PHP Fundamentals

Gayanath Jayarathne

PHP Fundamentals

Learn PHP Fundamentals in Plain English

Gayanath Jayarathne

This book is for sale at http://leanpub.com/phpfundamentals

This version was published on 2013-08-31

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Also By Gayanath Jayarathne

MySQL Basics with PHP

To my mother

Contents

Part 1 - Basics **1**

Chapter 1 - Role of PHP in Web Applications **2**
Dynamic Content 3
How PHP Engine Outputs Content 4
Embedding PHP in HTML 5

Chapter 2 - Installing PHP in Your Computer **7**

Chapter 3 - Working with XAMPP **8**
Downloading and Installing XAMPP 8
Starting Apache and MySQL 8
Apache Is Not Starting Error 9
Making Requests to the Server 9
Putting Stuff in Web Folder 10
Root URL and Home Page Content 10
Locations of Configuration Files 10
Restarting Apache and MySQL 11

Chapter 4 - How to Run PHP Files **12**

Chapter 5 - Getting Information about PHP Installation **13**

Chapter 6 - Choosing a PHP Editor **14**

Chapter 7 - Printing Content with echo() **15**
Different Usage of echo() 15
Difference Between echo() and print() 16

Chapter 8 - Usage of Quotes **17**

Chapter 9 - PHP Variables and Constants **18**
Naming Variables 18
Arrays 19
Predefined Variables 19
Constants 19

Chapter 10 - PHP Arrays **21**
Indexed vs Associative Arrays 22
Defining Arrays 22
Array Functions 23

Part 1 - Basics

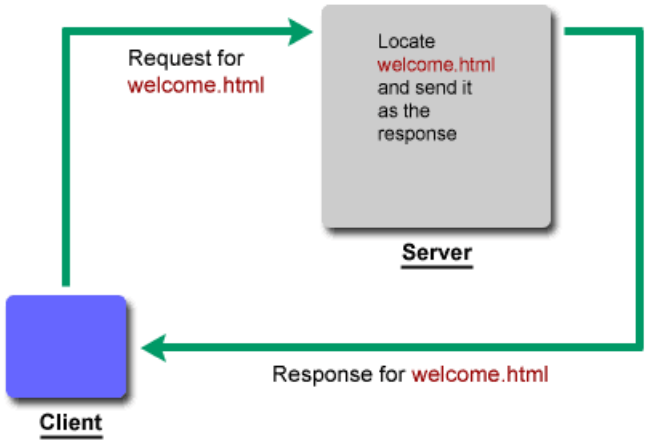
Chapter 1 - Role of PHP in Web Applications

PHP is a server side scripting language. That means its processing happens in the server by consuming server's resources and sends only the output to the client. In a client side scripting language like JavaScript, processing happens in the client's computer consuming its resources.

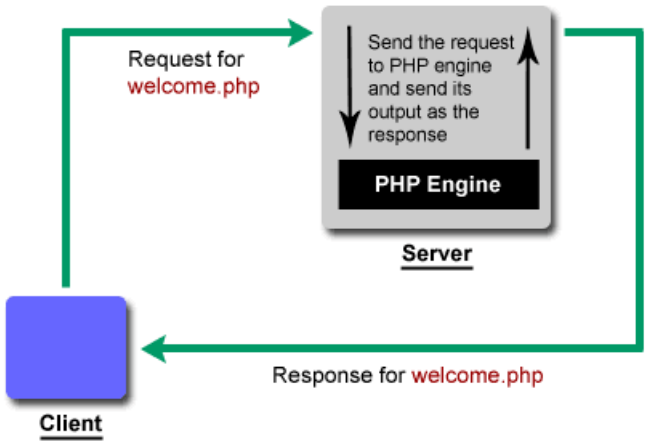
PHP can also be used as a language for Command line Scripting and Desktop applications. But it's widely used for web applications.

To understand the role of PHP, let's look at how a normal web request and a web request that involves PHP happen. Please note that there can be many intermediate steps involved but only the main and important ones have been mentioned for understanding purposes.

Cycle of a Normal Web Request



Cycle of a Web Request That Involves PHP



Dynamic Content

You might have probably heard that "using PHP you can create dynamic web sites". To understand what "dynamic" means here, let's take a look at **welcome.html** and **welcome.php**.

Don't worry if you don't understand the code yet. Try to understand accompanying explanations.

welcome.html

```
<html>
<head>
<title>Welcome to Our Web Site</title>
</head>
<body>

<h1>Welcome!</h1>

<p>Rest goes here...</p>

</body>
</html>
```

welcome.php

```
<html>
<head>
<title>Welcome to Our Web Site</title>
</head>
<body>

<h1>
<?php
if (date('G') < 12) {
    echo 'Good Morning!';
} else {
    echo 'Welcome!';
}
?>
</h1>

<p>Rest goes here...</p>

</body>
</html>
```

No matter what's the current time of the day, if you request **welcome.html**, you would always see **Welcome!** as the heading. But if you request **welcome.php** before 12 noon, you would see the heading as **Good Morning!** and **Welcome!**, if it is after 12 (**date()** function uses server's time and it may be different from your PC's time).

Here **welcome.html** has a static behavior because it delivers same content always. But **welcome.php** has a dynamic behavior because the content it delivers changes according to the time of the day. You can improve this file and have different greetings for different time periods. As illustrated in this example, PHP can be used to build dynamic content that is based on the context. You would experience the real benefits of PHP's dynamic behavior when you create database driven web applications.

How PHP Engine Outputs Content

When PHP interpreter reads a file, it only processes lines between **<?php** and **?>** tags. It outputs rest of the lines without any processing.

For example, in **welcome.php**, PHP interpreter outputs all the HTML content till **<?php** tag without any processing. Then based on the time of the day it adds **Good Morning!** or **Welcome!** to the output. Then from **</h1>** tag, it outputs rest of the lines without processing. Web server collects all these outputs and sends to the client who made the request.

As stated, only the output is sent to the client. So, no PHP content can be seen by viewing the source of PHP files in client's web browser.

You can have more than one block of PHP content in a page and variables you define in these blocks are available to next blocks. For example, below PHP file (**welcome-new.php**) behave as same as **welcome.php**.

```
<?php
if (date('G') < 12) {
    $greeting = 'Good Morning!';
} else {
    $greeting = 'Welcome!';
}
?>

<html>
<head>
<title>Welcome to Our Web Site</title>
</head>
<body>

<h1><?php echo $greeting; ?></h1>

<p>Rest goes here...</p>

</body>
</html>
```

Embedding PHP in HTML

When we mix PHP and HTML content in a PHP file as we did in **welcome.php** and **welcome-new.php** we call it as "embedding PHP in HTML". We could achieve the same behavior with following PHP file (**welcome-latest.php**) which contains only PHP content.

```php
<?php

if (date('G') < 12) {
    $greeting = 'Good Morning!';
} else {
    $greeting = 'Welcome!';
}

echo '<html>';
echo '<head>';
echo '<title>Welcome to Our Web Site</title>';
echo '</head>';
echo '<body>';

echo "<h1>$greeting</h1>";

echo '<p>Rest goes here...</p>';

echo '</body>';
echo '</html>';

?>
```

Embedding PHP blocks in HTML let us get the help of PHP only where it is necessary. As you can see in these welcome files, we only need PHP to output the correct greeting. We can add rest of the lines to the output by just keeping them outside of PHP blocks instead of outputting each of those lines using PHP. This saves PHP processing resources and lets us have clean and readable files in our development.

Chapter 2 - Installing PHP in Your Computer

For running PHP, you need a web server because PHP can't directly deal with web requests. After installing, PHP acts as a sub-module of the web server which handles requests for PHP files.

PHP is free and supported by almost all recognized operating systems and web servers. Refer official documentation on how to configure PHP based on the operating system and the web server[1].

If you are new to web development, you may not have any experience on installing a web server. And most web applications rarely live without a database system. So you will also need to install a database system like MySQL[2]. You need these installed in your computer to simulate a web server which is required for your PHP development.

If you are less experienced in this subject and going to install each of these manually, it can be a tedious task which would then decreases your enthusiasm on learning PHP. Therefore it's recommended that you get these software products installed easily as possible.

Once you get your hands dirty with PHP, you can consider learning more on manual installations which would definitely be a plus when you are going to manage your own web server.

For many PHP applications, Apache[3] has been the desired web server and MySQL has been the desired database system. Being free and providing advanced features are the main reasons for this popularity. AMP (Apache, MySQL, PHP) is a common abbreviation used for these three products.

You might have seen the term LAMP which stands for Linux, Apache, MySQL and PHP (can also be Perl or Python) which has been a popular combination for server environments.

Over time, Nginx[4] (pronounced "engine X") has become a popular alternative to Apache and the term LEMP ("engine X" in place of Apache) is also in use.

There are several AMP stacks available which let you install all three easily without digging into complex configuration instructions. XAMPP[5] is such an AMP stack which is popular and available for Windows, Mac OS and Linux. Refer **Working with XAMPP** for installation and usage instructions.

---

[1] http://php.net/manual/en/install.php
[2] http://www.mysql.com/
[3] http://www.apache.org/
[4] http://nginx.org/
[5] http://www.apachefriends.org/en/xampp.html

Chapter 3 - Working with XAMPP

XAMPP[6] is an AMP stack which lets you install Apache, MySQL and PHP in your computer together with some other useful software. It has been designed to provide you an easy installation experience. It's free and available for Windows, Mac OS and Linux.

Downloading and Installing XAMPP

Go to XAMPP[7] web site and download the installer based on your operating system. Installation should be similar to a normal software installation you do in your operating system.

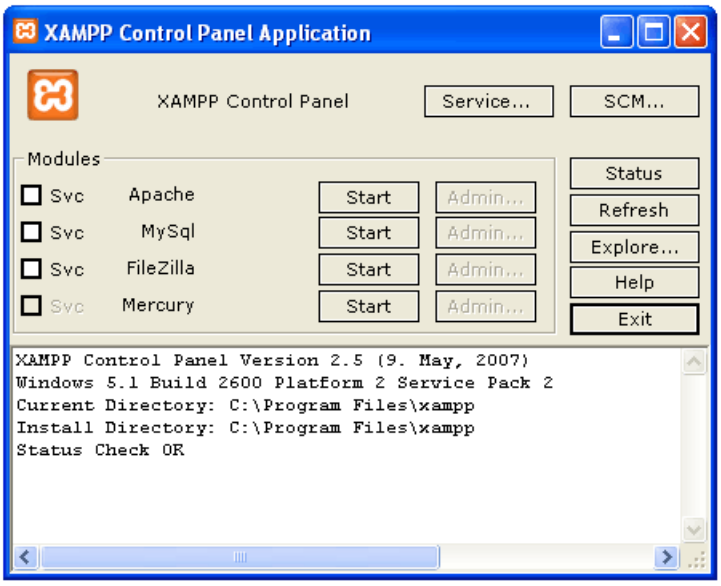
When installing, there would be an option to select whether you want to run Apache and MySQL as services. If you chose it, Apache and MySQL will start at system boot-up which may not need if your computer is tight with memory resources or if you are not doing PHP development frequently. You can change these settings after installation.

Starting Apache and MySQL

Instructions provided in this chapter are based on Windows operating system but the approaches should be similar for other operating systems.

Go to the location where you installed XAMPP (usually **C:\Program Files\xampp**) and double click on XAMPP Control Panel (**xampp-control.exe**). This will bring you following screen. Click on **Start** buttons next to Apache and MySQL for starting them.

[6] http://www.apachefriends.org/en/xampp.html
[7] http://www.apachefriends.org/en/xampp.html



Once opened, you would see XAMPP icon on the right of your task bar. Clicking on that icon will show/hide XAMPP control panel. To exit from XAMPP, click on **Exit** button in XAMPP Control Panel.

Apache Is Not Starting Error

Sometimes you would experience that even after clicking Start button several times, Apache is not starting. This is usually because some other service is running at the port required by Apache which is 80 by default. An easy reproducible way for this error is starting Skype[8] before starting Apache.

In such a case, you would need to stop other service temporary and restart it after starting Apache. For example, if you do so for Skype, it will find another port for working after restarting.

In the installation directory, you would see a program called Port Check (**xampp-portcheck.exe**). Double click on it and it will show you status of required port for Apache, MySQL and other software that comes with XAMPP. If required ports are already occupied, it will show the names of services that run on those ports.

Making Requests to the Server

Once you started Apache in control panel, type **http://localhost** in your web browser. This would bring you a web page that lists XAMPP related details.

[8] http://www.skype.com/

Putting Stuff in Web Folder

Under XAMPP root directory there is a folder called **htdocs**. That's where you should put your web site related stuff. For each web site you create, it's better to create a folder inside **htdocs** folder and then put content inside that to avoid conflicts.

For example, you can create a folder called **learnphp** inside **htdocs** folder and put **welcome.php** inside that. Then you can access it via the URL **http://localhost/learnphp/welcome.php**.

Root URL and Home Page Content

In above example, root URL of your web site is **http://localhost/learnphp/** and it's generally expected to see home page of the web site once root URL is typed in the web browser.

Usually web servers have been configured to look for an index file (can be **index.htm**, **index.html**, **index.php** etc) in the root of the web site folder and show its content for the home page of the site. So, if you had a file called **index.php** inside **learnphp**, you would see its output once you typed **http://localhost/learnphp/**.

Make sure you have only one index file in your web site to avoid conflicts. If you have more than one (say **index.html**, **index.php**), required file will be chosen based on the order defined in your web server's configuration settings.

Locations of Configuration Files

Based on your requirements, sometimes you would need to change default settings of your web and database servers. Usually this is done by altering their configuration files. In XAMPP, Apache, PHP and MySQL configurations files are located in following locations (This assumes XAMPP installation directory as **C:\Program Files\xampp**).

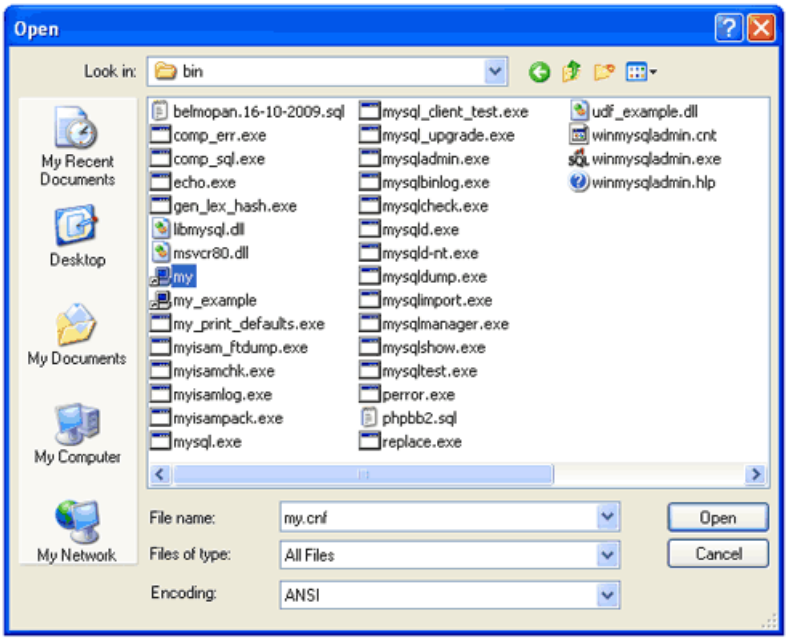
Apache Conf File (httpd.conf): **C:\Program Files\xampp\apache\conf\httpd.conf**

PHP Conf File (php.ini): **C:\Program Files\xampp\apache\bin\php.ini**

MySQL Conf File (my.cnf): **C:\Program Files\xampp\mysql\bin\my.cnf**

In XAMPP, some Apache configuration settings have been moved to sub configuration files under **xampp\apache\conf\extra**.

To open **my.cnf**, double clicking on it may not work sometimes. In that case, open Notepad and then locate **my.cnf** file by setting **Files of Type** to **All Files** as below.



Restarting Apache and MySQL

To take effect any configuration change, it's required to restart Apache and/or MySQL. In XAMPP, if these two are already running, stop them and start again in the Control Panel.

Chapter 4 - How to Run PHP Files

If you double click on a HTML file (files with **.html** or **.htm** extension), it would open on your web browser. But same won't happen if you double clicked on a PHP file (probably it would open in an editor). The reason is PHP files first need be processed in a web server before sending their output to the web browser.

Therefore before running PHP files, they should be placed inside the web folder of a web server and then make a request to desired PHP file by typing its URL in the web browser. If you installed a web server in your computer, usually the root of its web folder can be accessed by typing **http://localhost** in the web browser. So, if you placed a file called **hello.php** inside its web folder, you can run that file by calling **http://localhost/hello.php**.

Web folder can be changed based on your web host (if you hosted your web site online) or the method you installed the web server in your computer. If you used XAMPP to install Apache (web server) in your computer then the web folder would be **htdocs** which is under the root directory of XAMPP.

Chapter 5 - Getting Information about PHP Installation

PHP provides **phpinfo()** function that prints details about your PHP installation. It can be helpful when you want to know about a certain setting like the location of php.ini file.

Copy following code to a file located in your web server and run it in your web browser.

```
<?php
phpinfo();
?>
```

You will be able to see tabulated information about your php installation. Usually a file like this is used as a test file to check that PHP is up and running.

- First you will be able to see your PHP version.
- **Configuration File (php.ini) Path** shows where is your php.ini located. PHP may keep a separate php.ini file for command-line. This setting shows the location of php.ini file that corresponds to your web server (the one you will use most).
- Under **date** section, you will find the timezone used by PHP.
- For many PHP extensions installed, there would a table listing extension details.

Chapter 6 - Choosing a PHP Editor

PHP files can be edited using any text editor but you would prefer a bit advanced editor when you engage in your PHP development.

Chances are your operating system already offers a text editor with syntax highlighting[9] (If not, you should be able to find free one easily).

However when you work in a PHP project that involves many files, just syntax highlighting won't be enough. You will probably need an IDE (Integrated Development Environment).

An IDE is an application that integrates development tools like a debugger[10], an interpreter[11], a version control system[12] with the editor. So it lets you execute many development tasks at one place.

When you choose an IDE, look for following features.

- PHP syntax highlighting.
- Integrated interpreter so that if you make a syntax mistake it will be immediately visible.
- Autofilling that suggests PHP built-in code and the code you have already defined (like variables) when you start typing.
- Allowing easy refactoring like changing a method name across a file or across a set of files.
- Highlighting unused variables (this will allow you to cleanup your code).
- Integrated version controlling so that it clearly shows changed lines and allows you to revert back if needed.

EasyEclipse for PHP[13] and NetBeans for PHP[14] are free and feature-rich IDEs. Both are available for Windows, Mac and Linux operating systems.

---

[9] http://en.wikipedia.org/wiki/Syntax_highlighting
[10] http://en.wikipedia.org/wiki/Debugger
[11] http://en.wikipedia.org/wiki/Interpreter_%28computing%29
[12] http://en.wikipedia.org/wiki/Revision_control
[13] http://www.easyeclipse.org/site/distributions/php.html
[14] http://netbeans.org/features/php/

Chapter 7 - Printing Content with echo()

If you want to add anything from your PHP script to the output PHP engine generates (which then would be sent to the client), you can use **echo()** language construct for that.

```
<?php

if (date('G') < 12) {
    $greeting = 'Good Morning!';
} else {
    $greeting = 'Welcome!';
}

echo $greeting;

?>
```

Copy above code and run it as **greeting.php**. You would see either **Good Morning** or **Welcome** depending on current time (**date()** function uses server's time and it may be different from your PC's time). Now just comment out echo statement like below and re-run the file.

```
//echo $greeting;
```

You would see nothing. So, if you want anything to be added to the output, you should use **echo()** for that.

Different Usage of echo()

```
echo $greeting;
echo ($greeting);
```

Both statements above produce same output. You can use **echo()** with or without brackets because it's a language construct and not a function. You would usually see **echo()** without brackets.

```
$myAge = 10;
echo 'My age is '.$myAge;
echo "My age is $myAge";
```

In two statements above, first one uses string concatenation while second one includes the variable within double quotes. Refer **Usage of Quotes** for more explanation on using variables inside quotes.

```
$intro = 'My age is ';
$myAge = 10;

echo $intro, $myAge;
```

Above statement would output **My age is 10**. This is an example to show that **echo()** (without brackets) can take more than one argument. However this way of usage is bit uncommon. Instead you would usually see string concatenation or variables within double quotes when it's required to output more than one variable using a single **echo()** statement.

Difference Between echo() and print()

print() is also a language construct and behaves almost same as **echo()**. Noticeable differences are **print()** can't take more than one argument and **print()** has a return value which is always 1.

Chapter 8 - Usage of Quotes

In PHP, single and double quotes are used to enclose strings. Following code would throw a PHP error since the string hasn't been enclosed by quotes.

```
$name = Robin Jackman;
```

It has to be enclosed by quotes like below.

```
$name = 'Robin Jackman';
```

While both single and double quotes can be used to enclose strings, there is a difference in how PHP interprets two methods. If you used double quotes then PHP would interpret variables and special characters (**\t**, **\n** etc) inside the string but if you used single quotes PHP would consider only the literal value of the whole string.

```
$name = 'Robin Jackman';
echo "Name is $name"; // Would print Name is Robin Jackman
echo 'Name is $name'; // Would print Name is $name
```

Usually single quotes are used when the whole statement is a simple string (like 'Name is Robin Jackman') and double quotes are used when the statement contains variables to be interpreted (like "Name is $name").

Chapter 9 - PHP Variables and Constants

In programming, a variable is a value holder. A variable can hold the same value or the value it holds can get changed during the runtime of a program.

```
<?php

$greeting = 'Welcome';

$name = 'Tom';
echo "$greeting $name";
echo '<br />';

$name = 'Jack';
echo "$greeting $name";
echo '<br />';

$name = 'Edward';
echo "$greeting $name";
echo '<br />';

?>
```

Run above script in a web browser and you would see following three lines.

Welcome Tom
Welcome Jack
Welcome Edward

In this script both **$greeting** and **$name** are variables. You can see that **$greeting** holds the same value throughout the script while value of **$name** gets changed. tags are for having line breaks in the web browser.

Naming Variables

In PHP, all variables should start with $ sign. The part after $ sign is called variable name.

- Variable names can only contain letters, numbers and underscores.
- A variable name should only start with a letter or an underscore.
- Variable names are case-sensitive. That means **$Greeting** and **$greeting** are two different variables.

Refer **PHP Coding Standards** to see some good practices in variable naming.

```
$firstName // Correct
$first_name // Correct
$firstName1 // Correct
$_firstName // Correct
$first-name // Wrong (Hyphen is not allowed)
$4firstName // Wrong (Starts with a number)
```

Arrays

All the variables considered so far could contain only one value at a time. Arrays provide a way to store a set of values under one variable. Refer the chapter **Arrays** to learn more about arrays in PHP.

Predefined Variables

PHP provides a set of predefined variables. Most of them are arrays. Their availability and values are based on the context. For example, **$_POST** contains all the values submitted via a web form that used **post** method. Refer PHP manual for more information on predefined variables[15].

Constants

As name describes, constants hold values that don't get changed during the runtime of a script. Same naming rules apply for constants except that $ sign is not used when defining constants. To give an emphasis to constants, it's a common practice to define constant names in upper-case.

```
define('SITE_URL', 'http://www.example.com');
echo SITE_URL; // Would print http://www.example.com
```

Once defined, a new value cannot be assigned to a constant.

[15] http://www.php.net/manual/en/reserved.variables.php

```
define('SITE_URL', 'http://www.google.com');
echo SITE_URL; // Would still print http://www.example.com
```

Chapter 10 - PHP Arrays

Arrays are variables which can hold more than one value at a time. Arrays are useful when you want to store a group of data. Consider an employee in a company called **Robin Jackman** with staff ID **NX-97246** and a basic salary of **5000** dollars. You can represent this employee in a PHP script as below.

```
$empName = 'Robin Jackman';
$empId = 'NX-97246';
$empSalary = 5000;
```

This is fine for one employee. But what if you want to represent more employees? You would need to define three variables for each. Then remembering and accessing them may not be that easy. You can address this problem by putting these values into an array.

```
$emp1 = array('Robin Jackman', 'NX-97246', 5000);
```

You can access and print elements of this array as below.

```
echo $emp1[0]; // Robin Jackman
echo $emp1[1]; // NX-97246
echo $emp1[2]; // 5000
```

If you want to define another employee, you can follow same approach.

```
$emp2 = array('Taylor Edward', 'NX-97281', 4700);
```

Now assume you want to have all the employees in one array. You can have it as below.

```
$employees = array($emp1, $emp2);
```

You can access elements of **$employees** array as below.

```
echo $employees[0][0]; // Robin Jackman
echo $employees[0][1]; // NX-97246
echo $employees[0][2]; // 5000
echo $employees[1][0]; // Taylor Edward
echo $employees[1][1]; // NX-97281
echo $employees[1][2]; // 4700
```

$employees is called a multi-dimensional array because it's an array of arrays (Its elements are arrays).

Indexed vs Associative Arrays

Each array element has two parts as key and value. In **$emp1[0]**, **0** (which is inside brackets) is the key and **Robin Jackman** is the value. Array keys let us locate elements. In **$emp1** array, keys are **0**, **1** and **2**. When array keys are numeric, we call them Indexed arrays. You have to remember that Indexed arrays start from key 0 (not from 1).

You could also define **$emp1** array as below and access its elements.

```
$emp1 = array('name'=>'Robin Jackman', 'id'=>'NX-97246', 'salary'=>5000);

echo $emp1['name']; // Robin Jackman
echo $emp1['id']; // NX-97246
echo $emp1['salary']; // 5000
```

When the keys of an array are strings we call it an Associative array. It's easy to remember and call array elements in Associative arrays because it's easy to remember names than remembering ordered numbers.

Assume that it's required to define another seven properties for employees. If it's an Indexed array you have to remember order and position of keys to call a particular element. But in Associative arrays, you just need to remember the name of the key. For example, you can easily remember and call **$emp1['salary']** than **$emp1[2]**.

Defining Arrays

You already saw a way to define arrays using **array()** function. You can also define an array by directly assigning a value to an element. Assume that **$emp1** array hasn't been defined yet.

```
$emp1[0] = 'Robin Jackman' // Makes $emp1 an array
```

For Associative arrays,

```
$emp1['name'] = 'Robin Jackman' // Makes $emp1 an array
```

For Indexed arrays, you can also initiate and add elements as below (using empty brackets).

```
$emp1[] = 'Robin Jackman';
$emp1[] = 'NX-97246';
$emp1[] = 5000;
```

In above case, numeric keys will automatically be assigned according to the order the elements were defined. You can access and print elements of this array just as you did previously.

```
echo $emp1[0]; // Robin Jackman
echo $emp1[1]; // NX-97246
echo $emp1[2]; // 5000
```

Array Functions

PHP provides a collection of built-in functions for working with arrays. Together with these functions, arrays have become a powerful feature of PHP.

```
echo count($emp1) // Will print 3, number of elements in the array
```

Assuming that **$emp1** was defined as an Associative array,

```
$keys = array_keys($emp1); // $keys become an array that contains keys of $emp1

echo $keys[0]; // name
echo $keys[1]; // id
echo $keys[2]; // salary
```

For all available array functions[16], refer PHP manual.

[16] http://php.net/manual/en/ref.array.php