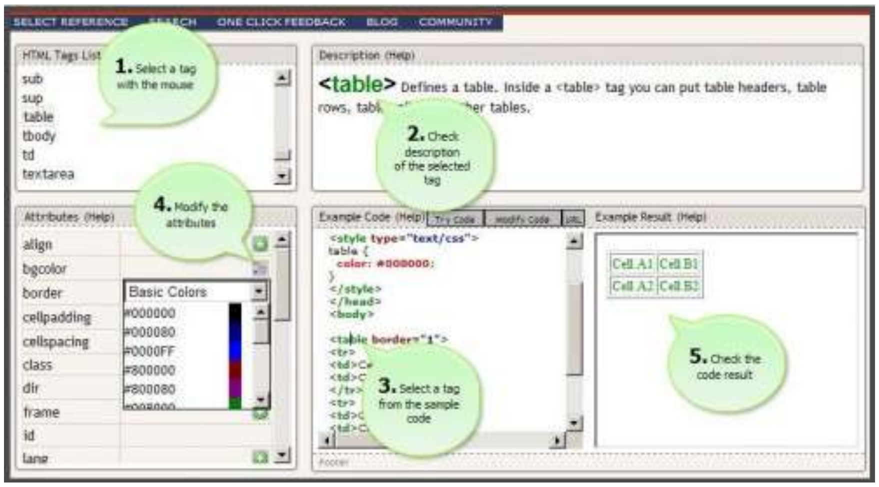
There is a lot more to learn in PHP, but those are the basic mechanics, the ability to create pages based on certain criteria, as opposed to static content on each page.

HTML Playground

The first language that you need to learn if you want to start with web development is HTML. HTML stands for Hypertext Markup Language, a set of tags and attributes that can be used to create websites. HTML comes with a limited amount of tags and a basic form that is always the same. It basically comes down to learning the general layout of a HTML document and the tags that can be used to create the website.

HTML Playground provides the means to play around with all HTML tags (and a bit of CSS). It does not require any prior knowledge although that could be of an advantage. The website makes use of an interface that is divided into four columns. One holding all HTML tags, another that is displaying "live" examples of a selected tag and a third that is displaying valid HTML code used to create the example. Valid HTML code meaning a full html page making use of that code. Each HTML and CSS tag is clickable in the example code leading directly to an example page of that tag.

It would not really be a HTML Playground if the website would not offer the means to edit the example code. This is excellent to test changes to the code to see how the tags work.



<table> Defines a table. Inside a <table> tag you can put table headers, table rows, tabl… …her tables.

```
<style type="text/css">
table {
  color: #000000;
}
</style>
</head>
<body>

<table border="1">
<tr>
<td>C
<td>C
</tr>
<tr>
<td>C
<td>C
```

Cell A1	Cell B1
Cell A2	Cell B2

Beginners might find it a lit difficulty to start and it is recommended to have read at least one article about learning HTML before playing around with the various elements. It is best for users who know the basic of HTML and want to start playing around with code.

PHP - what role does it fill

Before we try to work with PHP we need understand the role it fills – what problem does it address. The World Wide Web is built on a client-server model. A client computer requests a page which is supplied by a Web Server. The browser then renders the page for the user to view. The simplest type of pages contain static (unchanging) content. The server could serve plain text files, and the browser wouldn't have any trouble rendering them.

HTML is a markup language that lets us describe attributes of the text and blocks on our pages. This works great for simple requests, making pages much more interesting than plain text. However it leaves us with a very simple structure. One page from One url (address) results in one rendered content (every time this url is requested, the output is the same).

To give us more options we have programming languages. Some like PHP run on the server side. They modify the content that will be displayed before it is sent to the client and on to the browser. Others like JavaScript run on the client side and allow changes to be made in the browser after the page has been rendered – usually for interactivity or for adding a feature not normally available in that browser.

The very first thing you _must_ do before getting started with PHP is get a good grasp of HTML (and CSS). Many webmasters use a tool like Dreamweaver, Expression Web or KompoZer. To

work with PHP it is important that you understand the underlying HTML code. You will be modifying this code so you need to be able to understand the HTML well enough to understand what you see.

If you like video training try Lynda.com or VTC.com. If you prefer reading a book try "Head First html with CSS & Xhtml" or if you prefer free web instruction try w3schools.com and tizag.com.

Now that you understand the markup language which is what is sent to the browser (HTML) you are ready to tackle the server side use of PHP.

Standardizing variables to code faster

Welcome to a new exploratory post here on gHacks! I actually work as a web designer specializing in standalone PHP and MySQL based sites, and I've always wanted to write a bit about coding, but it doesn't really fit my own blog and I just don't have time to start a whole new blog for it. Since gHacks is a tech blog we thought we'd try it out here with Martin and see how it goes. if you like the idea of similar posts here on gHacks please let us know in the comments (and also if you don't), we want to post stuff you guys are interested in, so feel free to boo it on out of here! So here goes nothing:

If you've been coding for a while you probably have some set ways in which you work. I recommend you also widen this method to naming variables, which when done consistently can be a huge help, especially when you are using some fancy MySQL along with your PHP code. If you are building a site with 15 MySQL tables it is helpful if you know all the column names because you always name them in the same way, but the same thing goes for simple variables, and even ids and classes for HTML elements.

A very common example can be variables which hold a user's data. In my case I always use the variable '$u' for holding the username, and the variable '$p' for holding the password. If you are grabbing a lot of data, like first name, last name, email and so on you can make up your own standard set, I use '$fn', '$ln' and '$e' respectively for the three.

Since you usually use these variables in the same script the importance of naming them consistently may not seem like a huge time saver. However when you get into session variables it saves you a lot of time if you don't always have to look at where you are defining them to find out the names of each. I pull user data using a mysql query when a user logs in and I immediately place the whole array in a session variable name "$_SESSION['user']". The names of the columns represent the values of this array, so the username and email would respectively become "$_SESSION['user']['Username']", "$_SESSION['user']['Email']". This is the same in my case for every single website I build where applicable.

User data naming consistency is also very prominent in the MySQL tables themselves. As you can see above my columns are also named the same in each case, a user's email is always "Email", his last name is always "LastName". I steer clear of abbreviations with MySQL tables so other coders can easily see what's going on as well. Consistency doesn't just apply to the actual names, you can also apply it to cases. As you can see all my column names are always capitalized, they are like wikiwords, but user session variables which don't come from MySQL are not capitalized. For example, when a user logs in he is assigned a time, which is usually one hour. If he/she is inactive for more than this time (no refreshes), he/she is logged out. There is no need to store this in the database, and so I know that this is not in there the session variable here is "$_SESSION['user']['time']". This means that if I come back to work on a site a year after completion I know exactly where variables come from.

Even if you don't yet know any PHP, you're just getting into HTML and CSS you can still use these good practices. When assigning ids and classes to elements try and work out a structure for yourself that you always use. I use empty div tags to clear floats a lot, so I have a separate class named "cl" for this purpose. Whenever I start out a new webpage I include this class by default in the stylesheet since I know I will most likely need it.

That's about it for standardizing your coding work, most of it is just common sense really. Try and work out your own method which works best for you. It takes time and a lot of practice to get a system working well since there is so much to think of, but if you always keep this in mind somewhere while coding you'll be a faster and better coder in no time at all!

Web Development: A brief history of time()

Part of the beauty of PHP to me is the number of really useful variables that are built in. Some of these might seem very odd at first, but once you start creating pages you will run into some problems which you'll find can be solved by a function which seemed totally useless when you first heard of it. One of these functions for me was time().

Echoing the time() function will give you the amount of time passed since the Unix Epoch in seconds. Say what? Epoch means a point in time chosen as the start of a period or an era and thus the Unix Epoch is January 1 1970 00:00:00 GMT. So echoing the time() function will give us "1230978041" at the time I'm writing this, meaning that 1,230,978,041 seconds have passed since then. So why is this useful to us?

Mathematically it gives us a much easier way of keeping track of time. Sure, 2008, Jan 15th might seem all nice and organized to you, but to calculate the days passed from the 15th of Januaray to the 17th we's have to strip the numbers of "th", and in more complicated cases perform a bunch of other string changes. Using time you can essentially assign a number value to any given second, making it much easier to use, especially as a counter.

The place I use it most is to log out users of a website automatically after an inactivity period. When a user signs in I create all the session variables for him/her, and I also create one which holds the time his session should expire. The beauty of this whole system is that I do not need to know the actual time, I can just assign a value to the session variable like this:

```
$_SESSION['user']['time'] = time() +3600<
```

This means that the user can stay logged in for 3,600 seconds (in other words an hour) from this moment in time. This is a very convenient way of defining expiration time, since you can think in terms of how long you want it to be, as opposed to trying to calculate a specific date and time.

When a user refreshes a page, or moves on to a new one, a script will check the value of the session variable. If the current time is smaller than the session variable, the user can stay logged in and I also usually prolong the session by another 3,600 seconds. This gives it the true "inactivity" aspect, since the user is allowed to stay logged in for more than an hour, as long as he/she is using the system. You could however choose not to prolong the session, in this case the user would have to log in again after one hour no matter what. In some high security systems this might be the way to go, or if you want the user to spend exactly one hour on a specific puzzle, there are many uses for everything. Needless to say that if the current time is more than the session variable the user is signed out.

Another common use for time() is to serve as the basis for generating a random ID, or character set, in other words, it can serve as a seed. A quite efficient way of creating a very random string would be to use time(), divide it by a random number generated between 0 and 9,999, add some random characters to it, and encode the whole thing using the SHA1 algorithm for example. Code-wise this is not as difficult or as long as it may sound, and it is pretty random and strong, although I am no security specialist.