

# **XML AND AJAX**

# **XML AND AJAX**

**1.1 Introduction to XML**

**1.2 Working with Basics of XML**

**1.3 Converting XML Documents in other formats**

**1.4 Working with XSLT**

**1.5 Working with Xpath, Xlink and Xpointer**

**1.6 XML Application**

**1.7 Overview of AJAX**

**1.8 AJAX components**

**1.9 Asynchronous Data Transfer with XML Http request**

**1.10 Implementing AJAX Frameworks**

**1.11 Consuming web services using AJAX**

# INTRODUCTION TO XML

## Introduction

XML, or Extensible Markup Language, is a markup language that you can use to create your own tags. It was created by the World Wide Web Consortium (W3C) to overcome the limitations of HTML, the Hypertext Markup Language that is the basis for all Web pages. Like HTML, XML is based on SGML -- Standard Generalized Markup Language.

# **WHY DO WE NEED XML?**

HTML is the most successful markup language of all time. You can view the simplest HTML tags on virtually any device, from palmtops to mainframes, and you can even convert HTML markup into voice and other formats with the right tools.

XML and HTML were designed with different goals:

1. XML is designed to carry data emphasizing on what type of data it is.
2. HTML is designed to display data emphasizing on how data looks
3. XML tags are not predefined like HTML tags.
4. HTML is a markup language whereas XML provides a framework for defining markup languages.
5. HTML is about displaying data,hence it is static whereas XML is about carrying information,which makes it dynamic.

There are three important characteristics of XML that make it useful in a variety of systems and solutions –

**XML is extensible** – XML allows you to create your own self-descriptive tags, or language, that suits your application.

**XML carries the data, does not present it** – XML allows you to store the data irrespective of how it will be presented.

**XML is a public standard** – XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

# XML USAGE

A short list of XML usage says it all –

1. XML can work behind the scene to simplify the creation of HTML documents for large web sites.
2. XML can be used to exchange the information between organizations and systems.
3. XML can be used for offloading and reloading of databases.
4. XML can be used to store and arrange the data, which can customize your data handling needs.
5. XML can easily be merged with style sheets to create almost any desired output.

# **XML DOES NOT USE PREDEFINED TAGS**

The XML language has no predefined tags.

The tags in the example above (like `<to>` and `<from>`) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

HTML works with predefined tags like `<p>`, `<h1>`, `<table>`, etc.

With XML, the author must define both the tags and the document structure.

# XML IS EXTENSIBLE

Most XML applications will work as expected even if new data is added (or removed).

Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <body>).

Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.

The way XML is constructed, older version of the application can still work:

```
<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

- XML Separates Data from HTML

When displaying data in HTML, you should not have to edit the HTML file when the data changes.

With XML, the data can be stored in separate XML files.

With a few lines of JavaScript code, you can read an XML file and update the data content of any HTML page.

## Books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>

    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>

    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>

    <book category="web">
        <title lang="en">XQuery Kick Start</title>
        <author>James McGovern</author>
        <author>Per Bothner</author>
        <author>Kurt Cagle</author>
        <author>James Linn</author>
        <author>Vaidyanathan Nagarajan</author>
        <year>2003</year>
        <price>49.99</price>
    </book>

    <book category="web" cover="paperback">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>

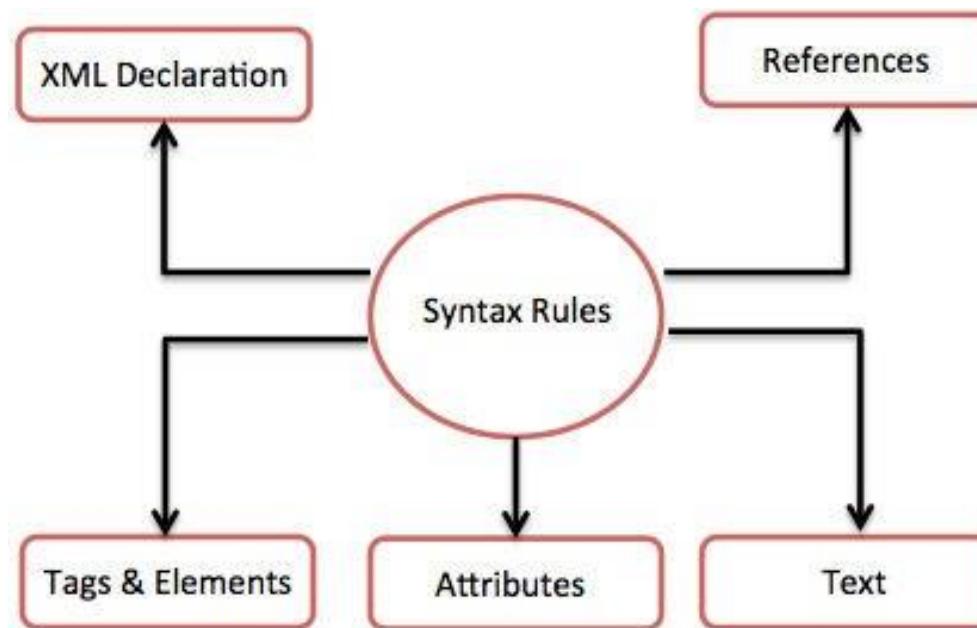
</bookstore>
```

# **IS XML A PROGRAMMING LANGUAGE?**

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instruct the computer to perform specific tasks. XML does not qualify to be a programming language as it does not perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

# XML - SYNTAX

The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



# XML DECLARATION

The XML document can optionally have an XML declaration. It is written as follows –

**<?xml version = "1.0" encoding = "UTF-8"?>** Where *version* is the XML version and *encoding* specifies the character encoding used in the document.

## Syntax Rules for XML Declaration

The XML declaration is case sensitive and must begin with "**<?xml>**" where "**xml**" is written in lower-case.

# **TAGS AND ELEMENTS**

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. The names of XML-elements are enclosed in triangular brackets < > as shown below –

**<element>**Syntax Rules for Tags and Elements

**Element Syntax** – Each XML-element needs to be closed either with start or with end elements as shown below –

**<element>....</element>**or in simple-cases, just this way –

**<element/>**

# XML ATTRIBUTES

An **attribute** specifies a single property for the element, using a name/value pair. An XML-element can have one or more attributes. For example –

**<a href = "http://www.CCT.COM/">CRIMSON!</a>** Here **href** is the attribute name and **http://www.CCT.COM/** is attribute value.

## Syntax Rules for XML Attributes

Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.

# XML REFERENCES

References usually allow you to add or include additional text or markup in an XML document. References always begin with the symbol "&" which is a reserved character and end with the symbol ";". XML has two types of references –

**Entity References** – An entity reference contains a name between the start and the end delimiters. For example &amp; where *amp* is *name*. The *name* refers to a predefined string of text and/or markup.

**Character References** – These contain references, such as &#65;, contains a hash mark ("#") followed by a number. The number always refers to the Unicode code of a character. In this case, 65 refers to alphabet "A".

# XML TEXT

The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case. To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.

Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.

# HOW TO CONVERT AN XML FILE

The best solution to converting an XML file to another format is to use one of the editors . The program that's creating the XML file is more than likely able to save the same file to a different format.

For example, a simple text editor, which can open a text document like XML, can usually save the file to another text-based format like TXT.

Here are some other free online XML converters that might be more useful for you:

XML to HTML

XML to CSV

XML to XSD

XML to PDF

Here are some free converters that convert *to* XML instead of *from* XML:

XLS/XLSX to XML

SQL to XML

CSV to XML

JSON to XML

# **WORKING WITH XSLT**

## **XSL**

XSL which stands for EXtensible Stylesheet Language. It is similar to XML as CSS is to HTML.

### **Need for XSL**

In case of HTML document, tags are predefined such as table, div, and span; and the browser knows how to add style to them and display those using CSS styles. But in case of XML documents, tags are not predefined. In order to understand and style an XML document, World Wide Web Consortium (W3C) developed XSL which can act as XML based Stylesheet Language. An XSL document specifies how a browser should render an XML document.

Following are the main parts of XSL –

**XSLT** – used to transform XML document into various other types of document.

**XPath** – used to navigate XML document.

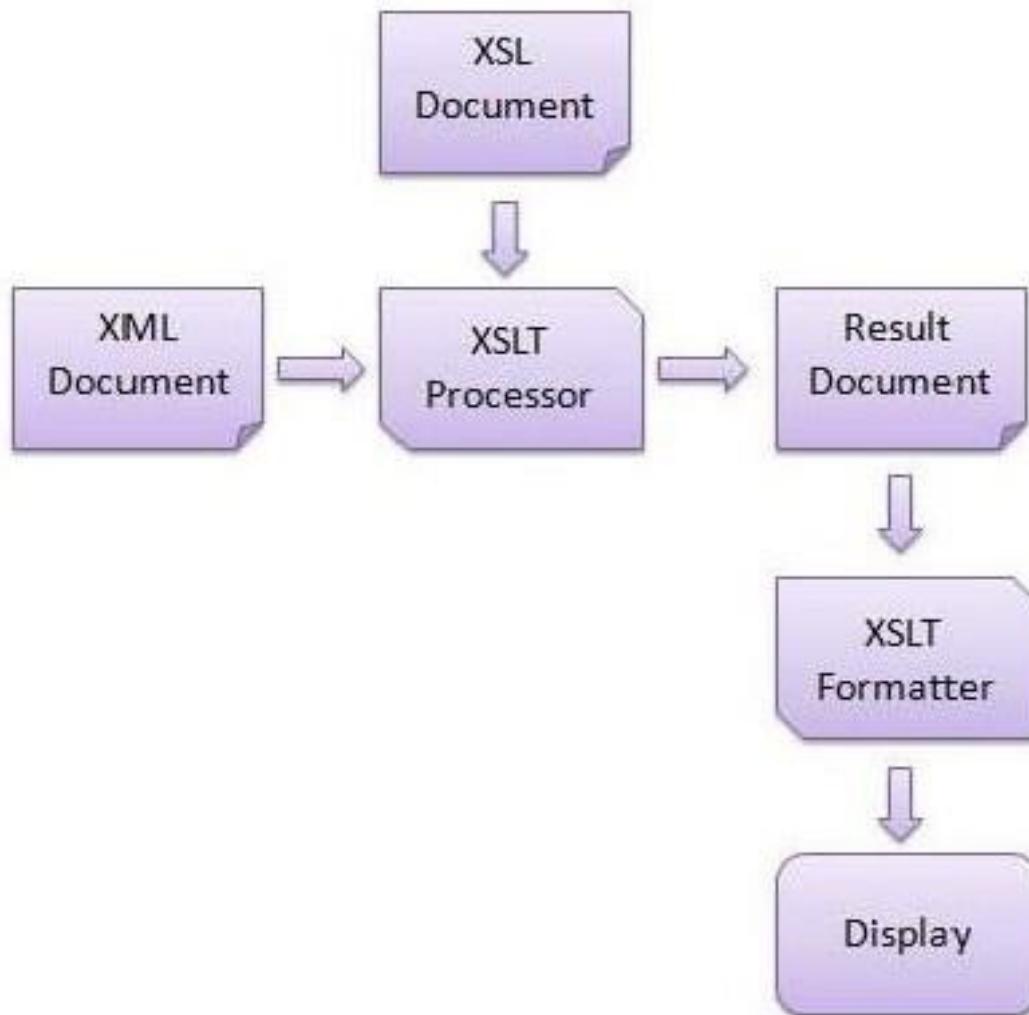
**XSL-FO** – used to format XML document.

# **WHAT IS XSLT**

XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically.

## How XSLT Works

An XSLT stylesheet is used to define the transformation rules to be applied on the target XML document. XSLT stylesheet is written in XML format. XSLT Processor takes the XSLT stylesheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format. This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.



## Advantages

Here are the advantages of using XSLT -

- Independent of programming. Transformations are written in a separate xsl file which is again an XML document.
- Output can be altered by simply modifying the transformations in xsl file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.

# XSLT SYNTAX

Let's suppose we have the following sample XML file, students.xml, which is required to be transformed into a well-formatted HTML document.

## **students.xml**

```
<?xml version = "1.0"?>  
  
<class>  
  
<student rollno = "393"> <firstname>Dinkar</firstname>  
<lastname>Kad</lastname> <nickname>Dinkar</nickname>  
<marks>85</marks> </student>  
  
<student rollno = "493"> <firstname>Vaneet</firstname>  
<lastname>Gupta</lastname> <nickname>Vinni</nickname>  
<marks>95</marks> </student>  
  
<student rollno = "593"> <firstname>Jasvir</firstname>  
<lastname>Singh</lastname> <nickname>Jazz</nickname>  
<marks>90</marks>  
  
</student>  
  
</class>
```

We need to define an XSLT style sheet document for the above XML document to meet the following criteria –

Page should have a title **Students**.

Page should have a table of student details.

Columns should have following headers: Roll No, First Name, Last Name, Nick Name, Marks

Table must contain details of the students accordingly.

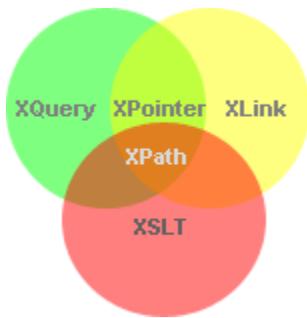
Step 1: Create XSLT document

Create an XSLT document to meet the above requirements, name it as students.xsl and save it in the same location where students.xml lies.

# WHAT IS XPATH?

XPath is a major element in the XSLT standard.

XPath can be used to navigate through elements and attributes in an XML document.



- XPath stands for XML Path Language
- XPath uses "path like" syntax to identify and navigate nodes in an XML document
- XPath contains over 200 built-in functions
- XPath is a major element in the XSLT standard
- XPath is a W3C recommendation

XPath is a language for addressing specific parts of a document. XPath models an XML document as a tree of nodes. An XPath expression is a mechanism for navigating through and selecting nodes from the document. An XPath expression is, in a way, analogous to an SQL query used to select records from a database.

There are various types of nodes, including element nodes, attribute nodes, and text nodes. XPath defines a way to compute a string-value for each type of node.

XPath defines a library of standard functions for working with strings, numbers and boolean expressions.

### **Examples:**

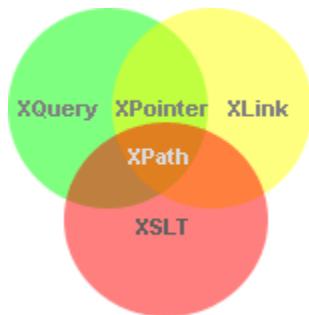
**child::\*** - Selects all children of the root node.

**.//name** - Selects all elements having the name "name", descendants of the current node.

**/catalog/cd[price>10.80]** - Selects all the cd elements that have a price element with a value larger than 10.80.

# XLINK

XLink is used to create hyperlinks in XML documents.



- XLink is used to create hyperlinks within XML documents
- Any element in an XML document can behave as a link
- With XLink, the links can be defined outside the linked files
- XLink is a W3C Recommendation

# XLINK BROWSER SUPPORT

There is no browser support for XLink in XML documents.

However, all major browsers support XLinks in SVG.

## XLink Syntax

In HTML, the `<a>` element defines a hyperlink. However, this is not how it works in XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what link elements will be called in XML documents.

Below is a simple example of how to use XLink to create links in an XML document:

```
<?xml version="1.0" encoding="UTF-8"?>

<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
    <homepage xlink:type="simple" xlink:href="https://www.w3schools.com">Visit W3Schools</homepage>
    <homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit W3C</homepage>
</homepages>
```

To get access to the XLink features we must declare the XLink namespace. The XLink namespace is:  
"http://www.w3.org/1999/xlink".

The xlink:type and the xlink:href attributes in the <homepage> elements come from the XLink namespace.

The xlink:type="simple" creates a simple "HTML-like" link (means "click here to go there").

The xlink:href attribute specifies the URL to link to.

## XLink Example

The following XML document contains XLink features:

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore xmlns:xlink="http://www.w3.org/1999/xlink">

<book title="Harry Potter">
  <description
    xlink:type="simple"
    xlink:href="/images/HPotter.gif"
    xlink:show="new">
    As his fifth year at Hogwarts School of Witchcraft and
    Wizardry approaches, 15-year-old Harry Potter is.....
  </description>
</book>

<book title="XQuery Kick Start">
  <description
    xlink:type="simple"
    xlink:href="/images/XQuery.gif"
    xlink:show="new">
    XQuery Kick Start delivers a concise introduction
    to the XQuery standard.....
  </description>
</book>

</bookstore>
```

## **Example explained:**

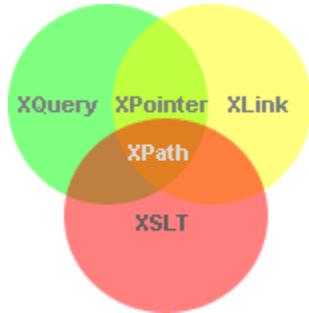
The XLink namespace is declared at the top of the document  
(`xmlns:xlink="http://www.w3.org/1999/xlink"`)

The `xlink:type="simple"` creates a simple "HTML-like" link

The `xlink:href` attribute specifies the URL to link to (in this case - an image)

The `xlink:show="new"` specifies that the link should open in a new window

# XPOINTER



- XPointer allows links to point to specific parts of an XML document
- XPointer uses XPath expressions to navigate in the XML document
- XPointer is a W3C Recommendation

# XPOINTER BROWSER SUPPORT

There is no browser support for XPointer. But XPointer is used in other XML languages.

## XPointer Example

In this example, we will use XPointer in conjunction with XLink to point to a specific part of another document.

We will start by looking at the target XML document (the document we are linking to):

```
<?xml version="1.0" encoding="UTF-8"?>

<dogbreeds>

<dog breed="Rottweiler" id="Rottweiler">
    <picture url="https://dog.com/rottweiler.gif" />
    <history>The Rottweiler's ancestors were probably Roman
    drover dogs.....</history>
    <temperament>Confident, bold, alert and imposing, the Rottweiler
    is a popular choice for its ability to protect....</temperament>
</dog>

<dog breed="FCRetriever" id="FCRetriever">
    <picture url="https://dog.com/fcretriever.gif" />
    <history>One of the earliest uses of retrieving dogs was to
    help fishermen retrieve fish from the water....</history>
    <temperament>The flat-coated retriever is a sweet, exuberant,
    lively dog that loves to play and retrieve....</temperament>
</dog>

</dogbreeds>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<mydogs xmlns:xlink="http://www.w3.org/1999/xlink">

<mydog>
    <description>
        Anton is my favorite dog. He has won a lot of.....
    </description>
    <fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#Rottweiler">
        Fact about Rottweiler
    </fact>
</mydog>

<mydog>
    <description>
        Pluto is the sweetest dog on earth.....
    </description>
    <fact xlink:type="simple" xlink:href="https://dog.com/dogbreeds.xml#FCRetriever">
        Fact about flat-coated Retriever
    </fact>
</mydog>

</mydogs>
```

# WHAT IS AJAX?

AJAX stands for **A**synchronous **J**ava**S**cript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.

Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.

With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.

## AJAX is Based on Open Standards

AJAX is based on the following open standards –

Browser-based presentation using HTML and Cascading Style Sheets (CSS).

Data is stored in XML format and fetched from the server.

Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.

JavaScript to make everything happen.

# THE ORIGINS OF AJAX

The key technical components of AJAX are:

- XHTML – a stricter, cleaner rendering of HTML into XML.
- CSS for marking up and adding styles.
- The Javascript Document Object Model (DOM) which allows the content, structure and style of a document to be dynamically accessed and updated.
- The XMLHttpRequest object which exchanges data asynchronously with the Web server reducing the need to continually fetch resources from the server.

Since data can be sent and retrieved without requiring the user to reload an entire Web page, small amounts of data can be transferred as and when required. Moreover, page elements can be dynamically refreshed at any level of granularity to reflect this. An AJAX application performs in a similar way to local applications residing on a user's machine, resulting in a user experience that may differ from traditional Web browsing.

Examples of AJAX usage include GMail and Flickr. It is largely due to these and other prominent sites that AJAX has become popular only relatively recently – the technology has been available for some time. One precursor was dynamic HTML (DHTML), which twinned HTML with CSS and JavaScript but suffered from cross-browser compatibility issues.

AJAX is not a technology, rather, the term refers to a proposed set of methods using a number of existing technologies. As yet, there is no firm AJAX standard, although the recent establishment of the Open AJAX Alliance [2], supported by major industry figures such as IBM and Google, suggests that one will become available soon.

# **ADVANTAGES AND DISADVANTAGES OF AJAX**

1. State can be maintained throughout a Web site.
2. A Web application can request only the content that needs to be updated, thus drastically reducing bandwidth usage and load time.
3. Users may perceive an AJAX-enabled application to be faster or more responsive.
4. Use of Ajax can reduce connections to the server, since scripts and style sheets only have to be requested once.

The disadvantages include:

1. Clicking the browser's "back" button may not function as expected.
2. Dynamic Web page updates make it difficult for a user to use bookmarks.
3. Browser does not support JavaScript or have JavaScript disabled, will not be able to use its functionality.
4. AJAX may provide a mechanism for attacks by malicious code

# AJAX - EXAMPLES

Here is a list of some famous web applications that make use of AJAX.

## **Google Maps**

A user can drag an entire map by using the mouse, rather than clicking on a button.

<https://maps.google.com/>

## **Google Suggest**

As you type, Google offers suggestions. Use the arrow keys to navigate the results.

<https://www.google.com/webhp?complete=1&hl=en>

## **Gmail**

Gmail is a webmail built on the idea that emails can be more intuitive, efficient, and useful.

<https://gmail.com/>

## **Yahoo Maps (new)**

Now it's even easier and more fun to get where you're going!

<https://maps.yahoo.com/>

# **ASYNCHRONOUS DATA TRANSFER WITH XMLHTTP REQUEST**

# IMPLEMENTING AJAX FRAMEWORK

The AJAX Framework is a cross browser framework that allows developers to quickly develop web pages that can call web services, web pages and other types of content through JavaScript without having to submit the current page. The AJAX Framework is:

Easy to use.

Works well with other existing frameworks.

Supports both "GET" and "POST".

Works with Internet Explorer 6+, Opera 8+, Safari 3+, Firefox 2+, Google's Chrome and other Mozilla based browsers.

# **CONSUMING WEB SERVICES USING AJAX**

# **INTRODUCTION TO PHP**

# **INTRODUCTION TO PHP**

- 2.1 Introduction to PHP Scripting Language**
- 2.2 PHP vs JSP vs ASP server-side programming**
- 2.3 Server-Side Scripting vs Client-Side Scripting**
- 2.4 Installing XAMPP or WAMP or other web server**
- 2.5 Setting server environment**
- 2.6 Configuration and Adjusting setting in PHP.ini and httpd.conf**
- 2.7 Running PHP scripts**
- 2.8 Formatting outputs**
- 2.9 Working with variables, global variables and constants**
- 2.10 Logical, concatenation, mathematical and relational operators**
- 2.11 Escape Sequences**

# WHAT IS PHP?

PHP stands for **Hypertext Pre-Processor**. PHP is a scripting language used to develop static and dynamic webpages and web applications. Here are a few important things you must know about PHP:

1. PHP is an Interpreted language, hence it doesn't need a compiler.
2. To run and execute PHP code, we need a Web server on which PHP must be installed.
3. PHP is a server side scripting language, which means that PHP is executed on the server and the result is sent to the browser in plain HTML.
4. PHP is open source and free.

# USES OF PHP

To further fortify your trust in PHP, here are a few applications of this amazing scripting language:

1. It can be used to **create Web applications** like Social Networks(Facebook, Digg), Blogs(Wordpress, Joomla), eCommerce websites(OpenCart, Magento etc.) etc.
2. **Command Line Scripting.** You can write PHP scripts to perform different operations on any machine, all you need is a PHP parser for this.
3. **Create Facebook applications** and easily integrate Facebook plugins in your website, using Facebook's PHP SDK.
4. **Sending Emails** or building email applications because PHP provides with a robust email sending function.
5. Wordpress is one of the most used blogging(CMS) platform in the World, and if you know PHP, you can try a hand in **Wordpress plugin development**.

# **WHAT SHOULD I KNOW BEFORE LEARNING PHP**

Before learning PHP you should have a basic understanding of:

**HTML** - it is strongly recommended to have a deep understanding of the HTML markup

**CSS** - it is recommended to know some CSS basics before learning PHP. This will help you on your way to developing cool and dynamic PHP websites

**JavaScript** - it would be nice of you to recognize and have a basic reading of the JavaScript programming language

**Programming Logic** - This is something that always helps. Any knowledge of another programming language will help you to learn and apply the logic to the new programming language, in this case PHP

# WHO IS USING PHP?

There are lots of big site using PHP:

Facebook

Wikipedia

Yahoo

Flickr

Wordpress

# **ADVANTAGES OF PHP OVER OTHER LANGUAGES**

There are several advantages why one should choose PHP.

1. **Easy to learn:** PHP is easy to learn and use. For beginner programmers who just started out in web development, PHP is often considered as the preferable choice of language to learn.
2. **Open source:** PHP is an open-source project. It is developed and maintained by a worldwide community of developers who make its source code freely available to download and use.
3. **Portability:** PHP runs on various platforms such as Microsoft Windows, Linux, Mac OS, etc. and it is compatible with almost all servers used today such Apache, IIS, etc.
4. **Fast Performance:** Scripts written in PHP usually execute or runs faster than those written in other scripting languages like ASP, Ruby, Python, Java, etc.
5. **Vast Community:** Since PHP is supported by the worldwide community, finding help or documentation related to PHP online is extremely easy.

## **Php Syntax**

```
<?php  
        echo 'Hello world';  
?>
```

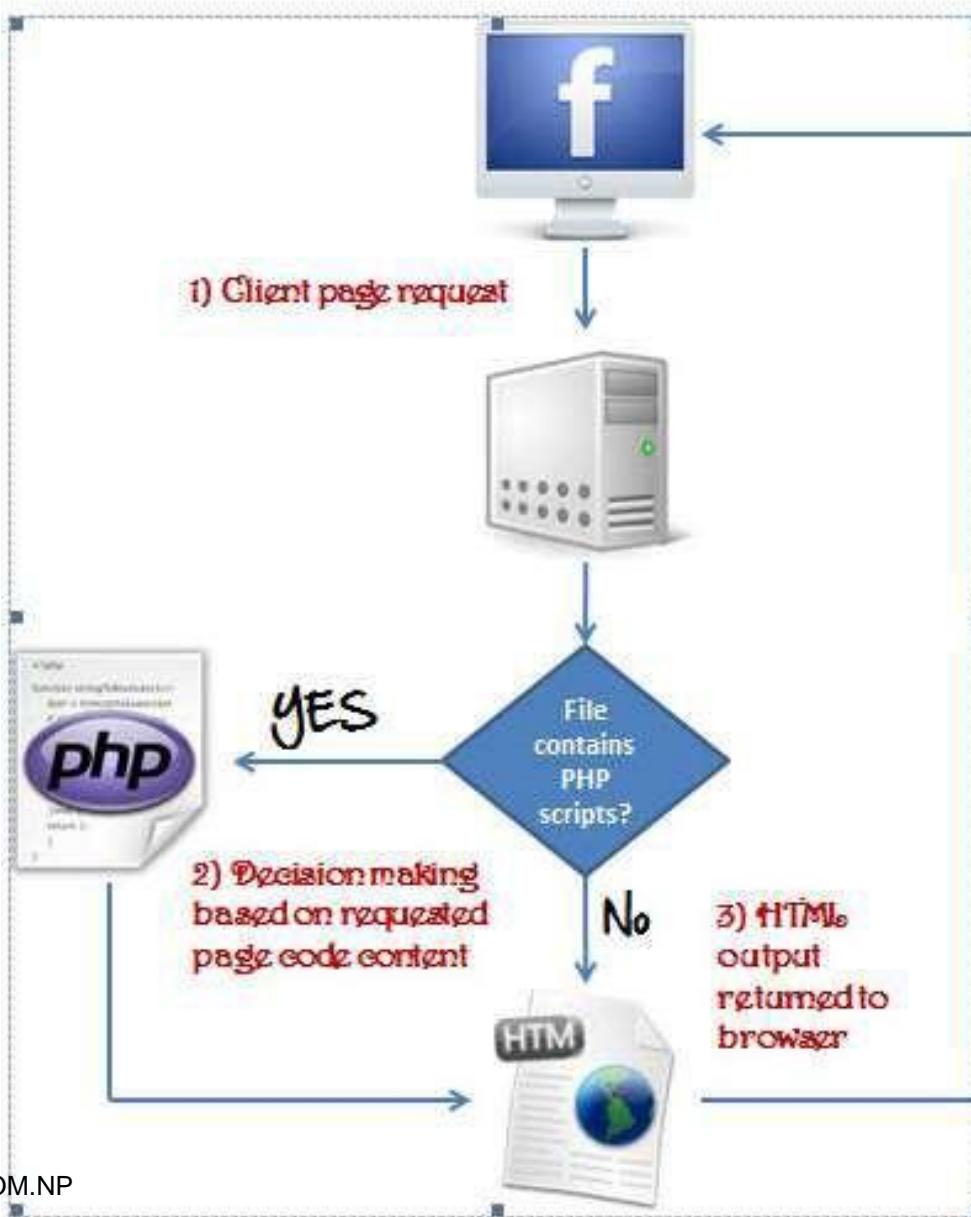
**A PHP file can also contain tags such as HTML and client side scripts such as JavaScript.**

**HTML is an added advantage when learning PHP Language. You can even learn PHP without knowing HTML but it's recommended you at least know the basics of HTML.**

**Database management systems DBMS for database powered applications.**

**For more advanced topics such as interactive applications and web services, you will need JavaScript and XML.**

**The flowchart diagram shown below illustrates the basic architecture of a PHP web application and how the server handles the requests.**

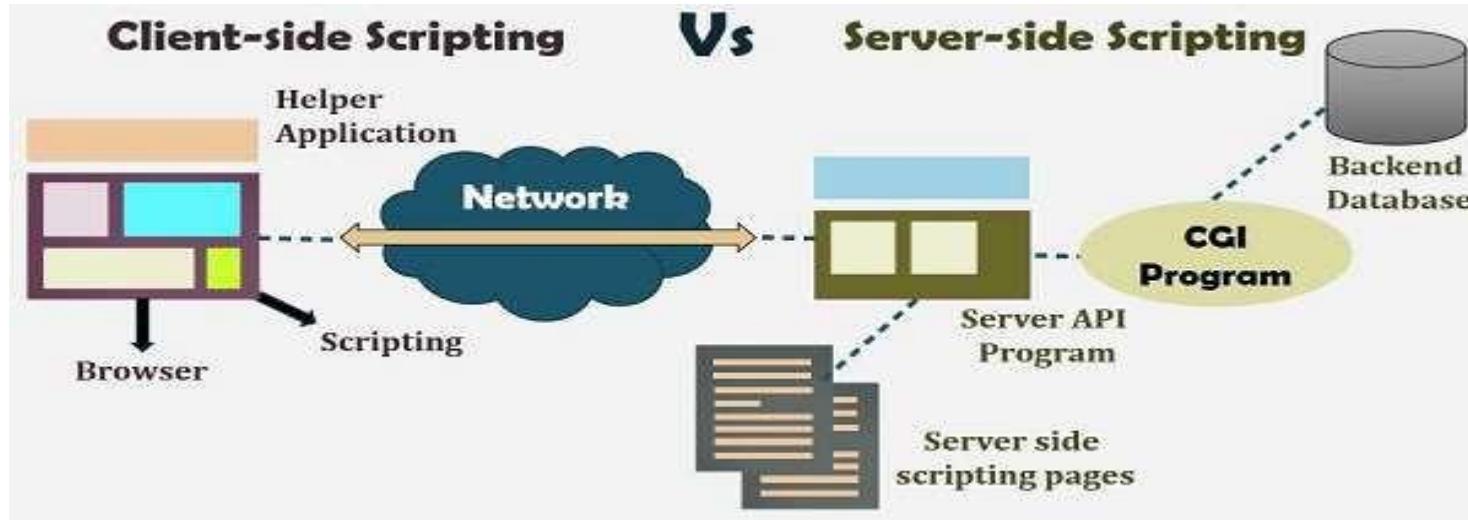


# **PHP VS JSP VS ASP SERVER-SIDE PROGRAMMING**

**The table below compares the various server side scripting languages with PHP**

FEATURE	PHP	ASP	JSP
Learning curve	short	Longer than PHP	Longer than PHP
Web hosting	Supported by almost all hosting servers	Needs dedicated server	Fairly supported
Open source	Yes	No	Yes
Web services support	Built in	Uses the .NET framework	Uses add on libraries
Integration with HTML	Easy	Fairly complex	Fairly complex
MySQL support	Native	Needs third party drivers	Needs third party drivers
Easily extended by other languages	Yes	No	Extended using Java classes and libraries.

# SERVER-SIDE SCRIPTING VS CLIENT-SIDE SCRIPTING



The scripts can be written in two forms, at the server end (back end) or at the client end (server end). The main difference between server-side scripting and client-side scripting is that the server side scripting involves server for its processing. On the other hand, client-side scripting requires browsers to run the scripts on the client machine but does not interact with the server while processing the client-side scripts.

<b>BASIS FOR COMPARISON</b>	<b>SERVER-SIDE SCRIPTING</b>	<b>CLIENT-SIDE SCRIPTING</b>
Basic	Works in the back end which could not be visible at the client end.	Works at the front end and script are visible among the users.
Processing	Requires server interaction.	Does not need interaction with the server.
Languages involved	PHP, ASP.net, Ruby on Rails, ColdFusion, Python, etcetera.	HTML, CSS, JavaScript, etc.
Affect	Could effectively customize the web pages and provide dynamic websites.	Can reduce the load to the server.
Security	Relatively secure.	Insecure

# **INSTALLING XAMPP OR WAMP OR OTHER WEB SERVER**

## Requirements for PHP

To run PHP scripts, we need the following services:

**PHP Parser:** To execute PHP scripts, PHP installation is required.

**Web Server:** Because PHP is mostly used to develop websites, hence most of its implementations comes bundled with Apache Web Server, which is required to host the application developed in PHP over HTTP.

**Database:** Any one database management system, which is generally MySQL, as PHP comes with a native support for MySQL

Now, you can install all the 3 services separately yourself, or you can simply download and install softwares which automatically installs all the above services.

The most popular such software package is **XAMPP**.

# WHAT IS XAMPP?

XAMPP stands for:

**X**: Cross Platform, as it supports all the modern operating systems like Windows, Mac OSX, Linux etc.

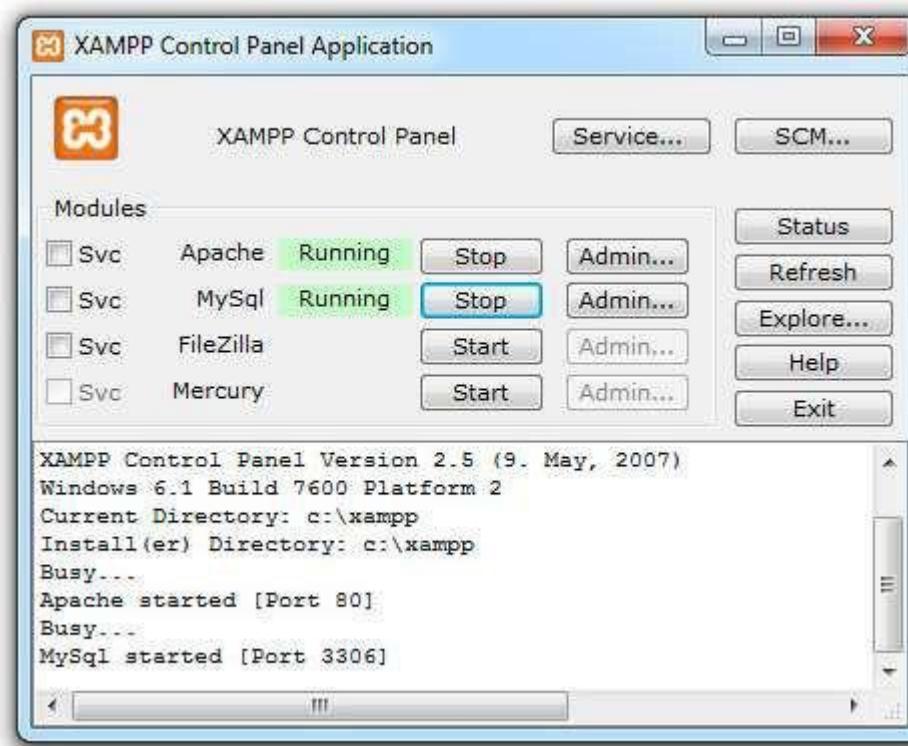
**A**: Apache Web Server

**M**: MySQL database management system.

**P**: PHP installation

**P**: Perl scripting language

You can easily download and install XAMPP from [this link](#). The installation process is simple and once installed you will see a small window like this, showing the status of various services which are running.



You can easily control, stop and restart, various services using the XAMPP Control Panel.

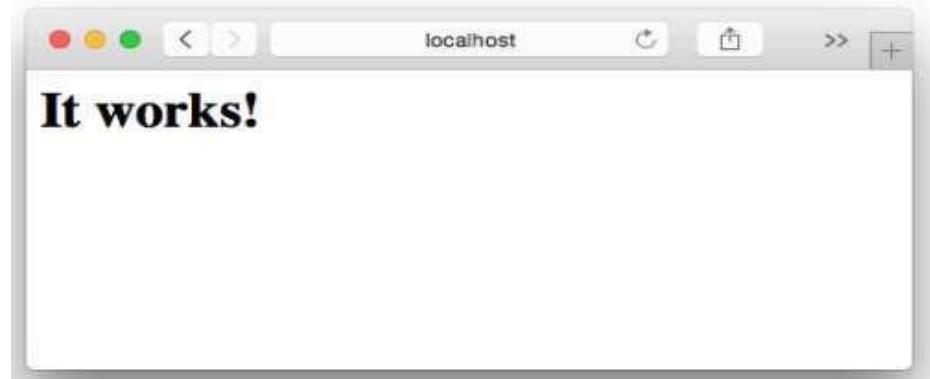
Upon successful installation, a folder with name **xampp** will be created in the C drive(by default). In the folder **xampp** there are many sub-folders like apache, cgi-bin, FileZillaFTP etc, but the most important sub-folders are:

**htdocs**: This is the folder in which we will keep all our PHP files.

**mysql**: This folder contains all the files for the MySQL database. By default the MySQL database runs on port number 3306.

**php**: This folder holds all the installation files for PHP. All the configurations for the current PHP installation is saved in **php.ini** file which is stored in this folder.

If everything seems fine and the apache server is running(check in the XAMPP control panel), open your **Web browser** and enter localhost in the address bar and hit enter. You will see the default page of Apache web server.



## Other Software Packages

XAMPP is not the only available package, although it is the most widely used one. Here are a few other Operating System specific options that you can download and install:

**WAMP**: It's for Windows (Window, Apache, MySQL and PHP)

**MAMP**: It's for Mac OSX (Macintosh, Apache, MySQL and PHP)

**LAMP**: It's for Linux (Linux, Apache, MySQL and PHP)

## IDE or Editor for writing PHP code

As PHP code is not compiled, you can even use a simple editor like Notepad to create PHP code files. But it's always good to have a nice looking Editor which can highlight the code, provide you suggestions while coding, and even analyze your code for syntax errors as you write it.

So here are a few good IDEs and Editors:

[Netbeans IDE](#)

[Eclipse IDE](#)

[PyStorm IDE](#)

[Atom Editor](#)

[Brackets Editor](#)

[Notepad ++](#)

[Sublime Text](#)

# FIRST PHP EXAMPLE

Hello World script in PHP

All the php code is written inside php code tags which is <?php and ?>. And the file with php code is saved with an extension .php. Here is a simple example:

```
<?php  
// code statements  
?>
```

Hence a simple **Hello, World!** program in PHP will be:

```
<?php  
echo "Hello, World!";  
?>  
output
```

Hello, World!

echo is a command used in PHP to display anything on screen.

If we want to include this php code in an HTML document, we can do so like this:

```
<!DOCTYPE>
<html>
<body>
<?
php echo "<h1>Hello, World!</h1>";
?>
</body>
</html>
```

Now copy the above code and paste it in you Code Editor or IDE and save the file with the name **hello\_world.php**

Now go to your C directory where XAMPP was installed, and open **xampp → htdocs** and create a new folder with name **studytonight** and put your php code file **hello\_world.php** in the **studytonight** folder.

Now visit the following link in your browser: **localhost/studytonight/hello\_world.php** and you would see **Hello, World!** written on the screen. Notice that *Hello, World!* is written as a heading because in the HTML code we specified that *Hello, World!* should be printed as a heading as put it inside the heading tag **<h1>**

```
<!DOCTYPE>
<html>
  <body>
    <?php
      echo "<h1>Hello, World!</h1>";
    ?>
    </body>
  </html>
```

php code file

PHP code is executed and the result in plain HTML is sent to the browser



```
<!DOCTYPE>
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

plain HTML code



Sent to browser over HTTP

# PHP SYNTAX RULES

Below we have listed down the main syntax rules that you must follow while writing php code.

1. All the php code in a php script should be enclosed within <?php and ?>, else it will not be considered as php code. Adding php code inside the PHP tags is known as **Escaping to php.<?php ... ?>**

Apart from the standard <?php and ?>, you can also use the **Short-open tags**:

<? ... ?> Or use the HTML script tags, like we do for adding javascript code in HTML document:

```
<script language="PHP"> ... </script>
```

**2.** Every expression in PHP ends with a **semicolon** ;

**3. Commenting PHP code:** Both single line and multi-line comments are supported in PHP. For single line comment, we can either use # or // before the comment line. For example,

<?php

# This is also a single line comment

# another line of comment

// This is a single line comment

echo "Example for Single line Comments";

?>

And for multi-line comments, we use /\* ... \*/. For example,

<?php

/\* This is also a single line comment another line of comment \*/

echo "Example for Multi-line Comments";

?>

**4. PHP is case sensitive**, which means that a variable **\$tiger** is not same as **\$Tiger**. Both of these, represent two different variables here. But all the predefined keywords and functions like if, else, echo etc are case insensitive.

```
<?php  
echo "Hello, World!";  
ECHO "Hello, World!";  
?>
```

OUTPUT

Hello, World! Hello, World!

## 5. PHP uses **curly braces** to define a code block.

```
<?php  
if($zero == 0)  
{  
echo "If condition satisfied";  
echo "This is a code block";  
}  
?>
```

Don't worry we will learn about if...else conditions in details, this is just to demonstrate the use of curly braces.

# PHP - VARIABLES

PHP variables are just a name for a value and you can simply use it like this `$var_name = value;`. They usually store sting characters and numeric values for later use in the programming block. Here is a simple example:

## PHP

```
<?php
$a      = "Welcome";
$b      = "Hello";
$first   = "John";
$last    = "Doe";
$c      = ", how are you today?";

echo "$a $first $last <br />"; // Returns: Welcome John Doe
echo "$b $first $c <br />";    // Returns: Hello John, How are you today?

$x=5;
$y=10.1;
echo $x+$y; // Returns: 15.1
?>
```

# WHAT IS A VARIABLES SCOPE TYPES IN PHP?

In PHP, the scope of a variable is the part of the script where a variable was defined and can be referenced. There are three types of scopes in PHP.

local - declared **within** a function and can only be accessed within that specific function

global - declared **outside** a function and can only be accessed outside a function. There is an exception for this when the *global* keyword is used. Keep on reading

static - declared **within** a function and maintaining his value in that function

## SPECIAL KEYWORDS FOR DECLARING VARIABLES SCOPES

There are cases when you declare a variable outside of a function and you want to use it inside and the other way around.

This is where the ***global*** keyword enters.

```
PHP
<?php
$x = 1;
$y = 2;

function sum() {
    global $x, $y;
    $result = $x + $y;
}

sum();
echo $result; // Outputs: 3
?>
```

On the other side **static** keyword is used to store a value of a variable declared inside a function, for consecutive calls. Normally all the local variables are reset when the function closes, unless static keyword is used.

```
<?php
function count_table() {
    static $x = 0;
    echo "Value of x is now $x <br />\n";
    return $x++;
}

while(count_table() < 10){
    // do some other stuff
}

/* Outputs:
Value of x is now 0
Value of x is now 1
Value of x is now 2
Value of x is now 3
Value of x is now 4
Value of x is now 5
Value of x is now 6
Value of x is now 7
Value of x is now 8
Value of x is now 9
Value of x is now 10
*/
?>
```

# RULES FOR CREATING VARIABLES IN PHP

Here we have a few basic rules that you must keep in mind while creating variables in PHP. All the rules are explained with help of simple examples.

A variable name will always start with a \$ sign, followed by the variable name.

A variable name should not start with a numeric value. It can either start with an alphabet or an underscore sign \_.

```
<?php
    $var = "I am a variable";
    $_var = 5;
    // Invalid variable name
    $7var = "I am a variable too";
?>
```

A variable name can only contain alphabets, numbers and underscore symbol \_.

Variable names in PHP are case-sensitive, which means \$love is not same as \$Love.

```
<?php
    $car = "Jaguar E-Pace";
    echo "My favorite car is $car";
    // below statement will give error
    echo "I love $Car";
?>
```

OUTPUT:

My favorite car is Jaguar E-Pace

Notice: Undefined variable: Car

# **PHP ECHO AND PRINT FUNCTIONS**

<https://www.studytonight.com/php/php-echo-and-print>

# **PHP VARIABLES**

**Variables are "containers" for storing information. A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).**

**Rules for PHP variables:**

**A variable starts with the \$ sign, followed by the name of the variable**

**A variable name must start with a letter or the underscore character**

**A variable name cannot start with a number**

**A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )**

**Variable names are case-sensitive (\$age and \$AGE are two different variables)**

# **ADD TWO NUMBER**

```
<html>  
<body>  
<?php  
$a =5 ;  
$b=6;  
$c=$a+$b;  
echo "The sum of $a and $b is $c";  
?>
```

```
</body>
```

```
</html>
```

**Output:**

**The sum of 5 and 6 is 11**

# **PHP VARIABLES SCOPE**

**In PHP, variables can be declared anywhere in the script.**

**The scope of a variable is the part of the script where the variable can be used.**

**PHP has three different variable scopes:**

**local**

**global**

**static**

# GLOBAL AND LOCAL SCOPE

A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function:

```
<html>
<body>
<?php
$x = 5; // global scope
function myTest() {
echo "<p>Variable x inside function is: $x</p>";
}
myTest();
echo "<p>Variable x outside function is: $x</p>";
?>
</body>
</html>
```

**Output:**  
WWW.ARJUN00.COM.NP

**A variable declared within a function has a LOCAL SCOPE and can only be accessed within that function:**

```
<html>
<body>
<?php
function myTest() {
    $x = 5; // local scope
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
</body>
</html>
```

**Output:**

**Variable x inside function is: 5**

**Variable x outside function is:**

# PHP THE STATIC KEYWORD

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job. To do this, use the static keyword when you first declare the variable:

```
<html>
<body>
<?php
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}
myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
```

Output:  
0  
1  
2

## PHP THE GLOBAL KEYWORD

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

```
<html>
<body>
<?php
$x = 5;
$y = 10;
function myTest() {
    global $x, $y;
    $y = $x + $y;
}
myTest(); // run function
echo $y; // output the new value for variable $y
```

**Output:**  
15

# PHP ECHO AND PRINT FUNCTIONS

## PHP Echo

`echo()` function is used to print or output one or more strings. We have specifically mentioned string here because, the syntax of the echo function is:

**`echo(string)`**

Although you can use `echo()` function to output anything, as PHP parser will automatically convert it into string type.

`echo` doesn't need parenthesis, although you can use parenthesis if you want.

```
<?php
```

```
echo "I am open";
```

```
echo ("I am enclosed in parenthesis");
```

```
?>
```

```
I am open I am enclosed in parenthesis
```

# EXAMPLE

Printing a basic sentence

We have already covered it multiple times, still:

```
<?php echo "I am a sentence"; ?>
```

I am a sentence

Printing multiple strings using comma ,

Here is how you can use multiple strings as parameters for echo.

```
<?php  
echo 'This','is','a','broken','sentence';  
?>
```

This is a broken sentence

Printing a multiline text(string)

```
<?php  
echo "This is a multiline sentence example";  
?>
```

This is a multiline sentence example

Printing a string variable

```
<?php  
$str = "I am a string variable";  
echo $str;  
?>
```

I am a string variable

## Printing a string variable with some text

Below we have explained how using **double quotes** and **single quotes** leads to different output when we use a string variable with some plain text in echo.

```
<?php  
$weird = "Stupid";  
echo "I am $weird";  
echo 'I am $weird'; ?>
```

I am Stupid

I am \$weird

As you can see, when we use **double quotes** the value of the string variable gets printed, while if we use **single quotes**, the variable is printed as it is.

## PHP Print

The PHP print is exactly the same as echo, with same syntax and same usage. Just replace echo with print in all the above examples, and they will work just fine.

# PHP - CONSTANTS

A constant is a type of variable that cannot be changed. A constant cannot be undefined and once his value has been set, it can not change his value along the script.

Constants are usually found in configuration files containing information such as database user and password, database name, site URL and other useful data.

This is how you define a constant in PHP:

```
<?php

// site name
define('SITE_NAME', 'my cool site name', flase);

// database conection
define('DB_INFO', array(
    'host'=> 'localhost',
    'usr' => 'my_user',
    'pass'=> 'my_database_pass',
    'db'   => 'my_database')
);

echo "Wellcome to " SITE_NAME;
echo "<br />"
var_dump(DB_INFO);
?>
```

Let's take a closer look at the examples above. We used the *define()* function to create the constant and with the following params:

- name - the defined name for our constant
- value - the defined value for our constant
- case-sensitive - this is an optional param and it is true by default. This means that our constant is case sensitive. Set it to false if you need it

## Naming constants

Just like a normal variable a valid constant name starts with a letter or underscore. On the other hand, a PHP constant doesn't need the "\$" sign before the constant name.

Although it is not a must, there is a convention that constant names to be written in uppercase letters. This makes it easy for the identification and distinguishes them from other variables when reviewing coding documents.

## Constants scope

Constants are automatically defined as globals and can be directly used anywhere around the script without further action.

```
<?php

// database connection
define('DB_INFO', array(
    'host'=> 'localhost',
    'usr' => 'my_user',
    'pass'=> 'my_database_pass',
    'db'   => 'my_database')
);

// simple function too print data
function nice_printing(){
    echo "<pre>";
    print_r(DB_INFO);
    echo "</pre>"
}

// function execution
nice_printing();
?>
```

## Check if a constant has been defined

Although we have not yet talked about the if/else tag yet, this is a good place to mention that constants have a specific function to check if they are defined.

For variables we will use *isSet()* while for constants we will use ***defined()***. Here is an example using it on a constant:

```
<?php
    define('DEBUG_LEVEL', 3);

    if(defined(DEBUG)) {
        // do something if debug is defined
    }
?>
```

# DATA TYPES IN PHP

PHP Data types specify the different types of data that are supported in PHP language. There are total 8 data types supported in PHP, which are categorized into 3 main types. They are:

**Scalar Types:** boolean, integer, float and string.

**Compound Types:** array and object.

**Special Types:** resource and NULL.

## PHP Boolean

A boolean data type can have two possible values, either **True** or **False**.

```
$a = true;
```

```
$b = false;
```

**NOTE:** Here the values true and false are not enclosed within quotes, because these are not strings.

## PHP Integer

An Integer data type is used to store any non-decimal numeric value within the range -2,147,483,648 to 2,147,483,647.

An integer value can be negative or positive, but it cannot have a decimal.

```
$x = -2671;
```

```
$y = 7007;
```

## PHP Float

Float data type is used to store any decimal numeric value.

A float(floating point) value can also be either negative or positive.

```
$a = -2671.01;
```

```
$b = 7007.70;
```

## PHP String

String data type in PHP and in general, is a sequence of characters(or anything, it can be numbers and special characters too) enclosed within quotes. You can use single or double quotes.

```
$str1 = "Hello";
```

```
$str2 = "What is your Roll No?";
```

```
$str3 = "4";
```

```
echo $str1;
```

```
echo "<br/>";
```

```
echo $str2;
```

```
echo "<br/>";
```

```
echo "Me: My Roll number is $str3";
```

### OUTPUT

Hello What is your Roll No?

Me: My Roll number is 4

## PHP NULL

NULL data type is a special data type which means **nothing**. It can only have one value, and that is NULL.

If you create any variable and do not assign any value to it, it will automatically have NULL stored in it.

Also, we can use NULL value to empty any variable.

**// holds a null value**

**\$a;**

**\$b = 7007.70;**

**// we can also assign null value**

**\$b = null;**

## PHP resource data type

A resource is a special variable, holding a reference to an external resource like a database connection for example.

```
PHP$conn =  
mysqli_connect(localhost,"root","admin","users");
```

# PHP OPERATORS

Operators are used to perform operations on PHP variables and simple values.

In PHP there are total 7 types of operators, they are:

Arithmetic Operators

Assignment Operators

Comparison Operators

Increment/Decrement Operators

Logical Operators

String Operators

Array Operators

There are a few additional operators as well like, Type operator, Bitwise operator, Execution operators etc.

Based on how these operators are used, they are categorised into 3 categories:

**Unary Operators:** Works on a single operand(variable or value).

**Binary Operators:** Works on two operands(variables or values).

**Ternary Operators:** Works on three operands.

## PHP Arithmetic Operators

These operators are used to perform basic arithmetic operations like addition, multiplication, division, etc.

Name	Operator	What does it do?	Example
Addition	<code>+</code>	It is used to perform normal addition.	<code>\$a + \$b</code>
Subtraction	<code>-</code>	It is used to perform normal subtraction.	<code>\$a - \$b</code>
Multiplication	<code>*</code>	It is used to perform multiplication.	<code>\$a * \$b</code>
Division	<code>/</code>	It is used to perform division.	<code>\$a / \$b</code>
Exponent	<code>**</code>	It returns the first operand raised to the power the second operand. $\$a \text{ } ** \text{ } \$b = \$a^{\$b}$	<code>\$a ** \$b</code>
Modulus(or, Remainder)	<code>%</code>	It returns the remainder of first operand divided by the second operand	<code>\$a \% \$b</code>

## PHP Assignment Operators

Assignment operators are used to assign values to variables, either as it is or after performing some arithmetic operation on it. The most basic assignment operator is **equal to=.**

Operator	Usage
=	<code>\$a = \$b</code> , will save the value of variable <code>\$b</code> to the variable <code>\$a</code>
+=	<code>\$a += \$b</code> is same as <code>\$a + \$b</code>
-=	<code>\$a -= \$b</code> is same as <code>\$a - \$b</code>
*=	<code>\$a *= \$b</code> is same as <code>\$a * \$b</code>
/=	<code>\$a /= \$b</code> is same as <code>\$a / \$b</code>
%=	<code>\$a %= \$b</code> is same as <code>\$a % \$b</code>

So basically, the assignment operator provides us with shorthand techniques to perform arithmetic operations.

# PHP Comparison Operators

As the name suggest, these are used to compare two values.

Name	Operator	What does it do?	Example
Equal	<code>==</code>	It returns <code>true</code> if left operand is equal to the right operand.	<code>\$a == \$b</code>
Identical	<code>===</code>	It returns <code>true</code> if left operand is equal to the right operand and they are of the same type.	<code>\$a === \$b</code>
Not Equal	<code>!=</code>	It returns <code>true</code> if left operand is not equal to the right operand.	<code>\$a != \$b</code>
Not Identical	<code>!==</code>	It returns <code>true</code> if left operand is not equal to the right operand, and they are of different type as well.	<code>\$a !== \$b</code>
Greater than	<code>&gt;</code>	It returns <code>true</code> if left operand is greater than the right operand.	<code>\$a &gt; \$b</code>
Less than	<code>&lt;</code>	It returns <code>true</code> if left operand is less than the right operand.	<code>\$a &lt; \$b</code>
Greater than or equal to	<code>&gt;=</code>	It returns <code>true</code> if left operand is greater than or equal to the right operand.	<code>\$a &gt;= \$b</code>
Less than or equal to	<code>&lt;=</code>	It returns <code>true</code> if left operand is less than or equal to the right operand.	<code>\$a &lt;= \$b</code>

## PHP Increment/Decrement Operators

These operators are **unary operators**, i.e they require only one operand.

Operator	Usage
<code>++\$a</code>	<b>Pre Increment</b> , It will first increment the operand by <code>1</code> (add one to it) and then use it or return it.
<code>\$a++</code>	<b>Post Increment</b> , It will first return the operand and then increment the operand by <code>1</code> .
<code>--\$b</code>	<b>Pre Decrement</b> , It will first decrement the operand by <code>1</code> (subtract one from it) and then use it or return it.
<code>\$b--</code>	<b>Post Decrement</b> , It will first return the operand and then decrement the operand by <code>1</code> .

These operators are very useful and handy when use loops or when we have simply increment any value by one in our program/script.

## PHP Logical Operators

Logical operators are generally used when any action depends on two or more conditions.

Name	Operator	What does it do?	Example
And	<code>and</code> or <code>&amp;&amp;</code>	It returns true if both the operands(or expressions) returns true.	<code>\$a &amp;&amp; \$b</code>
Or	<code>or</code> or <code>  </code>	It returns true if any one out of the two operands(or expressions) returns true, or both return true.	<code>\$a    \$b</code>
Xor	<code>xor</code>	It returns true if any one out of the two operands(or expressions) returns true, but not when both return true.	<code>\$a xor \$b</code>
Not	<code>!</code>	This is a <b>unary operator</b> . It returns true, if the operand(or expression) returns false.	<code>!\$a</code>

## PHP String Operators

String operators are used to perform operations on string. There are only two string operators, generally PHP built-in functions are used to perform various operations on strings,

Name	Operator	What does it do?	Example
Concatenation	. (a dot)	It is used to concatenate(join together) two strings.	\$a.\$b
Concatenation Assignment	.=	It is used to append one string to another.	\$a .= \$b

Here we have a simple example to demonstrate the usage of both the string operators.

```
<?php
    $a = "study";
    $b = "tonight";
    // concatenating $a and $b
    echo $a.$b;

    // appending $b to $a
    $a .= $b
    echo $a;
?>
```

# PHP Array Operators

These operators are used to compare arrays.

Name	Operator	What does it do?	Example
Equal	<code>==</code>	It returns <code>true</code> if both the arrays have same key/value pairs.	<code>\$a == \$b</code>
Identical	<code>===</code>	It returns <code>true</code> if both the arrays have same key/value pairs, in same order and of same type.	<code>\$a === \$b</code>
Not Equal	<code>!=</code>	It returns <code>true</code> if both the arrays are not same.	<code>\$a != \$b</code>
Not Identical	<code>!==</code>	It returns <code>true</code> if both the arrays are not identical, based on type of value etc.	<code>\$a !== \$b</code>
Union(Join)	<code>+</code>	It joins the arrays together	<code>\$a + \$b</code>

# ESCAPE SEQUENCE

Escape sequences are used for escaping a character during the string parsing. It is also used for giving special meaning to represent line breaks, tabs, alert and more. The escape sequences are interpolated into strings enclosed by double quotations or heredoc syntax.

Escape sequences are started with the escaping character backslash (\) followed by the character which may be an alphanumeric or a special character. If it is an alphanumeric character, it gives special meaning to represent the line breaks \n, carriage return \r and more. If it is a special character, then it will be considered as it is during the string parsing. For example, if we interpolate \\$var into a string then it will be taken as \$var. Without the escaping character (\), the \$var PHP variable is parsed to get its value.

## **Escape Sequence Example**

This code shows a simple PHP example program to distinguish the behavior of the escape sequences with alphanumeric and non-alpha numeric characters. In this example, I used the escape sequence \n to add a line break after label and I have used \\ to escape \ character and print it to the browser.

```
<?php  
echo "PHP Escape Sequences:\n Backslash \\ is used as an  
escaping character.";  
?>
```

## **Widely used Escape Sequences in PHP**

In this section, I have listed some of the widely used escape sequences and describe how they are used to escape the special character or to give meaning by combining with some alphanumeric characters.

\' – To escape ‘ within single quoted string.

\\" – To escape “ within double quoted string.

\\" – To escape the backslash.

\\$ – To escape \$.

\n – To add line breaks between string.

\t – To add tab space.

\r – For carriage return.

# **PROGRAMMING ON PHP**

# **PROGRAMMING ON PHP**

**3.1 Conditional Statements(If, If else, switch statement)**

**3.2 Iteration and Looping (do while, while, for loop, foreach loop)**

**3.3 Functions: Built-In and user-defined functions**

**3.4 String functions and pattern: String comparison, String Concatenation.**

**3.5 Array: Numeric Array, Associative Array**

**3.6 One dimensional and multi-dimensional array**

# CONDITIONAL STATEMENT

While writing programs/scripts, there will be scenarios where you would want to execute a particular statement only **if** some condition is satisfied. In such situations we use **Conditional statements**.

In PHP, there are 4 different types of Conditional Statements.

1. if statements
2. if...else statements
3. if...elseif...else statements
4. switch statement

## The if statement

When we want to execute some code when a condition is **true**, then we use if statement.

### Syntax:

```
if(condition)  
{ // code to be executed if 'condition' is true }
```

Here is a simple example,

```
<?php  
$age = 20;  
if($age <= 25)  
{  
echo "You are not allowed to consume alcohol";  
}  
?>
```

## The if...else statement

When we want to execute some code when a condition is **true**, and some other code when that condition is **false**, then we use the if...else pair.

### Syntax:

```
if(condition)
{ // code to be executed if 'condition' is true }
else { // code to be executed if 'condition' is false }
```

Here is a simple example,

```
<?php
$age = 26;
if($age <= 25)
{
echo "You are not allowed to consume alcohol";
}
else
{
echo "Enjoy the drinks";
}
?>
```

## The if...else...elseif statement

When we want to execute different code for different set of conditions, and we have more than 2 possible conditions, then we use if...elseif...else pair.

### Syntax:

```
if(condition1)
{ // code to be executed if 'condition1' is true }

elseif(condition2)
{ // code to be executed if 'condition2' is true }

else { /* code to be executed if both 'condition1' and 'condition2' are false */ }
```

Here is a simple example,

```
<?php
// speed in kmph
$speed = 110;
if($speed < 60)
{
    echo "Safe driving speed";
}
elseif($speed > 60 && $speed < 100)
{
    echo "You are burning extra fuel";
}
else
{
    // when speed is greater than 100
    echo "Its dangerous";
}
?>
```

## The switch statement

A switch statement is used to perform different actions, based on different conditions.

Using a switch statement, we can specify multiple conditions along with the code to be executed when that condition is **true**, thereby implementing a menu style program.

**syntax:**

```
switch(X)
{
    case value1:
        // execute this code when X=value1
        break;
    case value2:
        // execute this code when X=value2
        break;
    case value3:
        // execute this code when X=value3
        break;
    ...
    default:
        /* execute this when X matches none of the specified options */
}
```

In a switch statement, we provide the deciding factor which can be a variable or an expression to our switch statement, and then we specify the different **cases**, each with a **value**, a piece of code and a **break** statement.

break statement is specified to break the execution of the switch statement once the action related to a specified value has been performed.

If we do not specify a break statement, then all the switch cases, after the matched case, will get executed, until the next break statement.

The **default** statement is executed if no matching case is there.

```
<?php
$car = "Jaguar";
switch($car)
{
case "Audi":
echo "Audi is amazing";
break;
case "Mercedes":
echo "Mercedes is mindblowing";
break;
case "Jaguar":
echo "Jaguar is the best";
break;
default:
echo "$car is Ok";
}
?>
```

Jaguar is the best  
WWW.ARJUN00.COM.NP

# ITERATION AND LOOPING

As the name suggests, a **Loop** is used to execute something over and over again.

For example, if you want to display all the numbers from 1 to 1000, rather than using echo statement 1000 times, or specifying all the numbers in a single echo statement with newline character \n, we can just use a loop, which will run for 1000 times and every time it will display a number, starting from 1, incrementing the number after each iteration or cycle.

In a Loop, we generally specify a condition or a LIMIT up till which the loop will execute, because if we don't specify such a condition, how will we specify when the loop should end and not go on for infinite time.

In a Loop, we generally specify a condition or a LIMIT to stop the loop's execution.



After each cycle, execution returns to check the condition, if x is still less than 1000, it again enters the loop



```
x = 1;
Some Loop(till x is 1000)
{
    echo x;
    // increment value of x by 1
    x++;
}
```

The code inside the Loop block is executed every time.

## PHP while Loop

The while loop in PHP has two components, one is a condition and other is the code to be executed. It executes the given code until the specified condition is true.

### Syntax:

```
<?php  
while(condition)  
{  
/* execute this code till the condition is true */  
}  
?>
```

For example, Let's print numbers from 1 to 10.

```
<?php
$a = 1;
while($a <= 10)
{
echo "$a | ";
$a++;
// incrementing value of a by 1
}
?>
```

OUTPUT

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

## PHP do...while Loop

The do...while loop is a little different from all the loops in PHP because it will execute at least one time, even if the condition is false, can you guess how? Well because the condition is checked after the loop's execution, hence the first time when the condition is checked, the loop has already executed once.

### Syntax:

```
<?php
```

**Do**

```
{
```

```
/* execute this code till the condition is true */
```

```
}
```

```
while(condition)
```

```
?>
```

Let's implement the above example using do...while loop,

```
<?php  
$a = 1;  
do  
{ echo "$a | ";  
$a++;  
// incrementing value of a by 1  
}  
while($a <= 10)  
?>
```

OUTPUT

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

Let's take another example where even if the condition is **false**, still the loop will be executed once.

```
<?php  
$a = 11;  
do  
{  
echo $a;  
$a++;  
// incrementing value of a by 1  
}  
while($a <= 10)  
?>  
OUTPUT  
11
```

As we can see clearly, that the condition in the above do...while loop will return **false** because value of variable \$a is **11** and as per the condition the loop should be executed only if the value of \$a is less than or equal to **10**.

## PHP for Loop

The for loop in PHP doesn't work like while or do...while loop, in case of for loop, we have to declare beforehand how many times we want the loop to run.

**Syntax:**

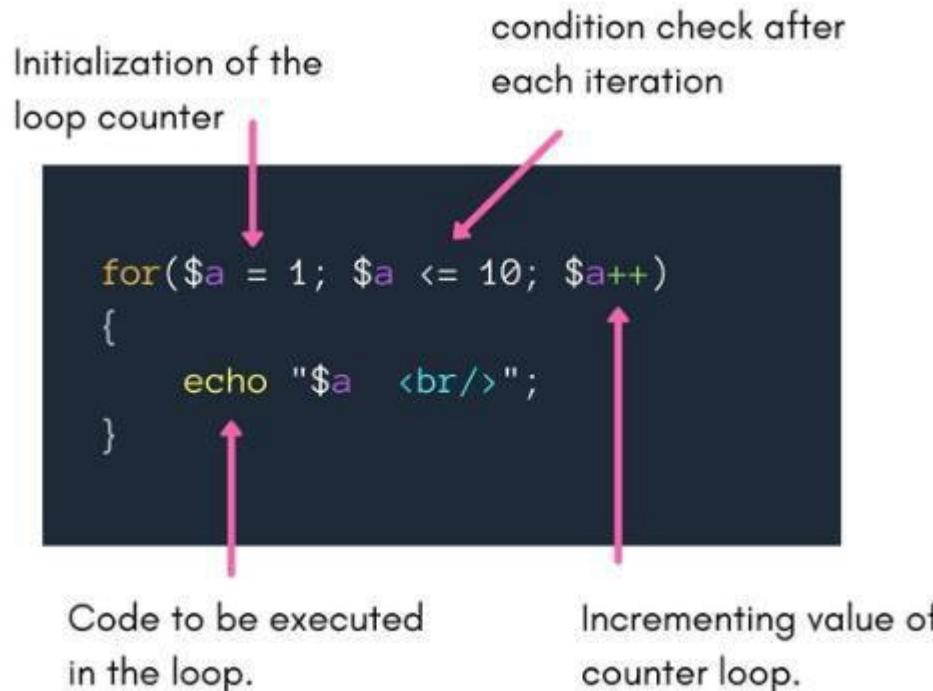
```
<?php  
for(initialization; condition; increment/decrement)  
{  
/* execute this code till the condition is true */  
}  
?>
```

The parameters used have following meaning:

**initialization:** Here we initialize a variable with some value. This variable acts as the loop counter.

**condition:** Here we define the condition which is checked after each iteration/cycle of the loop. If the condition returns **true**, then only the loop is executed.

**increment/decrement:** Here we increment or decrement the loop counter as per the requirements.



print numbers from 1 to 10, this time we will be using the for loop.

```
<?php  
for($a = 1; $a <= 10; $a++)  
{  
    echo "$a <br/>";  
}  
?>  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

## Nested for Loops

We can also use a for loop inside another for loop. Here is a simple example of nested for loops.

```
<?php  
for($a = 0; $a <= 2; $a++)  
{  
    for($b = 0; $b <= 2; $b++)  
    {  
        echo "$b $a ";  
    }  
}  
?  
0  
0  
1  
0  
2  
1  
2  
2  
2  
0 0 1 1 1 2 1 0 2 1 2 2 2
```

## foreach Loop

The foreach loop in PHP is used to access key-value pairs of an array. This loop only works with arrays and you do not have to initialise any loop counter or set any condition for exiting from the loop, everything is done implicitly(internally) by the loop.

### Syntax:

```
<?php  
foreach($array as $var)  
{  
/* execute this code for all the array elements $var will represent  
all the array elements starting from first element, one by one */  
}  
?>
```

Here is a simple example.

```
<?php  
$array = array("Jaguar", "Audi", "Mercedes", "BMW"); foreach($array  
as $var)
```

```
{  
echo "$var <br/>";  
}  
?>
```

OUTPUT

Jaguar

Audi

Mercedes

BMW

## break statement

We have already seen and used break statements in the switch conditional statements.

To recall, in switch code blocks, we used break statement to break out of the switch block when a valid case block gets executed.

Let's see an example for simple switch code:

```
<?php
$a = 1;
switch($a)
{
case 1: echo "This is case 1";
break;
case 2: echo "This is case 2";
break;
default: echo "This is default case";
}
?>
```

OUTPUT

This is case 1

# **FUNCTIONS: BUILT-IN AND USER-DEFINED FUNCTIONS**

A **Function** is nothing but a 'block of statements' which generally performs a specific task and can be used repeatedly in our program. This 'block of statements' is also given a **name** so that whenever we want to use it in our program/script, we can call it by its **name**.

In PHP there are thousands of built-in functions which we can directly use in our program/script.

PHP also supports **user defined functions**, where we can define our own functions.

A function doesn't execute when its defined, it executed when it is called.

## PHP User Defined Functions

Let's understand how we can define our own functions in our program and use those functions.

### Syntax:

```
<?php  
function function_name()  
{  
// function code statements  
}  
?>
```

## Few Rules to name Functions

1. A **function name** can only contain alphabets, numbers and underscores. No other special character is allowed.
2. The name should start with either an alphabet or an underscore. It should not start with a number.
3. And last but not least, function names are not case-sensitive.
4. The opening curly brace { after the function name marks the start of the function code, and the closing curly brace } marks the end of function code.

write a very simple function which will display a simple "Merry Christmas and a Very Happy New Year" message. This script can actually be very useful when you have to send festive emails to every friend of yours and you have to write the same message in all of them.

```
<?php
// defining the function
function greetings()
{
    echo "Merry Christmas and a Very Happy New Year";
}

echo "Hey Martha <br/>";
// calling the function
greetings();

// next line
echo "<br/>";

echo "Hey Jon <br/>";
// calling the function again
greetings();

?>
```

## Advantages of User-defined Functions

As we have already seen a simple example of using a function above, you must have understood how time-saving it can be for large programs. Here are a few advantages of using functions for you:

**Reusable Code:** As it's clear from the example above, you write a function once and can use it for a thousand times in your program.

**Less Code Repetition:** In the example above we just had one line of code in the function, but what if we have 10 lines of code. So rather than repeating all those lines of code over and over again, we can just create a function for them and simply call the function.

**Easy to Understand:** Using functions in your program, makes the code more readable and easy to understand.

## PHP Function Arguments

We can even pass data to a function, which can be used inside the function block. This is done using arguments. An argument is nothing but a variable.

Arguments are specified after the function name, in parentheses, separated by comma. When we define a function, we must define the number of arguments it will accept and only that much arguments can be passed while calling the function.

### Syntax:

```
<?php
```

```
/* we can have as many arguments as we want to have in a function */  
function function_name(argument1, argument2)
```

```
{
```

```
// function code statements
```

```
}
```

```
?>
```

```
<?php
// defining the function with argument function greetings($festival)
{
echo "Wish you a very Happy $festival";
}
echo "Hey Jai <br/>";
// calling the function
greetings("Diwali");
// next line
echo "<br/>";
echo "Hey Jon <br/>";
// calling the function again
greetings("New Year");
?>
```

## PHP Default Function Arguments

Sometimes function arguments play an important role in the function code execution. In such cases, if a user forgets to provide the argument while calling the function, it might lead to some error.

To avoid such errors, we can provide a default value for the arguments which is used when no value is provided for the argument when the function is called.

Let's take an example.

```
<?php
// defining the function with default argument function greetings($festival = "Life")
{
echo "Wish you a very Happy $festival";
}
echo "Hey Jai <br/>";
// calling the function with an argument
greetings("Diwali");
// next line
echo "<br/>";
echo "Hey Jon <br/>";
// and without an argument
greetings();
?>
```

## PHP Function Returning Values

Yes, functions can even return results. When we have functions which are defined to perform some mathematical operation etc, we would want to output the result of the operation, so we return the result.

return statement is used to return any variable or value from a function in PHP.

Let's see an example.

```
<?php  
function add($a, $b)  
{  
    $sum = $a + $b;  
    // returning the result  
    return $sum;  
}  
echo "5 + 10 = " . add(5, 10) . "";  
?>
```

OUTPUT

5 + 10 = 15

## PHP Function Overloading

Function overloading allows you to have multiple different variants of one function, differentiated by the number and type of arguments they take.

For example, we defined the function add() which takes two arguments, and return the sum of those two. What if we wish to provide support for adding 3 numbers.

To tackle such situations, what we can do is, we can define two different variants of the function add(), one which takes in 2 arguments and another which accepts 3 arguments. This is called **Function Overloading**.

Let's take an example,

```
<?php
// add function with 2 arguments
function add($a, $b)
{
    $sum = $a + $b;
    // returning the result return $sum;
}

// overloaded add function with 3 arguments
function add($a, $b, $c)
{
    $sum = $a + $b + $c;
    // returning the result
    return $sum;
}

// calling add with 2 arguments
echo "5 + 10 = " . add(5, 10) . "<br/>";
// calling add with 3 arguments
echo "5 + 10 + 15 = " .add(5, 10, 15) . "<br/>";
?>
```

## **STRING FUNCTIONS AND PATTERN: STRING COMPARISON, STRING CONCATENATION.**

A string in PHP is a simple character concatenation like "Hello world!", "Lorem Ipsum..." etc. Strings in Php are also a collection of alphanumeric characters, enclosed in single quotes for simple string data and double quotes for complex string data.

```
<?php  
$a = 'Simple string value';  
$b = "Another string value using double quotes.";  
echo $a;  
echo "<br />";  
echo $b;  
?>
```

We can define a string using **double quotes** as well as **single quotes**.

But, what if the string has a single or double quote in it, then what?

```
<?php  
$str1 = "This is "a" String";  
$str2 = 'This is also 'a' String';  
?>
```

Both the above string definitions will result in **error**, as we cannot include a double quote in the string, if the string is defined inside double quotes.

But it will work, if the string contains double quotes, but it is defined using single quotes and vice versa, for example,

```
<?php  
// string inside double quotes, with a single quote  
$str1 = "This is 'a' String"; // string inside single quotes, with  
a double quote  
$str2 = 'This is also "a" String';  
?>
```

There is one more simple technique to deal with the quotes problem, it is known as **Escaping Special Character**, which can be done using a **backslash**.

Let's take an example,

```
<?php  
// escaping double quote using backslash  
$str1 = "I'll handle this.";  
// escaping single quote using backslash  
$str2 = 'I'll handle this.';  
echo $str1;  
echo "\n"; // new line  
echo $str2;  
?>
```

# STRING FUNCTIONS

In general practice, using the right string function will save you a lot of time as they are pre-defined in PHP libraries and all you have to do is call them to use them.

## Commonly used PHP 5 String Functions

Below we have a list of some commonly used string functions in PHP:

**strlen(\$str)**

This function returns the length of the string or the number of characters in the string including whitespaces

```
<?php  
$str = "Welcome to Studytonight";  
// using strlen in echo method  
echo "Length of the string is: ". strlen($str);  
?>
```

**OUTPUT**

Length of the string is: 23

```
str_word_count($str)
```

This function returns the number of words in the string. This function comes in handly in form field validation for some simple validations.

```
<?php  
$str = "Welcome to Studytonight";  
// using str_word_count in echo method  
echo "Number of words in the string are: ".  
str_word_count($str);  
?>
```

OUTPUT

Number of words in the string are: 3

`strrev($str)`

This function is used to reverse a string.

Let's take an example and see,

```
<?php  
$str = "Welcome to Studytonight";  
// using strrev in echo method  
echo "Reverse: ". strrev($str);  
?>
```

OUTPUT

Reverse: thginotyduS ot emocleW

`strpos($str, $text)`

This function is used to find the position of any text/word in a given string. Just like an array, string also assign index value to the characters stored in it, starting from zero.

```
<?php  
$str = "Welcome to Studytonight";  
// using strpos in echo method  
echo "Position of 'Studytonight' in string: ". strpos($str,  
'Studytonight');  
?>
```

OUTPUT

Position of 'Studytonight' in string: 11

## Concatenation of two strings in PHP

There are two string operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side.

Examples :

**Input : string1: Hello**

**string2 : World!**

**Output : HelloWorld!**

**Input : string1: geeksfor**

**string2: geeks**

**Output : geeksforgeeks**

Output:  
HelloWorld!

```
// First String
$a = 'Hello';

// Second String
$b = 'World!';

// Concatenation Of String
$c = $a.$b;

// print Concatenate String
echo " $c \n";
?>
```

Output:  
John Carter!

```
// First String
$fname = 'John';

// Second String
$lname = 'Carter!';

// Concatenation Of String
$c = $fname." ".$lname;

// print Concatenate String
echo " $c \n";
?>
```

Output:  
HelloWorld!

```
// First String
$a = 'Hello';

// now $a contains "HelloWorld!"
$a. = "World!";

// Print The String $a
echo " $a \n";
?>
```

# COMPARISON OF STRING

## **== operator**

The most common way you will see of comparing two strings is simply by using the == operator if the two strings are equal to each other then it returns true.

**// Using the == operator, Strings match is printed**

```
if('string1' == 'string1')
{
echo 'Strings match.';
}
else
{
echo 'Strings do not match.';
}
```

```
// Using the == operator, Strings do not match is printed
if('string1' == 'STRING1')

{
echo 'Strings match.');

}
Else

{
echo 'Strings do not match.');

}
```

## **strcmp Function**

Another way to compare strings is to use the PHP function **strcmp**, this is a binary safe string comparison function that will return a 0 if the strings match.

```
// strcmp function, Strings match is printed if(strcmp('string1',
'string1') == 0)

{
echo 'Strings match.';

}
else
{
echo 'Strings do not match.';

}
```

## **strcasecmp Function**

The previous examples will not allow you to compare different case strings, the following function will allow you to compare case insensitive strings.

```
// Both strings will match
if(strcasecmp('string1', 'string1') == 0)
{
echo 'Strings match.';
}

else
{
echo 'Strings do not match.';
}

// Both strings will match even with different case
if(strcasecmp('string1', 'String1') == 0)
{
echo 'Strings match.';
}

else
{ echo 'Strings do not match.';
}

// Both strings will match even with different case
if(strcasecmp('string1', 'STRING1') == 0)
{
echo 'Strings match.';
}

else
{
echo 'Strings do not match.';
}
```

# **ARRAY: NUMERIC ARRAY, ASSOCIATIVE ARRAY**

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data.

An array is created using an **array()** function in PHP.

There are basically three types of arrays in PHP:

**Indexed or Numeric Arrays:** An array with a numeric index where values are stored linearly.

**Associative Arrays:** An array with a string index where instead of linear storage, each value can be assigned a specific key.

**Multidimensional Arrays:** An array which contains single or multiple array within it and can be accessed via multiple indices.

## **Indexed or Numeric Arrays**

These type of arrays can be used to store any type of elements, but an index is always a number. By default, the index starts at zero. These arrays can be created in two different ways as shown in the following example:

```
<?php
```

```
// One way to create an indexed array  
$name_one = array("Zack", "Anthony", "Ram", "Salim", "Raghav");
```

```
// Accessing the elements directly  
echo "Accessing the 1st array elements directly:\n";  
echo $name_one[2], "\n";  
echo $name_one[0], "\n";  
echo $name_one[4], "\n";
```

Output:

```
// Second way to create an indexed array  
$name_two[0] = "ZACK";  
$name_two[1] = "ANTHONY";  
$name_two[2] = "RAM";  
$name_two[3] = "SALIM";  
$name_two[4] = "RAGHAV";
```

```
// Accessing the elements directly  
echo "Accessing the 2nd array elements directly:\n";  
echo $name_two[2], "\n";  
echo $name_two[0], "\n";  
echo $name_two[4], "\n";
```

```
Accessing the 1st array elements directly:  
Ram  
Zack  
Raghav  
Accessing the 2nd array elements directly:  
RAM  
ZACK  
RAGHAV
```

```
?>
```

## **Associative Arrays**

These type of arrays are similar to the indexed arrays but instead of linear storage, every value can be assigned with a user-defined key of string type.

Example:

```
<?php
```

```
// One way to create an associative array
```

```
$name_one = array("Zack"=>"Zara", "Anthony"=>"Any",  
    "Ram"=>"Rani", "Salim"=>"Sara",  
    "Raghav"=>"Ravina");
```

```
// Second way to create an associative array
```

```
$name_two["zack"] = "zara";  
$name_two["anthony"] = "any";  
$name_two["ram"] = "rani";  
$name_two["salim"] = "sara";  
$name_two["raghav"] = "ravina";
```

```
// Accessing the elements directly
```

```
echo "Accessing the elements directly:\n";  
echo $name_two["zack"], "\n";  
echo $name_two["salim"], "\n";  
echo $name_two["anthony"], "\n";  
echo $name_one["Ram"], "\n";  
echo $name_one["Raghav"], "\n";
```

```
?>
```

Output:

Accessing the elements directly:

zara

sara

any

Rani

Ravina

Associative arrays are used to store key value pairs. For example, to store the marks of different subject of a student in an array, a numerically indexed array would not be the best choice. Instead, we could use the respective subject's names as the keys in our associative array, and the value would be their respective marks gained.

Example:

Here **array()** function is used to create associative array.

```
<?php  
/* First method to create an associate array. */  
$student_one = array("Maths"=>95, "Physics"=>90,  
"Chemistry"=>96, "English"=>93,  
"Computer"=>98);
```

```
/* Second method to create an associate array. */
```

```
$student_two["Maths"] = 95;  
$student_two["Physics"] = 90;  
$student_two["Chemistry"] = 96;  
$student_two["English"] = 93;  
$student_two["Computer"] = 98;
```

```
/* Accessing the elements directly */
```

```
echo "Marks for student one is:\n";  
echo "Maths:" . $student_two["Maths"], "\n";  
echo "Physics:" . $student_two["Physics"], "\n";  
echo "Chemistry:" . $student_two["Chemistry"], "\n";  
echo "English:" . $student_one["English"], "\n";  
echo "Computer:" . $student_one["Computer"], "\n";  
?>
```

**Output:**

```
Marks for student one is:  
Maths:95  
Physics:90  
Chemistry:96  
English:93  
Computer:98
```

```
<?php  
// Defining a multidimensional array  
$favorites = array(  
    array(  
        "name" => "Ganesh",  
        "mob" => "5689741523",  
        "email" => "Ganesh@gmail.com",  
    ),  
    array(  
        "name" => "Saroj", "mob"  
        => "2584369721",  
        "email" => "montysmith@gmail.com",  
    ),  
    array(  
        "name" => "Arjun",  
        "mob" => "9875147536",  
        "email" => "Arjun.7.com.np@gmail.com",  
    )  
);  
// Accessing elements  
echo "Ganesh email-id is: " . $favorites[0]["email"], "\n"; echo  
"Arjun mobile number is: " . $favorites[2]["mob"];  
?>
```

## Multidimensional Arrays

Multi-dimensional arrays are such arrays which stores an another array at each index instead of single element. In other words, we can define multi-dimensional arrays as array of arrays. As the name suggests, every element in this array can be an array and they can also hold other sub-arrays within. Arrays or sub-arrays in multidimensional arrays can be accessed using multiple dimensions.

Example:

```
<?php  
// Defining a multidimensional array  
$favorites = array(  
    array(  
        "name" => "Ganesh",  
        "mob" => "5689741523",  
        "email" => "Ganesh@gmail.com",  
    ),  
    array(  
        "name" => "Saroj", "mob"  
        => "2584369721",  
        "email" => "Saroj@gmail.com",  
    ),  
    array(  
        "name" => "Arjun", "mob"  
        => "9875147536",  
        "email" => "Arjun.7.com.np@gmail.com",  
    )  
);  
// Accessing elements  
echo "Ganesh email-id is: " . $favorites[0]["email"], "\n";  
echo "Arjun mobile number is: " . $favorites[2]["mob"];  
?>
```

# **PROGRAMMING ON PHP**

# **GET AND POST**

We have two HTTP request methods in PHP for handling the forms, where submitted form-data from users can be collected using these methods. In order to send information to the web server from the browser client, we use GET and POST methods.

**GET Method:** Data is requested from a specific resource

**POST Method:** Data is submitted to be processed to a specific resource

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character.

**`http://www.test.com/index.htm?name1=value1&name2=value2`**

```
<?php
if( $_GET["name"] || $_GET["age"] )
{
echo "Welcome ". $_GET['name']. "<br />";
echo "You are ". $_GET['age']. " years old.";
exit();
}
?>
<html>
<body>
<form action = "<?php $_PHP_SELF ?>" method = "GET">
Name: <input type = "text" name = "name" />
Age: <input type = "text" name = "age" />
<input type = "submit" />
</form>
</body>
</html>
```

## The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY\_STRING.

```
<?php
if( $_POST["name"] || $_POST["age"] )
{
if (preg_match("/[^A-Za-z'-]/",$_POST['name'] ))
{
die ("invalid name and name should be alpha");
}
echo "Welcome ". $_POST['name']. "<br />";
echo "You are ". $_POST['age']. " years old.";
exit();
}
?>
<html>
<body>
<form action = "<?php $_PHP_SELF ?>" method = "POST">
Name: <input type = "text" name = "name" />
Age: <input type = "text" name = "age" />
<input type = "submit" />
</form>
</body>
</html>
```

## GET VS POST: Difference between GET Method and POST Method

GET	POST
GET Parameters are included in URL	POST parameters are included in the body
GET requests are often used for fetching documents and GET parameters are used to describe which document we are looking for (or) what page we are on (or) things of that nature.	POST parameters are often used for updating data for actually making changes to the server (or) to the data held on the server
Because they are in URL, have a maximum URL length because you can encode many parameters. For eg: Internet Explorer allows 2000 characters in the URL or something like that which can be quite limiting.	By default, they don't have any maximum length. Now the Server can be configured and most are to have a maximum length but it is usually substantially longer than 2000 characters.
When we make a GET request- a simple request for URL. There are a lot of machines between	Post parameters are almost never cached because you are probably updating data on the

you and server It saves a lot of effort if we know  
the document has not changed

server so the industry standard is: Don't cache  
POST request

They should not change the server. You should  
be able to make the same GET request over and  
the server should not change.

Post requests are okay to change the server.  
That is what they are generally used for  
requesting an update for the server and are not  
cached and there is no maximum length

## The `$_REQUEST` variable

The PHP `$_REQUEST` variable contains the contents of both `$_GET`, `$_POST`, and `$_COOKIE`.

The PHP `$_REQUEST` variable can be used to get the result from form data sent with both the GET and POST methods.

Try out following example by putting the source code in `test.php` script.

```
<?php
if( $_REQUEST["name"] || $_REQUEST["age"] )
{
    echo "Welcome ". $_REQUEST['name']. "<br />";
    echo "You are ". $_REQUEST['age']. " years old.";
    exit();
}
?>
<html>
<body>
<form action = "<?php $_PHP_SELF ?>" method = "POST"> Name:
<input type = "text" name = "name" />
Age: <input type = "text" name = "age" />
<input type = "submit" />
</form>
</body>
</html>
```

# **FILE INCLUSION AND FILES & I/O**

## **PHP - File Inclusion**

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

The `include()` Function

The `require()` Function

## The include() Function

The **include()** function takes all the text in a specified file and copies it into the file that uses the **include** function. If there is any problem in loading a file then the **include()** function generates a warning but the script will continue execution.

Assume you want to create a common menu for your website.  
Then create a file menu.php with the following content.

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a>
- <a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> - <a
href="http://www.tutorialspoint.com/perl">PERL</a> <br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

```
<html>
<body>
<?php
include("menu.php");
?>
<p>This is an example to show how to include PHP file!</p>
</body>
</html>
```

It will produce the following result –

[Home](#) -  
[ebXML](#) -  
[AJAX](#) -  
[PERL](#)

This is an example to show how to include PHP file!

## The require() Function

The `require()` function takes all the text in a specified file and copies it into the file that uses the `include` function. If there is any problem in loading a file then the `require()` function generates a fatal error and halt the execution of the script.

You can try using above example with require() function and it will generate same result. But if you will try following two examples where file does not exist then you will get different results.

```
<html>
<body>
<?php
include("xxmenu.php");
?>
<p>This is an example to show how to include wrong PHP
file!</p>
</body>
</html>
```

Now lets try same example with require() function.

```
<html>
```

```
<body>
```

```
<?php
```

```
require("xxmenu.php");
```

```
?>
```

**<p>This is an example to show how to include wrong PHP**

**file!</p>**

```
</body>
```

```
</html>
```

# **PHP - FILES & I/O**

following functions related to files –

Opening a file

Reading a file

Writing a file

Closing a file

## Opening and Closing Files

The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Sr.No	Mode & Purpose	
1	r Opens the file for reading only. Places the file pointer at the beginning of the file.	4 <b>w+</b> Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
2	r+ Opens the file for reading and writing. Places the file pointer at the beginning of the file.	5 <b>a</b> Opens the file for writing only. Places the file pointer at the end of the file.
3	w Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.	6 <b>a+</b> Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

## Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

So here are the steps required to read a file with PHP.

Open a file using **fopen()** function.

Get the file's length using **filesize()** function.

Read the file's content using **fread()** function.

Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

```
html>
<head>
<title>Reading a file using PHP</title> </head>
<body>
<?php
$filename = "tmp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
echo ( "Error in opening file" );
exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );
fclose( $file );
echo ( "File size : $filesize bytes" );
echo ( "<pre>$filetext</pre>" );
?>
</body>
</html>
```

It will produce the following result -

File size : 278 bytes

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based software applications. This tutorial helps you to build your base with PHP.

## Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using **file\_exist()** function which takes file name as an argument

```
<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
echo ( "Error in opening new file" );
exit();
}
fwrite( $file, "This is a simple test\n" );
fclose( $file );
?>
<html>
<head>
<title>Writing a file using PHP</title> </head>
<body>
<?php
$filename = "newfile.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
echo ( "Error in opening file" );
exit();
}
```

```
$filesize = filesize( $filename );  
$filetext = fread( $file, $filesize );  
fclose( $file );  
echo ( "File size : $filesize bytes" );  
echo ( "$filetext" );  
echo("file name: $filename");  
?  
</body>  
</html>
```

It will produce the following result -

```
File size : 23 bytes  
This is a simple test  
file name: newfile.txt
```

# FUNCTIONS

A function is a piece of code which takes one or more input in the form of parameter and does some processing and returns a value.

PHP provides us with two major types of functions:

**Built-in functions** : PHP provides us with huge collection of built-in library functions. These functions are already coded and stored in form of functions. To use those we just need to call them as per our requirement like, var\_dump, fopen(), print\_r(), gettype() and so on.

**User Defined Functions** : Apart from the built-in functions, PHP allows us to create our own customised functions called the user-defined functions.

Using this we can create our own packages of code and use it wherever necessary by simply calling it.

## Creating a Function

While creating a user defined function we need to keep few things in mind:

Any name ending with an open and closed parenthesis is a function.

A function name always begins with the keyword *function*.

To call a function we just need to write its name followed by the parenthesis

A function name cannot start with a number. It can start with an alphabet or underscore.

A function name is not case-sensitive.

**Syntax:**

```
function function_name()
```

```
{
```

```
executable code;
```

```
}
```

```
<?php  
  
function funcGeek()  
{  
    echo "This is Geeks for Geeks";  
}  
  
// Calling the function  
funcGeek();  
  
?>
```

## PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (\$fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}
familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

# USER DEFINED FUNCTIONS

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads. A function will be executed by a call to the function.

Function names are NOT case-sensitive.

## Syntax

```
function functionName() {  
    code to be executed;  
}
```

# EXAMPLE OF USER DEFINED FUNCTIONS

```
<html>
<body>
<?php
function writeMsg() {
    echo "Hello Nabraj!";
}
writeMsg();
?>
</body>
</html>
Output:
Hello Nabraj!
```

# **FUNCTION ARGUMENTS**

**Information can be passed to functions through arguments. An argument is just like a variable.**

**Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.**

# FUNCTION ARGUMENTS

```
<html>
<body>
<?php
function familyName($fname) {
    echo "$fname<br>";
}
familyName("Nabraj");
familyName("Ramesh");
familyName("Pawan");
familyName("Nabin");
familyName("Khim");
?>
</body>
</html>
```

Output:  
Nabraj  
Ramesh  
Pawan  
Nabin  
Khim

# FUNCTIONS - RETURNING VALUES

To let a function return a value, use the return statement:

```
<html>
<body>
<?php
function sum(int $x, int $y) {
    $z = $x + $y;
    return $z;
}
echo sum(5,10);
echo "<br>";
echo sum(7,13);
?>
</body>
</html>
Output:
15
20
```

# **FUNCTION TO FIND AREA OF RECTANGLE AND RETURN THE VALUE**

```
<html>
<body>
<?php
function sum( int $l, int $b) {
    $a = $l * $b;
    return $a;
}
$b= sum(15,200);
echo $b;
?>
</body>
</html>
```

# FUNCTION TO ADD TWO NUMBER

```
<html>
<body>
<?php
function sum(int $x, int $y) {
    $z = $x + $y;
    echo $z;
}
echo sum(5,10);
?>
</body>
</html>
```

# **COOKIE**

## **What is a cookie**

**Cookies are used to store the information of a web page in a remote browser, so that when the same user comes back to that page, that information can be retrieved from the browser itself.**

# **USES OF COOKIE**

**Cookies are often used to perform following tasks:**

**Session management:** Cookies are widely used to manage user sessions. For example, when you use an online shopping cart, you keep adding items in the cart and finally when you checkout, all of those items are added to the list of items you have purchased. This can be achieved using cookies.

**User identification:** Once a user visits a webpage, using cookies, that user can be remembered. And later on, depending upon the search/visit pattern of the user, content which the user likely to be visited are served. A good example of this is 'Retargetting'. A concept used in online marketing, where depending upon the user's choice of content, advertisements of the relevant product, which the user may buy, are served.

**Tracking / Analytics:** Cookies are used to track the user. Which, in turn, is used to analyze and serve various kind of data of great value, like location, technologies (e.g. browser, OS) form where the user visited, how long (s)he stayed on various pages etc.

# **HOW TO CREATE A COOKIE IN PHP**

**PHP has a setcookie() function to send a cookie. We will discuss this function in detail now.**

**Syntax:**

**setcookie(name, value, expire)**

# Example to Create/Retrieve a Cookie:

```
<html>
<?php
$cookie_name = "user";
$cookie_value = "Nabraj Koirala";
setcookie($cookie_name, $cookie_value, time() + 10);
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named " . $cookie_name . " is not set!";
} else {
    echo "Cookie " . $cookie_name . " is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

# PHP SESSIONS

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

# Start a PHP Session:

A session is started with the `session_start()` function.

Session variables are set with the PHP global variable: `$_SESSION`.

```
<?php
// Start the session
session_start();
?>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

# Get PHP Session Variable Values:

```
<?php
session_start();
?>
<html>
<body>
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
</body>
</html>
```

# DESTROY A PHP SESSION

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

## Example

```
<?php
session_start();
?>
<html>
<body>
<?php
// remove all session variables
session_unset();
// destroy the session
session_destroy();
echo "All session variables are now removed, and the session is destroyed."
?>
</body>
</html>
```

# **SENDING EMAILS**

## **What is PHP mail?**

PHP mail is the built in PHP function that is used to send emails from PHP scripts.

The mail function accepts the following parameters;

Email address

Subject

Message

CC or BC email addresses

## Sending mail using PHP

The PHP mail function has the following basic syntax

```
<?php  
mail($to_email_address,$subject,$message,[$headers],[$parameters]);  
?>
```

HERE,

“\$to\_email\_address” is the email address of the mail recipient

“\$subject” is the email subject

“\$message” is the message to be sent.

“[\$headers]” is optional, it can be used to include information such as CC, BCC

- CC is the acronym for carbon copy. It's used when you want to send a copy to an interested person i.e. a complaint email sent to a company can also be sent as CC to the complaints board.
- BCC is the acronym for blind carbon copy. It is similar to CC. The email addresses included in the BCC section will not be shown to the other recipients.

## **Simple Mail Transmission Protocol (SMTP)**

PHP mailer uses Simple Mail Transmission Protocol (SMTP) to send mail.

On a hosted server, the SMTP settings would have already been set.

The SMTP mail settings can be configured from “php.ini” file in the PHP installation folder.

Configuring SMTP settings on your localhost Assuming you are using xampp on windows, locate the “php.ini” in the directory “C:\xampp\php”.

Open it using notepad or any text editor. We will use notepad in this example. Click on the edit menu

php.ini - Notepad

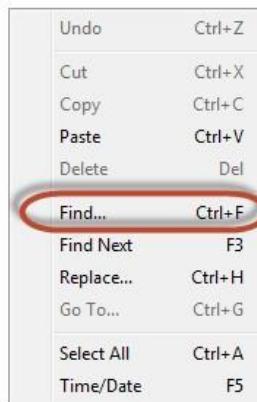
File Edit Format View Help

[PHP]

```
; About php.ini
; PHP's initialization file, generally called php.ini, is
; responsible for
; configuring many of the aspects of PHP's behavior.

; PHP attempts to find and load this configuration from a
; number of locations.
; The following is a summary of its search order:
; 1. SAPI module specific location.
; 2. The PHPRC environment variable. (As of PHP 5.2.0)
; 3. A number of predefined registry keys on windows (As of
```

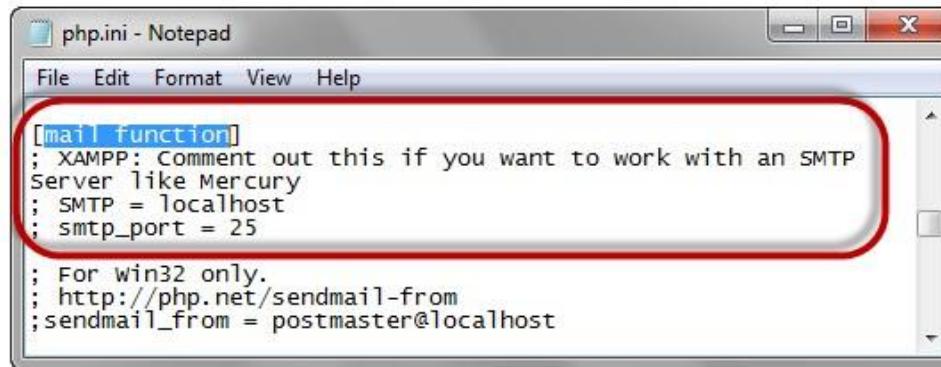
- Click on Find... menu



- The find dialog menu will appear



- Click on Find Next button



locate the entries [*mail function*]

; XAMPP: Don't remove the semi column if you want to work with an SMTP Server like Mercury

; SMTP = localhost

; smtp\_port = 25

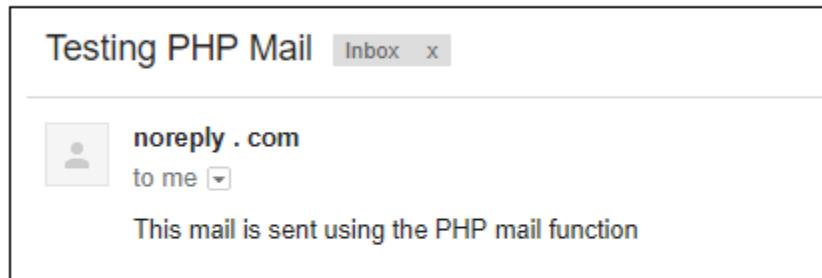
Remove the semi colons before SMTP and smtp\_port and set the SMTP to your smtp server and the port to your smtp port. Your settings should look as follows

- SMTP = smtp.example.com
- smtp\_port = 25
- *Note the SMTP settings can be gotten from your web hosting providers.*
- If the server requires authentication, then add the following lines.
  - auth\_username = example\_username@example.com
  - auth\_password = example\_password
  - Save the new changes.
  - Restart Apache server.

Let's now look at an example that sends a simple mail.

```
<?php  
$to_email = 'name @ company . com';  
$subject = 'Testing PHP Mail';  
$message = 'This mail is sent using the PHP mail function';  
$headers = 'From: noreply @ company . com';  
mail($to_email,$subject,$message,$headers);  
?>
```

Output:



*Note: the above example only takes the 4 mandatory parameters.*

*You should replace the above fictitious email address with a real email address.*

# **FILE UPLOADING**

# ERROR HANDLING

# PHP and Database Connection

# Basic database concepts

# How to Create a MySQL Database with phpMyAdmin

- **Make sure you have phpMyAdmin Installed.** This guide assumes you have phpMyAdmin installed and ready to use.
- **Go to your phpMyAdmin homepage.** For this guide you need to navigate to the phpMyAdmin homepage. It can usually be found under these directories, depending on your server setup.  
`http://localhost:8888/phpMyAdmin/`  
`http://localhost/phpmyadmin`  
`http://{your-ip-address}/phpmyadmin/`

- **Login to your phpMyAdmin Page.** Make sure you are logged in to phpMyAdmin.
- **Enter your database name.** Enter the name you want to call your new database, here we have named it "my\_new\_database".
- **Click "Create" and then you're done!**

# MySQL Database

- With PHP, you can connect to and manipulate databases.
- MySQL is the most popular database system used with PHP.
- The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.
- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation

# Connecting to MySQL Server

- Before we can access data in the MySQL database, we need to be able to connect to the server:
- The process to connect MySQL server is given below:

```
<?php  
$user = 'root';  
$pass = ";  
// Create connection  
$conn = mysqli_connect('localhost', $user, $pass);  
// Check connection  
if (!$conn) {  
    die("Connection failed: " . mysqli_connect_error());  
}  
else  
{  
echo "Connected successfully";  
}  
?>
```

# Create a MySQL Database

- The CREATE DATABASE statement is used to create a database in MySQL.
- The following examples create a database named “nawaraj”:

```
<?php  
$user = 'root';  
$pass = ";  
$conn = mysqli_connect('localhost', $user, $pass);  
// Check connection  
//Create database  
$sql = "CREATE DATABASE cct1";  
if (mysqli_query($conn, $sql)) {  
    echo "Database created successfullyyy";  
} else {  
    echo "Error creating database: " . mysqli_error($conn);  
}  
?>
```

# Create MySQL Tables

- A database table has its own unique name and consists of columns and rows.
- The CREATE TABLE statement is used to create a table in MySQL.
- We will create a table named "contacts", with five columns: "id", "firstName", "lastName", "email" :
- **CREATE TABLE contacts (**  
**id INT(6) UNSIGNED AUTO\_INCREMENT PRIMARY KEY,**  
**firstname VARCHAR(30) NOT NULL,**  
**lastname VARCHAR(30) NOT NULL,**  
**email VARCHAR(50),**  
**)**

## Create table

```
<?php  
$server = "localhost";  
$dbuser = "root";  
$dbpass = "";  
$dbname = "n1";  
$conn = mysqli_connect($server, $dbuser, $dbpass,$dbname);  
$sqlQuery = "CREATE TABLE contacts (  
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
firstName VARCHAR(35) NOT NULL,  
lastName VARCHAR(35) NOT NULL,  
email VARCHAR(55)  
);  
?>
```

# Insert Data Into MySQL

- After a database and a table have been created, we can start adding data in them.
- The INSERT INTO statement is used to add new records to a MySQL table:
- `INSERT INTO table_name (column1, column2, column3,...)  
VALUES (value1, value2, value3,...)`

**Here are some syntax rules to follow:**

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

## ● Example to insert single record into a table

```
<?php  
$user = 'root';  
$pass = ":";  
$dbname = "n1";  
// Create connection  
$conn = mysqli_connect('localhost', $user, $pass, $dbname);  
$sql = "INSERT INTO contacts (id,firstName, lastName, email)  
VALUES (' ', 'John', 'Doe', 'john@example.com')";  
?>
```

- Example to insert multiple record into a table

```
<?php  
$user = 'root';  
$pass = " ";  
$dbname = "nabraj";  
// Create connection  
$conn = mysqli_connect('localhost', $user, $pass, $dbname);  
$sql = "INSERT INTO MyGuests (firstname, lastname, email)  
VALUES ('Nabraj', 'Koirala', 'koiralanabraj@gmail.com'),  
('Apil', 'Koirala', 'apil@gmail.com'),  
('Ram', 'Pandey', 'rampandey@gmail.com')";  
?>
```

# Select Data From a MySQL Database

- The SELECT statement is used to select data from one or more tables:

**SELECT column\_name(s) FROM table\_name**

- or we can use the \* character to select ALL columns from a table:

**SELECT \* FROM table\_name**

## ● Example of Select Data From a MySQL Database

```
<?php
$user = 'root';
$pass = "";
$dbname = "n1";
// Create connection
$conn = mysqli_connect('localhost', $user, $pass, $dbname);
$sql = "SELECT * FROM contacts";
$result = mysqli_query($conn, $sql);
// output data of each row
while($row = $result->fetch_assoc()) {
echo "<br> id: ". $row["id"]. " - Name: ". $row["firstName"]. " " . $row["lastName"]
." Email:". $row["email"]. "<br>";
}
?>
</body>
</html>
```

- Select particular data from table

```
<?php
$user = 'root';
$pass = "";
$dbname = "n1";
// Create connection
$conn = mysqli_connect('localhost', $user, $pass, $dbname);
$sql = "SELECT id, firstName, lastName FROM contacts";
$result = $conn->query($sql);
// output data of each row
while($row = $result->fetch_assoc()) {
    echo "<br> id: ". $row["id"]. " - Name: ". $row["firstName"]. " " .
    $row["lastName"] . "<br>";
}
?>
</body>
</html>
```

# Delete Data From MySQL

- The DELETE statement is used to delete records from a table:
- `DELETE FROM table_name  
WHERE some_column = some_value`

## Example of Delete Data From MySQL

```
<?php  
$user = 'root';  
$pass = " ";  
$dbname = "n1";  
// Create connection  
$conn = mysqli_connect('localhost', $user, $pass, $dbname);  
// sql to delete a record  
$sql = "DELETE FROM contacts WHERE id=5";  
?>
```

# Update Data in MySQL

- The UPDATE statement is used to update existing records in a table:
- `UPDATE table_name  
SET column1=value, column2=value2,...  
WHERE some_column=some_value`

- **Example of Update Data in MySQL**

```
<?php  
$user = 'root';  
$pass = " ";  
$dbname = "n1";  
// Create connection  
$conn = mysqli_connect('localhost', $user, $pass, $dbname);  
// sql to update a record  
$sql = "UPDATE contacts SET lastname='Koirala' WHERE id=4";  
?>
```

- Close the Connection
- The connection will be closed automatically when the script ends.  
To close the connection before, use the following:
  - MySQLi Object-Oriented:  
`$conn->close();`
  - MySQLi Procedural:  
`mysqli_close($conn);`
- PDO:  
`$conn = null;`

- Opening Database Connection
- PHP provides **mysql\_connect** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.
- Syntax
- connection
  - mysql\_connect(server,user,passwd,new\_link,client\_flag);

- Closing Database Connection
- Its simplest function **mysql\_close** PHP provides to close a database connection. This function takes connection resource returned by mysql\_connect function. It returns TRUE on success or FALSE on failure.
- Syntax
- `bool mysql_close ( resource $link_identifier );`

# Example

- <?php

```
$dbhost = 'localhost:3036';
$dbuser = 'guest';
$dbpass = 'guest123';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn ) {
die('Could not connect: ' .mysql_error()); }
echo 'Connected successfully';
mysql_close($conn);
?>
```

# Create MySQL Database Using PHP

- Creating a Database
- To create and delete a database you should have admin privilege. Its very easy to create a new MySQL database. PHP uses **mysql\_query** function to create a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.
- Syntax
- `bool mysql_query( sql, connection );`

## Example

Try out following example to create a database –

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }

    echo 'Connected successfully';

    $sql = 'CREATE Database test_db';
    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {
        die('Could not create database: ' . mysql_error());
    }

    echo "Database test_db created successfully\n";
    mysql_close($conn);
?>
```

- Selecting a Database
- Once you establish a connection with a database server then it is required to select a particular database where your all the tables are associated.
- PHP provides function **mysql\_select\_db** to select a database. It returns TRUE on success or FALSE on failure.
- Syntax
- `bool mysql_select_db( db_name, connection );`

## Example

Here is the example showing you how to select a database.

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'guest';
    $dbpass = 'guest123';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }

    echo 'Connected successfully';

    mysql_select_db( 'test_db' );
    mysql_close($conn);

?>
```

- Creating Database Tables
- To create tables in the new database you need to do the same thing as creating the database. First create the SQL query to create the tables then execute the query using mysql\_query() function.

```
<?php

$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}

echo 'Connected successfully';

$sql = 'CREATE TABLE employee( '.
    'emp_id INT NOT NULL AUTO_INCREMENT, '.
    'emp_name VARCHAR(20) NOT NULL, '.
    'emp_address  VARCHAR(20) NOT NULL, '.
    'emp_salary   INT NOT NULL, '.
    'join_date    timestamp(14) NOT NULL, '.
    'primary key ( emp_id ))';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if(! $retval ) {
    die('Could not create table: ' . mysql_error());
}

echo "Table employee created successfully\n";

mysql_close($conn);
?>
```

- In case you need to create many tables then its better to create a text file first and put all the SQL commands in that text file and then load that file into \$sql variable and execute those commands.
- Consider the following content in sql\_query.txt file
- ```
CREATE TABLE employee( emp_id INT NOT NULL
AUTO_INCREMENT, emp_name VARCHAR(20) NOT
NULL, emp_address VARCHAR(20) NOT NULL,
emp_salary INT NOT NULL, join_date timestamp(14) NOT
NULL, primary key ( emp_id ));
```

```
<?php

$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}

$query_file = 'sql_query.txt';

$fp = fopen($query_file, 'r');
$sql = fread($fp, filesize($query_file));
fclose($fp);

mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if(! $retval ) {
    die('Could not create table: ' . mysql_error());
}

echo "Table employee created successfully\n";
mysql_close($conn);
?>
```

# Deleting MySQL Database Using PHP

- Deleting a Database
- If a database is no longer required then it can be deleted forever. You can use pass an SQL command to **mysql\_query** to delete a database.
- Example
- Try out following example to drop a database.

```
<?php  
    $dbhost = 'localhost:3036';  
    $dbuser = 'root';  
    $dbpass = 'rootpassword';  
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);  
  
    if(! $conn ) {  
        die('Could not connect: ' . mysql_error());  
    }  
  
    $sql = 'DROP DATABASE test_db';  
    $retval = mysql_query( $sql, $conn );  
  
    if(! $retval ) {  
        die('Could not delete database db_test: ' . mysql_error());  
    }  
  
    echo "Database deleted successfully\n";  
  
    mysql_close($conn);  
?>
```

- Deleting a Table
- Its again a matter of issuing one SQL command through **mysql\_query** function to delete any database table. But be very careful while using this command because by doing so you can delete some important information you have in your table.
- Example
- Try out following example to drop a table –

```
<?php  
    $dbhost = 'localhost:3036';  
    $dbuser = 'root';  
    $dbpass = 'rootpassword';  
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);  
  
    if(! $conn ) {  
        die('Could not connect: ' . mysql_error());  
    }  
  
    $sql = 'DROP TABLE employee';  
    $retval = mysql_query( $sql, $conn );  
  
    if(! $retval ) {  
        die('Could not delete table employee: ' . mysql_error());  
    }  
  
    echo "Table deleted successfully\n";  
  
    mysql_close($conn);  
?>
```

# Insert Data into MySQL Database

- Data can be entered into MySQL tables by executing SQL INSERT statement through PHP function `mysql_query`. Below a simple example to insert a record into **employee** table.
- Example
- Try out following example to insert record into employee table.

```
<?php  
    $dbhost = 'localhost:3036';  
    $dbuser = 'root';  
    $dbpass = 'rootpassword';  
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);  
  
    if(! $conn ) {  
        die('Could not connect: ' . mysql_error());  
    }  
  
    $sql = 'INSERT INTO employee '.  
        '(emp_name,emp_address, emp_salary, join_date) '.  
        'VALUES ( "guest", "XYZ", 2000, NOW() )';  
  
    mysql_select_db('test_db');  
    $retval = mysql_query( $sql, $conn );  
  
    if(! $retval ) {  
        die('Could not enter data: ' . mysql_error());  
    }  
  
    echo "Entered data successfully\n";  
  
    mysql_close($conn);  
?>
```

# Getting Data From MySQL Database

- Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function `mysql_query`. You have several options to fetch data from MySQL.
- The most frequently used option is to use function `mysql_fetch_array()`. This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.
- Below is a simple example to fetch records from **employee** table.

```
<?php

$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}

$sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if(! $retval ) {
    die('Could not get data: ' . mysql_error());
}

while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) {
    echo "EMP ID :{$row['emp_id']} <br> ".
        "EMP NAME : {$row['emp_name']} <br> ".
        "EMP SALARY : {$row['emp_salary']} <br> ".
        "-----<br>";
}

echo "Fetched data successfully\n";

mysql_close($conn);
?>
```

- Example
- Try out following example to display all the records from employee table using mysql\_fetch\_assoc() function.

```
<?php

$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}

$sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if(! $retval ) {
    die('Could not get data: ' . mysql_error());
}

while($row = mysql_fetch_assoc($retval)) {
    echo "EMP ID :{$row['emp_id']} <br> ".
        "EMP NAME : {$row['emp_name']} <br> ".
        "EMP SALARY : {$row['emp_salary']} <br> ".
        "-----<br>";
}

echo "Fetched data successfully\n";

mysql_close($conn);

?>
```

- You can also use the constant **MYSQL\_NUM**, as the second argument to `mysql_fetch_array()`. This will cause the function to return an array with numeric index.
- Example
- Try out following example to display all the records from employee table using **MYSQL\_NUM** argument.

```
<?php

    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';

    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }

    $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
    mysql_select_db('test_db');
    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {
        die('Could not get data: ' . mysql_error());
    }

    while($row = mysql_fetch_array($retval, MYSQL_NUM)) {
        echo "EMP ID :{$row[0]} <br> ".
            "EMP NAME : {$row[1]} <br> ".
            "EMP SALARY : {$row[2]} <br> ".
            "-----<br>";
    }

    echo "Fetched data successfully\n";

    mysql_close($conn);
?>
```

# Releasing Memory

- It's a good practice to release cursor memory at the end of each SELECT statement. This can be done by using PHP function **mysql\_free\_result()**. Below is the example to show how it has to be used.
- Example
- Try out following example

```
<?php

$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if( ! $conn ) {
    die('Could not connect: ' . mysql_error());
}

$sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if( ! $retval ) {
    die('Could not get data: ' . mysql_error());
}

while($row = mysql_fetch_array($retval, MYSQL_NUM)) {
    echo "EMP ID :{$row[0]} <br> ".
        "EMP NAME : {$row[1]} <br> ".
        "EMP SALARY : {$row[2]} <br> ".
        "-----<br>";
}

mysql_free_result($retval);
echo "Fetched data successfully\n";

mysql_close($conn);

?>
```

# Using Paging through PHP

- Paging means showing your query result in multiple pages instead of just putting them all in one long page.
- MySQL helps to generate paging by using **LIMIT** clause which will take two arguments. First argument as **OFFSET** and second argument how many records should be returned from the database.
- Below is a simple example to fetch records using **LIMIT** clause to generate paging.
- Example
- Try out following example to display 10 records per page.

```
<html>

    <head>
        <title>Paging Using PHP</title>
    </head>

    <body>
        <?php
            $dbhost = 'localhost:3036';
            $dbuser = 'root';
            $dbpass = 'rootpassword';

            $rec_limit = 10;
            $conn = mysql_connect($dbhost, $dbuser, $dbpass);

            if(! $conn ) {
                die('Could not connect: ' . mysql_error());
            }
            mysql_select_db('test_db');

            /* Get total number of records */
            $sql = "SELECT count(emp_id) FROM employee ";
            $retval = mysql_query( $sql, $conn );

            if(! $retval ) {
                die('Could not get data: ' . mysql_error());
            }
            $row = mysql_fetch_array($retval, MYSQL_NUM );
            $rec_count = $row[0];
```

```
if( isset($_GET{'page'}) ) {
    $page = $_GET{'page'} + 1;
    $offset = $rec_limit * $page ;
}else {
    $page = 0;
    $offset = 0;
}

$left_rec = $rec_count - ($page * $rec_limit);
$sql = "SELECT emp_id, emp_name, emp_salary ".
    "FROM employee ".
    "LIMIT $offset, $rec_limit";

$retval = mysql_query( $sql, $conn );

if(! $retval ) {
    die('Could not get data: ' . mysql_error());
}

while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) {
    echo "EMP ID :{$row['emp_id']} <br> ".
        "EMP NAME : {$row['emp_name']} <br> ".
        "EMP SALARY : {$row['emp_salary']} <br> ".
        "-----<br>";
}
```

```
}

if( $page > 0 ) {
    $last = $page - 2;
    echo "<a href = \"$_PHP_SELF?page = $last\">Last 10 Records</a> | ";
    echo "<a href = \"$_PHP_SELF?page = $page\">Next 10 Records</a>";
} else if( $page == 0 ) {
    echo "<a href = \"$_PHP_SELF?page = $page\">Next 10 Records</a>";
} else if( $left_rec < $rec_limit ) {
    $last = $page - 2;
    echo "<a href = \"$_PHP_SELF?page = $last\">Last 10 Records</a>";
}

mysql_close($conn);
?>

</body>
</html>
```

# Updating Data into MySQL Database

- Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function **mysql\_query**.
- Example
- Try out following example to understand update operation. You need to provide an employee ID to update an employee salary.

```
<html>

    <head>
        <title>Update a Record in MySQL Database</title>
    </head>

    <body>
        <?php
            if(isset($_POST['update'])) {
                $dbhost = 'localhost:3036';
                $dbuser = 'root';
                $dbpass = 'rootpassword';

                $conn = mysql_connect($dbhost, $dbuser, $dbpass);

                if(! $conn ) {
                    die('Could not connect: ' . mysql_error());
                }

                $emp_id = $_POST['emp_id'];
                $emp_salary = $_POST['emp_salary'];

                $sql = "UPDATE employee ". "SET emp_salary = $emp_salary ".
                    "WHERE emp_id = $emp_id" ;
                mysql_select_db('test_db');
                $retval = mysql_query( $sql, $conn );

                if(! $retval ) {
                    die('Could not update data: ' . mysql_error());
                }
                echo "Updated data successfully\n";
            }
        </?php>
    </body>
</html>
```

```
    mysql_close($conn);
}else {
?>
<form method = "post" action = "<?php $_PHP_SELF ?>">
    <table width = "400" border = " 0" cellspacing = "1"
        cellpadding = "2">

        <tr>
            <td width = "100">Employee ID</td>
            <td><input name = "emp_id" type = "text"
                id = "emp_id"></td>
        </tr>

        <tr>
            <td width = "100">Employee Salary</td>
            <td><input name = "emp_salary" type = "text"
                id = "emp_salary"></td>
        </tr>

        <tr>
            <td width = "100"> </td>
            <td> </td>
        </tr>

        <tr>
            <td width = "100"> </td>
            <td>
                <input name = "update" type = "submit"
                    id = "update" value = "Update">
            </td>
        </tr>
```

```
        </table>
    </form>
<?php
}
?>

</body>
</html>
```

# Deleting Data from MySQL Database

- Data can be deleted from MySQL tables by executing SQL DELETE statement through PHP function **mysql\_query**.
- Example
- Try out following example to understand delete operation. You need to provide an employee ID to delete an employee record from employee table.

```
<html>

    <head>
        <title>Delete a Record from MySQL Database</title>
    </head>

    <body>
        <?php
            if(isset($_POST['delete'])) {
                $dbhost = 'localhost:3036';
                $dbuser = 'root';
                $dbpass = 'rootpassword';
                $conn = mysql_connect($dbhost, $dbuser, $dbpass);

                if(! $conn ) {
                    die('Could not connect: ' . mysql_error());
                }

                $emp_id = $_POST['emp_id'];

                $sql = "DELETE FROM employee WHERE emp_id = $emp_id" ;
                mysql_select_db('test_db');
                $retval = mysql_query( $sql, $conn );

                if(! $retval ) {
                    die('Could not delete data: ' . mysql_error());
                }

                echo "Deleted data successfully\n";

                mysql_close($conn);
            } else {
        
```

```
?>
<form method = "post" action = "<?php $_PHP_SELF ?>">
    <table width = "400" border = "0" cellspacing = "1"
           cellpadding = "2">

        <tr>
            <td width = "100">Employee ID</td>
            <td><input name = "emp_id" type = "text"
                   id = "emp_id"></td>
        </tr>

        <tr>
            <td width = "100"> </td>
            <td> </td>
        </tr>

        <tr>
            <td width = "100"> </td>
            <td>
                <input name = "delete" type = "submit"
                       id = "delete" value = "Delete">
            </td>
        </tr>

    </table>
</form>
<?php
}
?>

</body>
```

# Perform MySQL backup using PHP

- It is always good practice to take a regular backup of your database. There are three ways you can use to take backup of your MySQL database.
- Using SQL Command through PHP.
- Using MySQL binary mysqldump through PHP.
- Using phpMyAdmin user interface.
- Using SQL Command through PHP
- You can execute SQL SELECT command to take a backup of any table. To take a complete database dump you will need to write separate query for separate table. Each table will be stored into separate text file.

- Example
- Try out following example of using SELECT INTO OUTFILE query for creating table backup –

```
<?php

$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}

$table_name = "employee";
$backup_file = "/tmp/employee.sql";
$sql = "SELECT * INTO OUTFILE '$backup_file' FROM $table_name";

mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if(! $retval ) {
    die('Could not take data backup: ' . mysql_error());
}

echo "Backuped data successfully\n";

mysql_close($conn);
?>
```

- There may be instances when you would need to restore data which you have backed up some time ago. To restore the backup you just need to run LOAD DATA INFILE query like this –

```
<?php

$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}

$table_name = "employee";
$backup_file = "/tmp/employee.sql";
$sql = "LOAD DATA INFILE '$backup_file' INTO TABLE $table_name";

mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if(! $retval ) {
    die('Could not load data : ' . mysql_error());
}
echo "Loaded data successfully\n";

mysql_close($conn);
?>
```

- Using MySQL binary mysqldump through PHP
- MySQL provides one utility **mysqldump** to perform database backup. Using this binary you can take complete database dump in a single command.
- Example
- Try out following example to take your complete database dump –

- <?php \$dbhost = 'localhost:3036';  
    \$dbuser = 'root';  
    \$dbpass = 'rootpassword';  
    \$backup\_file = \$dbname . date("Y-m-d-H-i-s") . '.gz';  
    \$command = "mysqldump --opt -h \$dbhost -u \$dbuser  
-p \$dbpass ". "test\_db | gzip > \$backup\_file";  
    system(\$command);  
?>

- Using phpMyAdmin user interface
- If you have **phpMyAdmin** user interface available then its very easy for your to take backup of your database.
- To backup your MySQL database using phpMyAdmin click on the "export" link on phpMyAdmin main page. Choose the database you wish to backup, check the appropriate SQL options and enter the name for the backup file.

- [https://www.tutorialspoint.com/php/connect\\_to\\_mysql\\_using\\_php.htm](https://www.tutorialspoint.com/php/connect_to_mysql_using_php.htm)

PHP

# FRAMEWORK

---

# LARAVEL

# PHP Frameworks

---

- A PHP Framework is a basic platform that allows us to develop web applications. PHP Framework, helps for stopping the need to produce repetitive code, and you'll be able to build applications rapidly (RAD). Without a PHP Framework in place, it gets much more difficult to produce applications since you'll have to repeatedly code a lot of PHP.



- PHP operates on the Model View Controller (MVC) fundamentals.
- Model View Controller or **MVC** as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts –
  - **Model** – The lowest level of the pattern which is responsible for maintaining data.
  - **View** – This is responsible for displaying all or a portion of the data to the user.
  - **Controller** – Software Code that controls the interactions between the Model and View.



## □ **What to Look for When Choosing Your Frameworks?**

### a) **Database Support**

- Database support is very important. For example, CodeIgniter framework supports MySQL, Oracle, and SQLite, while the Kohana framework doesn't support Oracle or SQLite. Depending on which database you prefer to use or choose for your project at hand.

### b) **Community Support**

- Your framework should have a strong community, not just in terms of size but also in activity and helpfulness. Even if it's a small community, as long as you're able to find support, then that's a plus point.

### c) **Documentation Support**

- You should also be tired of frameworks that don't have any documentation and absolutely no user guide. Make sure that your PHP Framework has good documentation that's kept up-to-date, and that the user guide is relatively easy to follow.



# PHP framework:

---

- 1) Laravel
- 2) Symfony



# Laravel

---

- Laravel is an open-source PHP framework, which is robust and easy to understand. It follows a model-view-controller design pattern. Laravel reuses the existing components of different frameworks which helps in creating a web application. The web application thus designed is more structured and pragmatic.
- Laravel offers a rich set of functionalities which incorporates the basic features of PHP frameworks. Laravel has a very rich set of features which will boost the speed of web development.
- If you are familiar with Core PHP and Advanced PHP, Laravel will make your task easier. It saves a lot time if you are planning to develop a website from scratch. Moreover, a website built in Laravel is secure and prevents several web attacks.



# Advantages of Laravel

---

Laravel offers you the following advantages, when you are designing a web application based on it –

- The web application becomes more scalable, owing to the Laravel framework.
  - Considerable time is saved in designing the web application, since Laravel reuses the components from other framework in developing web application.
  - It includes namespaces and interfaces, thus helps to organize and manage resources.
- 



---

## □ Composer

- Composer is a tool which includes all the dependencies and libraries. It allows a user to create a project with respect to the mentioned framework (for example, those used in Laravel installation). Third party libraries can be installed easily with help of composer.
  - All the dependencies are noted in **composer.json** file which is placed in the source folder.
-

- 
- Artisan
  - Command line interface used in Laravel is called **Artisan**. It includes a set of commands which assists in building a web application. These commands are incorporated from Symphony framework, resulting in add-on features in Laravel 5.1 (latest version of Laravel).
-

# Features of Laravel

---

Laravel offers the following key features which makes it an ideal choice for designing web applications –

- **Modularity:**

Laravel provides 20 built in libraries and modules which helps in enhancement of the application. Every module is integrated with Composer dependency manager which eases updates.

- **Testability:**

Laravel helps in testing through various test cases. This feature helps in maintaining the code as per the requirements.



## □ Routing

---

Laravel provides a flexible approach to the user to define routes in the web application. Routing helps to scale the application in a better way and increases its performance.

## □ Configuration Management

A web application designed in Laravel will be running on different environments. Laravel provides a consistent approach to handle the configuration in an efficient way.

## □ Query Builder and ORM

Laravel incorporates a query builder which helps in querying databases using various simple methods. It provides **ORM** (Object Relational Mapper) and **ActiveRecord** implementation called Eloquent.



## □ Schema Builder

Schema Builder maintains the database definitions and schema in PHP code.

- ### Template Engine

Laravel uses the **Blade Template** engine, a lightweight template language used to design hierarchical blocks and layouts that include dynamic content.

## □ E-mail

Laravel includes a **mail** class which helps in sending mail with rich content and attachments from the web application.

## □ Authentication

User authentication is a common feature in web applications. Laravel eases designing authentication as it includes features such as **register**, **forgot password**.



## □ Redis

---

Laravel uses **Redis** to connect to an existing session.  
Redis interacts with session directly.

## □ Queues

Laravel includes queue services like emailing large number of users. These queues help in completing tasks in an easier manner without waiting for the previous task to be completed.

## □ Event and Command Bus

Laravel includes **Command Bus** which helps in executing commands and dispatch events in a simple way.



# Laravel - Installation

---

- For managing dependencies, Laravel uses **composer**. Make sure you have a Composer installed on your system before you install Laravel.
- You will have to follow the steps given below for installing Laravel onto your system –
- **Step 1** – Visit the following URL and download composer to install it on your system.
- <https://getcomposer.org/download/>
- **Step 2** – After the Composer is installed, check the installation by typing the Composer command in the command prompt as shown in the following screenshot.



```
c:\Administrator:C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\wamp\www\laravel>php artisan --version
Laravel Framework version 5.1.23 (LTS)

C:\wamp\www\laravel>cd\

C:\>composer
██

Composer version 1.0-dev (c7ed232ef42c2bd63cdba057b6c7c8043b37cd5a) 2015-10-29 0
9:52:59

Usage:
  command [options] [arguments]

Options:
  -h, --help          Display this help message
  -q, --quiet         Do not output any message
  -V, --version       Display this application version
  --ansi             Force ANSI output
  --no-ansi          Disable ANSI output
  --no-interaction   Do not ask any interactive question
  --profile          Display timing and memory usage information
  -d, --working-dir=WORKING-DIR If specified, use the given directory as workin
g directory.
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for norma
l output, 2 for more verbose output and 3 for debug
```

- 
- Step 3 – Create a new directory anywhere in your system for your new Laravel project. After that, move to path where you have created the new directory and type the following command there to install Laravel.
    - **`composer create-project laravel/laravel --prefer-dist`**
  - Now, we will focus on installation of version 5.7. In Laravel version 5.7, you can install the complete framework by typing the following command –
    - `composer create-project laravel/laravel test dev-develop`
    - The output of the command is as shown below –



```
➔ code composer create-project laravel/laravel test dev-develop
Installing laravel/laravel (dev-develop d6acad21cb2288713d9c09a31f9b4ab86f116039)
  - Installing laravel/laravel (dev-develop develop): Cloning develop from cache
Created project in test
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 71 installs, 0 updates, 0 removals
  - Installing vlucas/phpdotenv (v2.5.1): Loading from cache
  - Installing symfony/css-selector (v4.1.3): Loading from cache
  - Installing tijsverkoyen/css-to-inline-styles (2.2.1): Loading from cache
  - Installing symfony/polyfill-php72 (v1.9.0): Loading from cache
  - Installing symfony/polyfill-mbstring (v1.9.0): Loading from cache
  - Installing symfony/var-dumper (v4.1.3): Loading from cache
  - Installing symfony/routing (v4.1.3): Loading from cache
  - Installing symfony/process (v4.1.3): Loading from cache
  - Installing symfony/polyfill-ctype (v1.9.0): Loading from cache
  - Installing symfony/http-foundation (v4.1.3): Loading from cache
  - Installing symfony/event-dispatcher (v4.1.3): Loading from cache
  - Installing psr/log (1.0.2): Loading from cache
  - Installing symfony/debug (v4.1.3): Loading from cache
  - Installing symfony/http-kernel (v4.1.3): Loading from cache
  - Installing paragonie/random_compat (v9.99.99): Loading from cache
```



- 
- The Laravel framework can be directly installed with develop branch which includes the latest framework.
  - **Step 4** – The above command will install Laravel in the current directory. Start the Laravel service by executing the following command.
  - `php artisan serve`
  - **Step 5** – After executing the above command, you will see a screen as shown below –
- 



```
Administrator: C:\Windows\System32\cmd.exe - php artisan serve
C:\laravel-master\laravel>php artisan serve
Laravel development server started on http://localhost:8000/
```

- 
- Step 6 – Copy the URL underlined in gray in the above screenshot and open that URL in the browser. If you see the following screen, it implies Laravel has been installed successfully.



# Laravel - Configuration

---

- configuration files of Laravel are included in the **config** directory



# Maintenance

---

- Maintenance Mode
- Sometimes you may need to update some configuration values or perform maintenance on your website. In such cases, keeping it in **maintenance mode**, makes it easier for you. Such web applications which are kept in maintenance mode, throw an exception namely **MaintenanceModeException** with a status code of 503.
- You can enable the maintenance mode on your Laravel web application using the following command –
- `php artisan down`



ca C:\Windows\system32\cmd.exe

Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Admin>cd ..

C:\Users>cd ..

C:\>cd xampp

C:\xampp>cd htdocs

C:\xampp\htdocs>cd laravel-project

C:\xampp\htdocs\laravel-project>php artisan down  
Application is now in maintenance mode.

C:\xampp\htdocs\laravel-project>

- 
- Once you finish working on updates and other maintenance, you can disable the maintenance mode on your web application using following command –
  - `php artisan up p`



# Laravel - Routing

---

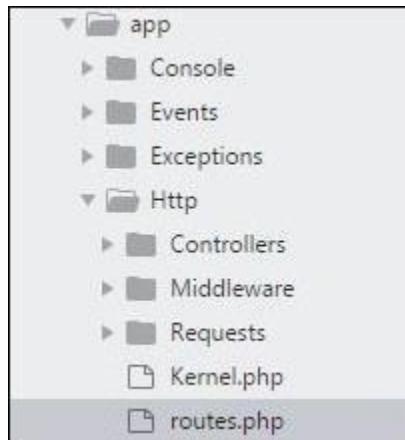
- In Laravel, all requests are mapped with the help of routes. Basic routing routes the request to the associated controllers.
- Routing in Laravel includes the following categories –
  1. Basic Routing
  2. Route parameters
  3. Named Routes



# 1. Basic Routing

---

- All the application routes are registered within the **app/routes.php** file. This file tells Laravel for the URIs it should respond to and the associated controller will give it a particular call. The sample route for the welcome page can be seen as shown in the screenshot given below –



---

- Route::get ('/',function ()
- {
- return view('welcome');
- }
- );



- 
- Example
  - Observe the following example to understand more about Routing –
  - **app/Http/routes.php**
  - <?phpRoute::get('/',function () {  
    return view('welcome');  
});



# resources/view/welcome.blade.php

---

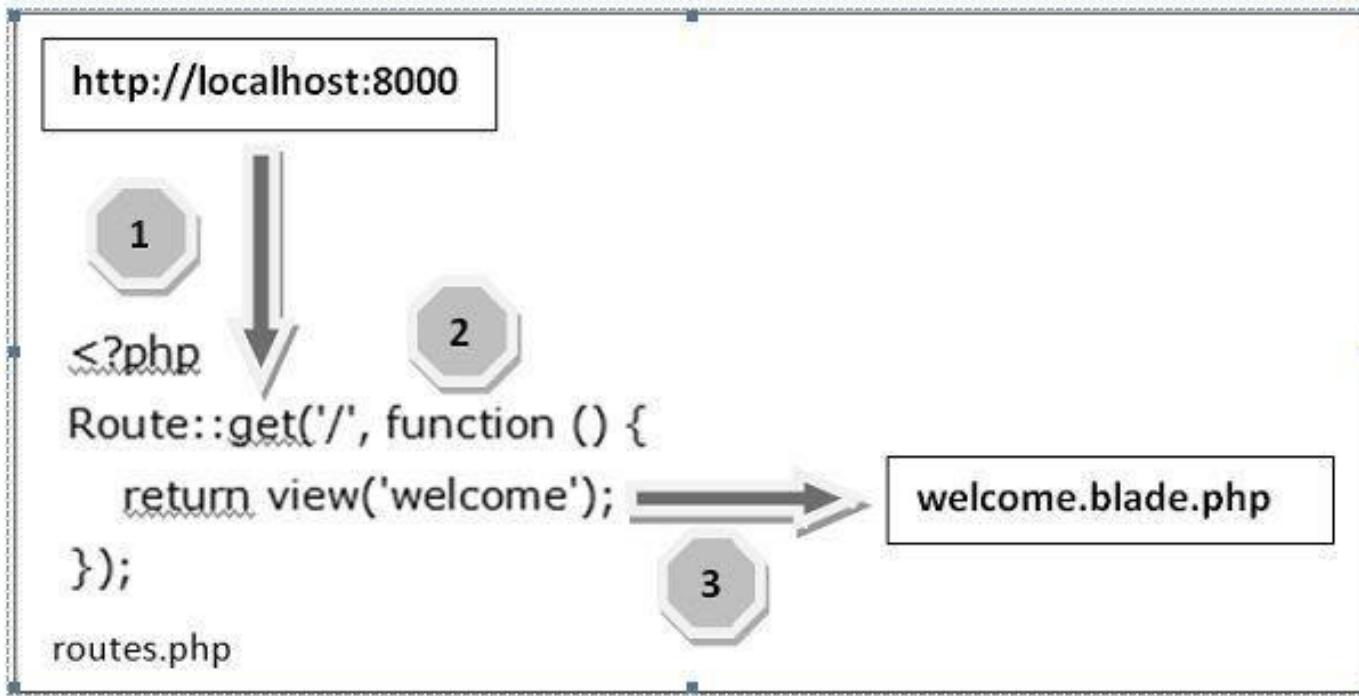
```
RESOURCES/VIEW/WELCOME.BLADE.PHP
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>Laravel</title>
        <link href =
"https://fonts.googleapis.com/css?family=Lato:100" rel =
"stylesheet"
            type = "text/css">

        <style>
html, body {
```



```
        height: 100%;  
    }  
    body {  
        margin: 0;  
        padding: 0;  
        width: 100%;  
        display: table;  
        font-weight: 100;  
        font-family: 'Lato';  
    }  
    .container {  
        text-align: center;  
        display: table-cell;  
        vertical-align: middle;  
    }  
    .content {  
        text-align: center;  
        display: inline-block;  
    }  
    .title {  
        font-size: 96px;  
    }  
    </style>  
</head>  
  
<body>  
<div class = "container">  
  
    <div class = "content">  
        <div class = "title">Laravel 5.1</div>  
    </div>  
  
    </div>  
</body>
```



- 
- Let us now understand the steps involved in routing mechanism in detail –
  - **Step 1** – Initially, we should execute the root URL of the application.
  - **Step 2** – Now, the executed URL should match with the appropriate method in the **route.php** file. In the present case, it should match the method and the root ('/') URL. This will execute the related function.
  - **Step 3** – The function calls the template file **resources/views/welcome.blade.php**. Next, the function calls the **view()** function with argument ‘**welcome**’ without using the **blade.php**.
  - This will produce the HTML output as shown in the image below –



---

# Laravel 5

---

## 2. Route Parameters

---

- Sometimes in the web application, you may need to capture the parameters passed with the URL. For this, you should modify the code in **routes.php** file.
- You can capture the parameters in **routes.php** file in two ways as discussed here –
  - Required Parameters
  - These parameters are those which should be mandatorily captured for routing the web application. For example, it is important to capture the user's identification number from the URL. This can be possible by defining route parameters as shown below –
    - `Route::get('ID/{id}',function($id) {`
    - `echo 'ID: '.$id;`
    - `});`



- 
- Optional Parameters
  - Sometimes developers can produce parameters as optional and it is possible with the inclusion of ? after the parameter name in URL. It is important to keep the default value mentioned as a parameter name. Look at the following example that shows how to define an optional parameter –
  - ```
Route::get('user/{name?}', function ($name = 'TutorialsPoint') { return $name; })
```
  - The example above checks if the value matches to **TutorialsPoint** and accordingly routes to the defined URL.
-

### 3. Named Routes

---

- Named routes allow a convenient way of creating routes. The chaining of routes can be specified using name method onto the route definition. The following code shows an example for creating named routes with controller –
  - `Route::get('user/profile', 'UserController@showProfile')->name('profile');`
  - The user controller will call for the function **showProfile** with parameter as **profile**. The parameters use name method onto the route definition.



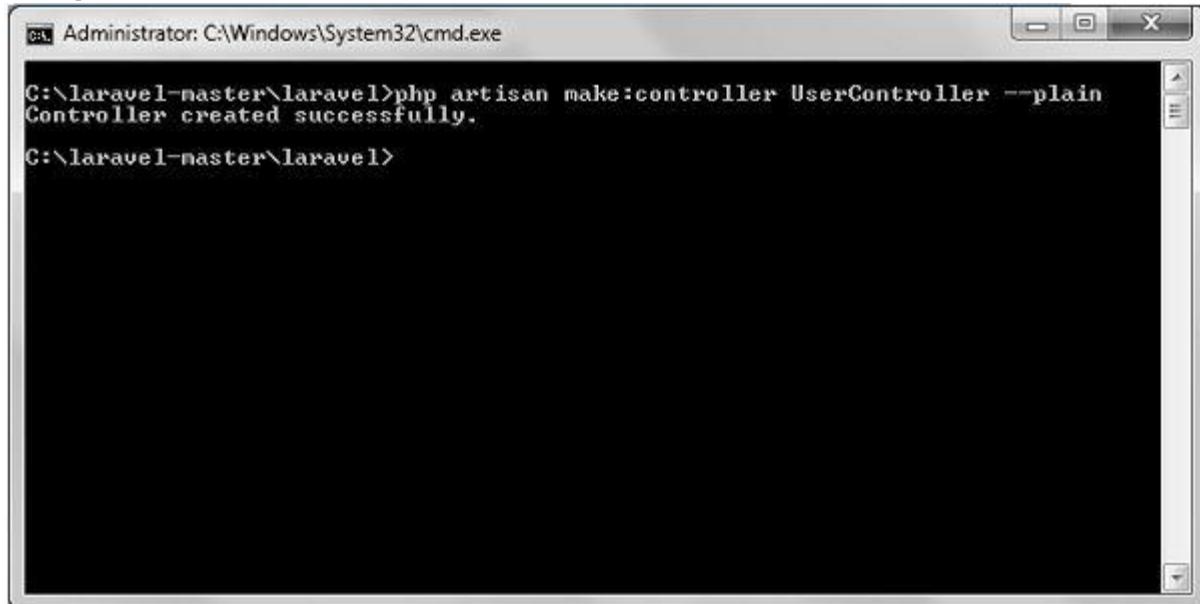
# Laravel - Controllers

---

- In the MVC framework, the letter 'C' stands for Controller. It acts as a directing traffic between Views and Models
- Creating a Controller
- Open the command prompt or terminal based on the operating system you are using and type the following command to create controller using the Artisan CLI (Command Line Interface).
- `php artisan make:controller <controller-name> --plain` Replace the `<controller-name>` with the name of your controller. This will create a plain constructor as we are passing the argument — `plain`. If you don't want to create a plain constructor, you can simply ignore the argument. The created constructor can be seen at **app/Http/Controllers**.
- You will see that some basic coding has already been done for you and you can add your custom coding. The created controller can be called from `routes.php` by the following syntax.
- Syntax
- `Route::get('base URI','controller@method');`



- 
- Example
  - Step 1 – Execute the following command to create **UserController**.
  - `php artisan make:controller UserController --plain`
  - Step 2 – After successful execution, you will receive the following output.



A screenshot of a Windows Command Prompt window titled "Administrator: C:\Windows\System32\cmd.exe". The window shows the command `php artisan make:controller UserController --plain` being run, followed by the message "Controller created successfully." The command prompt is located at the path `C:\laravel-master\laravel>`.

```
Administrator: C:\Windows\System32\cmd.exe
C:\laravel-master\laravel>php artisan make:controller UserController --plain
Controller created successfully.
C:\laravel-master\laravel>
```

- 
- Step 3 – You can see the created controller at **app/Http/Controller/UserController.php** with some basic coding already written for you and you can add your own coding based on your need.

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
use App\Http\Requests;  
use App\Http\Controllers\Controller;  
  
class UserController extends Controller {  
    //  
}
```

---



# Laravel - Views

---

- In MVC framework, the letter “V” stands for **Views**. It separates the application logic and the presentation logic. Views are stored in **resources/views** directory. Generally, the view contains the HTML which will be served by the application.
- Example
- Observe the following example to understand more about **Views** –
- Step 1 – Copy the following code and save it at **resources/views/test.php**
  - <html>
  - <body> <h1>Hello,World</h1>
  - ▶   </body> </html>

- Step 2 – Add the following line in `app/Http/routes.php` file to set the route for the above view.
- `app/Http/routes.php`
- ```
Route::get('/test', function() {  
    return view('test');  
});
```
- Step 3 – Visit the following URL to see the output of the view.
- `http://localhost:8000/test` Step 4 – The output will appear as shown in the following image.



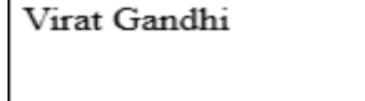
Hello, World



- Passing Data to Views
- While building application it may be required to pass data to the views. Pass an array to view helper function. After passing an array, we can use the key to get the value of that key in the HTML file.
- Example
- Observe the following example to understand more about passing data to views –
- Step 1 – Copy the following code and save it at **resources/views/test.php**
  - <html>
  - <body>
  - <h1><?php echo \$name; ?></h1>
  - </body>
  - </html>

- Step 2 – Add the following line in `app/Http/routes.php` file to set the route for the above view.
- 

- `app/Http/routes.php`
- ```
Route::get('/test',function() {
return view('test',['name'=>'Virat Gandhi']);
});
```
- Step 3 – The value of the key name will be passed to `test.php` file and `$name` will be replaced by that value.
- Step 4 – Visit the following URL to see the output of the view.
- `http://localhost:8000/test` Step 5 – The output will appear as shown in the following image.



- 
- Sharing Data with all Views
  - We have seen how we can pass data to views but at times, there is a need to pass data to all the views. Laravel makes this simpler. There is a method called **share()** which can be used for this purpose. The **share()** method will take two arguments, key and value. Typically **share()** method can be called from boot method of service provider. We can use any service provider, **AppServiceProvider** or our own service provider.
  - Example
  - Observe the following example to understand more about sharing data with all views –

- 
- Step 1 – Add the following line in **app/Http/routes.php** file.
  - **app/Http/routes.php**
  - ```
Route::get('/test',function() {  
    return view('test');  
});  
  
Route::get('/test2',function() {  
    return view('test2');  
});
```
-

- 
- Step 2 – Create two view files — **test.php** and **test2.php** with the same code. These are the two files which will share data. Copy the following code in both the files. **resources/views/test.php** & **resources/views/test2.php**
  - <html>
  - <body>
  - <h1><?php echo \$name; ?></h1>
  - </body>
  - </html>



- 
- Step 3 – Change the code of boot method in the file **app/Providers/AppServiceProvider.php** as shown below. (Here, we have used share method and the data that we have passed will be shared with all the views.) **app/Providers/AppServiceProvider.php**



```
<?php

namespace App\Providers;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider {

    /**
     * Bootstrap any application services.
     *
     * @return void
     */

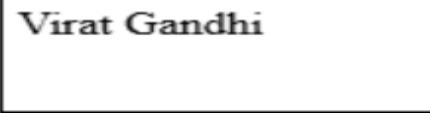
    public function boot() {
        view()->share('name', 'Virat Gandhi');
    }

    /**
     * Register any application services.
     *
     * @return void
     */

    public function register() {
        //
    }
}
```



- 
- **Step 4 – Visit the following URLs.**
  - `http://localhost:8000/test` `http://localhost:8000/test2`
  - **Step 5 – The output will appear as shown in the following image.**



# Laravel - Request

---

- Retrieving the Request URI
- The “path” method is used to retrieve the requested URI. The **is** method is used to retrieve the requested URI which matches the particular pattern specified in the argument of the method. To get the full URL, we can use the **url** method.
- Example
- Step 1 – Execute the below command to create a new controller called **UriController**.  
`php artisan make:controller UriController –plain`
- Step 2 – After successful execution of the URL, you will receive the following output –



Administrator: C:\Windows\System32\cmd.exe

```
C:\laravel-master\laravel>php artisan make:controller UriController --plain  
Controller created successfully.
```

```
C:\laravel-master\laravel>
```

- 
- Step 3 – After creating a controller, add the following code in that file.
  - **app/Http/Controllers/UriController.php**



```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\Http\Controllers\Controller;

class UriController extends Controller {

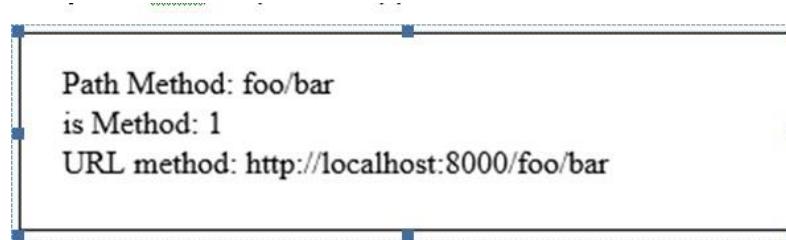
    public function index(Request $request) {
        // Usage of path method
        $path = $request->path();
        echo 'Path Method: '.$path;
        echo '<br>';

        // Usage of is method
        $pattern = $request->is('foo/*');
        echo 'is Method: '.$pattern;
        echo '<br>';

        // Usage of url method
        $url = $request->url();
        echo 'URL method: '.$url;
    }
}
```



- 
- Step 4 – Add the following line in the **app/Http/route.php** file.
  - **app/Http/route.php**
  - `Route::get('/foo/bar','UriController@index');`
  - Step 5 – Visit the following URL.
  - `http://localhost:8000/foo/bar`
  - Step 6 – The output will appear as shown in the following image.



# Retrieving Input

---

- The input values can be easily retrieved in Laravel. No matter what method was used “get” or “post”, the Laravel method will retrieve input values for both the methods the same way. There are two ways we can retrieve the input values.
- Using the `input()` method
- Using the properties of Request instance



- 
- Using the `input()` method
  - The `input()` method takes one argument, the name of the field in form. For example, if the form contains `username` field then we can access it by the following way.

```
$name = $request->input('username');
```
  
  - Using the properties of Request instance
  - Like the `input()` method, we can get the `username` property directly from the request instance.

```
$request->username
```
- 



- 
- Example
  - Observe the following example to understand more about Requests –
  - Step 1 – Create a Registration form, where user can register himself and store the form at **resources/views/register.php**



```
<head>
    <title>Form Example</title>
</head>

<body>
    <form action = "/user/register" method = "post">
        <input type = "hidden" name = "_token" value = "<?php echo
csrf_token() ?>">

        <table>
            <tr>
                <td>Name</td>
                <td><input type = "text" name = "name" /></td>
            </tr>
            <tr>
                <td>Username</td>
                <td><input type = "text" name = "username" /></td>
            </tr>
            <tr>
                <td>Password</td>
                <td><input type = "text" name = "password" /></td>
            </tr>
            <tr>
                <td colspan = "2" align = "center">
                    <input type = "submit" value = "Register" />
                </td>
            </tr>
        </table>

    </form>
</body>
```



- 
- **Step 2** – Execute the below command to create a **UserRegistration** controller.
  - `php artisan make:controller UserRegistration --plain`
  - **Step 3** – After successful execution of the above step, you will receive the following output –
  - **Step 4** – Copy the following code in **app/Http/Controllers/UserRegistration.php** controller.
  - **app/Http/Controllers/UserRegistration.php**
-

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\Http\Controllers\Controller;

class UserRegistration extends Controller {
    public function postRegister(Request $request) {
        //Retrieve the name input field
        $name = $request->input('name');
        echo 'Name: '.$name;
        echo '<br>';

        //Retrieve the username input field
        $username = $request->username;
        echo 'Username: '.$username;
        echo '<br>';

        //Retrieve the password input field
        $password = $request->password;
    }
}
```



- 
- Step 5 – Add the following line in **app/Http/routes.php** file.
  - **app/Http/routes.php**
  - ```
Route::get('/register',function() {  
    return view('register');  
});  
  
Route::post('/user/register',array('uses'=>'UserRegistration@postRegister'));
```
-

- 
- **Step 6** – Visit the following URL and you will see the registration form as shown in the below figure. Type the registration details and click Register and you will see on the second page that we have retrieved and displayed the user registration details.
  - <http://localhost:8000/register>
  - **Step 7** – The output will look something like as shown in below the following images.
-

|   |   |
|---|---|
| Name                                    | <input type="text" value="Virat"/>        |
| Username                                | <input type="text" value="virat.gandhi"/> |
| Password                                | <input type="password" value="asdadf"/>   |
| <input type="button" value="Register"/> |   |



Name: Virat  
Username: virat.gandhi  
Password: asdadf

# Laravel - Response

---

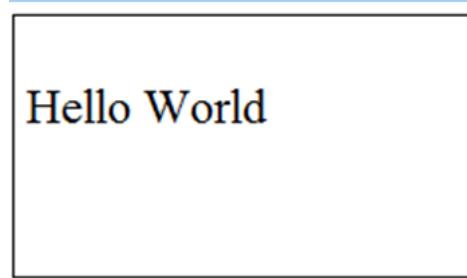
- A web application responds to a user's request in many ways depending on many parameters. This chapter explains you in detail about responses in Laravel web applications.
  - Basic Response
  - Laravel provides several different ways to return response. Response can be sent either from route or from controller. The basic response that can be sent is simple string as shown in the below sample code. This string will be automatically converted to appropriate HTTP response.
- 



- 
- Example
  - Step 1 – Add the following code to app/Http/routes.php file.
  - **app/Http/routes.php**
  - ```
Route::get('/basic_response',function () {  
    return 'Hello World';  
});
```
  - Step 2 – Visit the following URL to test the basic response.
  - [http://localhost:8000/basic\\_response](http://localhost:8000/basic_response)
- 



- 
- Step 3 – The output will appear as shown in the following image.



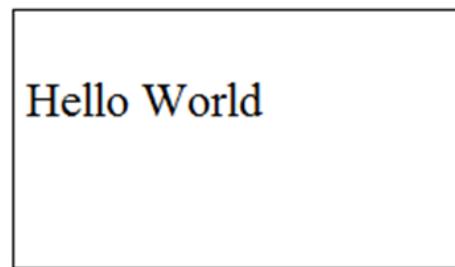
- 
- Attaching Cookies
  - The `withcookie()` helper method is used to attach cookies. The cookie generated with this method can be attached by calling `withcookie()` method with response instance. By default, all cookies generated by Laravel are encrypted and signed so that they can't be modified or read by the client.
  - Example
  - Observe the following example to understand more about attaching cookies –
  - Step 1 – Add the following code to `app/Http/routes.php` file.
    - ▶ □ `app/Http/routes.php`

---

- Route::get('/cookie',function() {  
    return response("Hello", 200)->header('Content-Type',  
        'text/html')  
    ->withcookie('name','Virat Gandhi');  
});

---

- 
- Step 2 – Visit the following URL to test the basic response.
  - <http://localhost:8000/cookie> Step 3 – The output will appear as shown in the following image



- 
- **JSON Response**
  - JSON response can be sent using the `json` method. This method will automatically set the Content-Type header to `application/json`. The `json` method will automatically convert the array into appropriate json response.
  - **Example**
  - Observe the following example to understand more about JSON Response –
  - **Step 1** – Add the following line in `app/Http/routes.php` file.
  - **app/Http/routes.php**
-

- Route::get('json',function() {
- return response()->json(['name' => 'Virat Gandhi', 'state' => 'Gujarat']);
- });
- **Step 2** – Visit the following URL to test the json response.
- <http://localhost:8000/json>
- **Step 3** – The output will appear as shown in the following image.

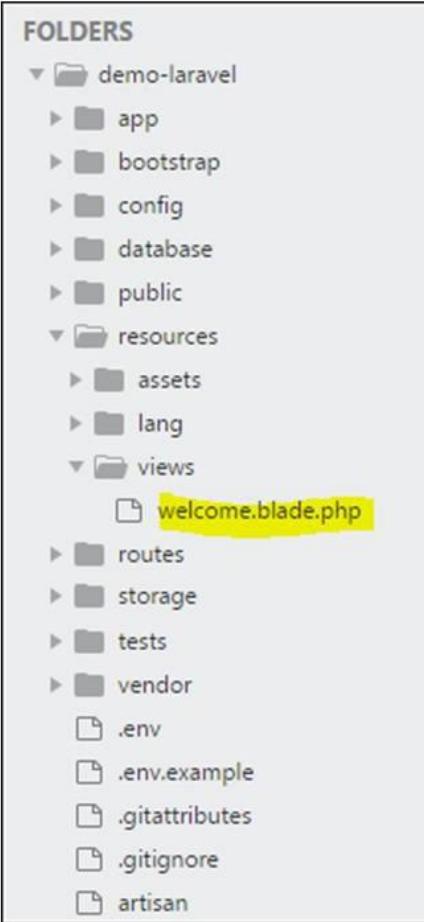
```
{"name":"Virat Gandhi","state":"Gujrat"}
```

# Laravel - Blade Templates

---

- Laravel 5.1 introduces the concept of using **Blade**, a templating engine to design a unique layout. The layout thus designed can be used by other views, and includes a consistent design and structure.
- When compared to other templating engines, Blade is unique in the following ways –
- It does not restrict the developer from using plain PHP code in views.
- The blade views thus designed, are compiled and cached until they are modified.





- 
- The complete directory structure of Laravel is shown in the screenshot given here.
  - You can observe that all views are stored in the **resources/views** directory and the default view for Laravel framework is **welcome.blade.php**.
  - Please note that other blade templates are also created similarly.
-

- 
- Steps for Creating a Blade Template Layout
  - You will have to use the following steps to create a blade template layout –
    - Step 1
    - Create a layout folder inside the **resources/views** folder. We are going to use this folder to store all layouts together.
    - Create a file name **master.blade.php** which will have the following code associated with it –
-

---

- `<html>`
- `<head>`
- `<title>DemoLaravel - @yield('title')</title>`
- `</head>`
- `<body>`
- `@yield('content')`
- `</body>`
- `</html>`



- Step 2
- In this step, you should extend the layout. Extending a layout involves defining the child elements. Laravel uses the **Blade @extends** directive for defining the child elements.
- When you are extending a layout, please note the following points –
  - Views defined in the Blade Layout injects the container in a unique way.
  - Various sections of view are created as child elements.
  - Child elements are stored in layouts folder as **child.blade.php**
  - An example that shows extending the layout created above is shown here –



```
@extends('layouts.app')
@section('title', 'Page Title')
@section('sidebar')
    @parent


This refers to the master sidebar.


@endsection
@section('content')


This is my body content.


@endsection
```



---

- Step 3

- To implement the child elements in views, you should define the layout in the way it is needed.



# Laravel - Session

---

- Sessions are used to store information about the user across the requests. Laravel provides various drivers like **file**, **cookie**, **apc**, **array**, **Memcached**, **Redis**, and **database** to handle session data. By default, file driver is used because it is lightweight. Session can be configured in the file stored at **config/session.php**.



- 
- Accessing Session Data
  - To access the session data, we need an instance of session which can be accessed via HTTP request. After getting the instance, we can use the **get()** method, which will take one argument, “**key**”, to get the session data.
  - `$value = $request->session()->get('key');`
  - You can use **all()** method to get all session data instead of **get()** method.

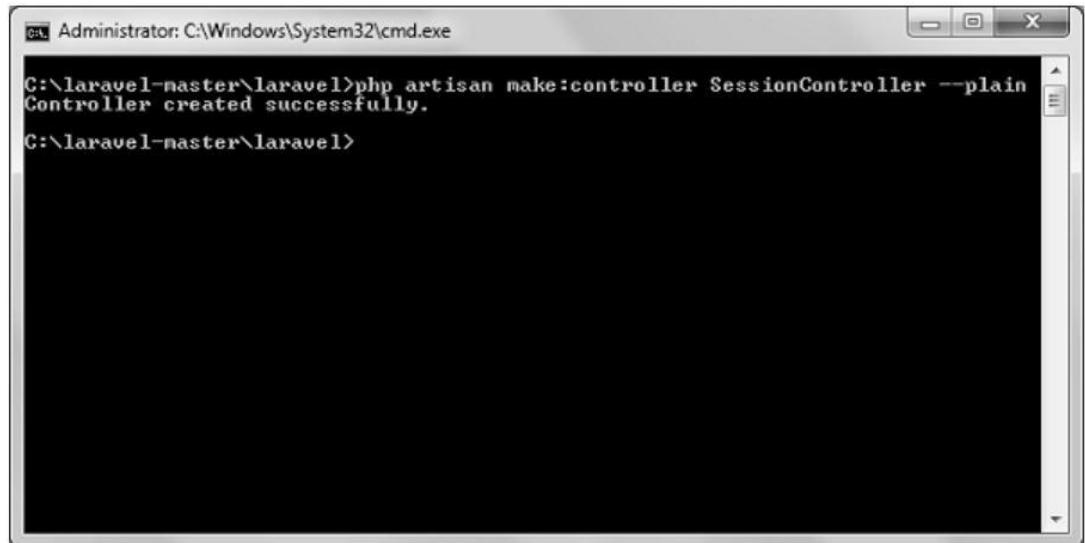


- 
- Storing Session Data
  - Data can be stored in session using the `put()` method.  
The `put()` method will take two arguments,  
the “key” and the “value”.
  - `$request->session()->put('key', 'value');`



- 
- Deleting Session Data
  - The **forget()** method is used to delete an item from the session. This method will take “key” as the argument.
  - `$request->session()->forget('key');` Use **flush()** method instead of **forget()** method to delete all session data. Use the **pull()** method to retrieve data from session and delete it afterwards. The **pull()** method will also take **key** as the argument. The difference between the **forget()** and the **pull()** method is that **forget()** method will not return the value of the session and **pull()** method will return it and delete that value from session.
-

- 
- Example
  - Step 1 – Create a controller called **SessionController** by executing the following command.
  - `php artisan make:controller SessionController --plain`  
Step 2 – After successful execution, you will receive the following output –



A screenshot of a Windows Command Prompt window titled "Administrator: C:\Windows\System32\cmd.exe". The window shows the following text output:

```
C:\laravel-master\laravel>php artisan make:controller SessionController --plain
Controller created successfully.

C:\laravel-master\laravel>
```



- 
- Step 3 – Copy the following code in a file at
  - **app/Http/Controllers/SessionController.php.**
  - **app/Http/Controllers/SessionController.php**



```
<?php

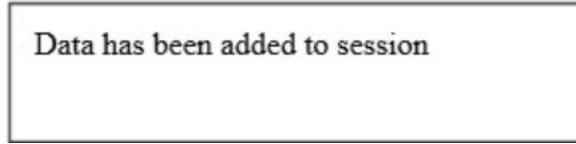
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\Http\Controllers\Controller;

class SessionController extends Controller {
    public function accessSessionData(Request $request) {
        if($request->session()->has('my_name'))
            echo $request->session()->get('my_name');
        else
            echo 'No data in the session';
    }
    public function storeSessionData(Request $request) {
        $request->session()->put('my_name','Virat Gandhi');
        echo "Data has been added to session";
    }
    public function deleteSessionData(Request $request) {
        $request->session()->forget('my_name');
        echo "Data has been removed from session.";
    }
}
```



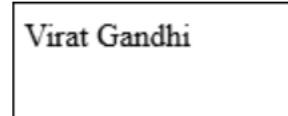
- Step 4 – Add the following lines at app/Http/routes.php file.
- app/Http/routes.php
- Route::get('session/get','SessionController@accessSessionData');
- Route::get('session/set','SessionController@storeSessionData');
- Route::get('session/remove','SessionController@deleteSessionData');
- Step 5 – Visit the following URL to set data in session.**
- <http://localhost:8000/session/set> **Step 6 – The output will appear as shown in the following image.**



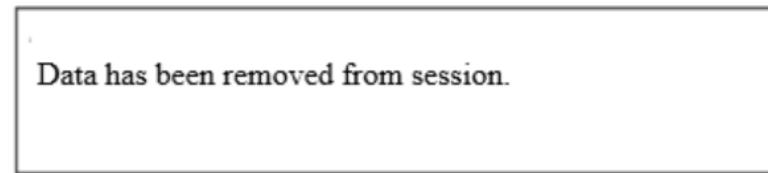
Data has been added to session



- 
- **Step 7 – Visit the following URL to get data from session.**
  - <http://localhost:8000/session/get> **Step 8 – The output will appear as shown in the following image.**
  - **Step 9 – Visit the following URL to remove session data.**
  - <http://localhost:8000/session/remove> **Step 10 – You will see a message as shown in the following image.**



Virat Gandhi



Data has been removed from session.



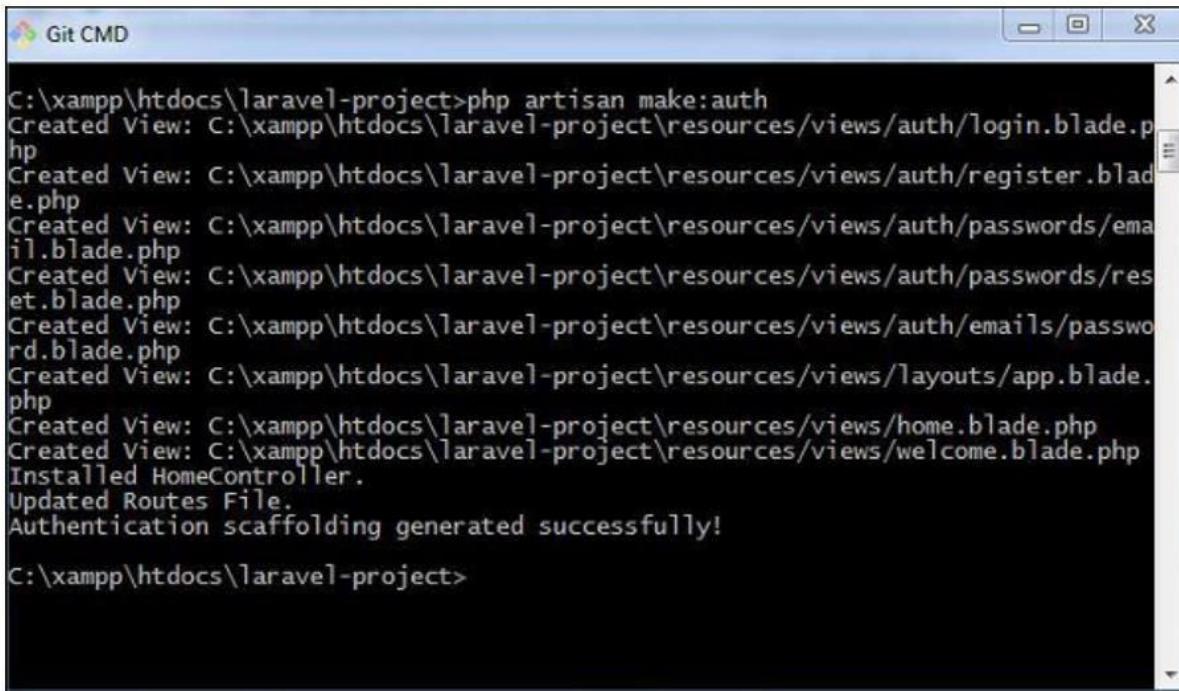
# Laravel - Authentication

---

- Authentication is the process of identifying the user credentials. In web applications, authentication is managed by sessions which take the input parameters such as email or username and password, for user identification. If these parameters match, the user is said to be authenticated.
- Command
- Laravel uses the following command to create forms and the associated controllers to perform authentication –
  - `php artisan make:auth`



This command helps in creating authentication scaffolding successfully, as shown in the following screenshot –



The screenshot shows a Windows Command Prompt window titled "Git CMD". The command entered was "php artisan make:auth". The output of the command is displayed in the terminal window, showing the creation of various files and the installation of a controller and routes. The output includes:

```
C:\xampp\htdocs\laravel-project>php artisan make:auth
Created View: C:\xampp\htdocs\laravel-project\resources\views\auth\login.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources\views\auth\register.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources\views\auth\passwords\email.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources\views\auth\passwords\reset.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources\views\auth\emails\password.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources\views\layouts\app.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources\views\home.blade.php
Created View: C:\xampp\htdocs\laravel-project\resources\views\welcome.blade.php
Installed HomeController.
Updated Routes File.
Authentication scaffolding generated successfully!
C:\xampp\htdocs\laravel-project>
```

- 
- Controller
  - The controller which is used for the authentication process is **HomeController**.



```
namespace App\Http\Controllers;

use App\Http\Requests;
use Illuminate\Http\Request;

class HomeController extends Controller{
    /**
     * Create a new controller instance.
     *
     * @return void
     */

    public function __construct() {
        $this->middleware('auth');
    }

    /**
     * Show the application dashboard.
     *
     * @return \Illuminate\Http\Response
     */

    public function index() {
        return view('home');
    }
}
```



- As a result, the scaffold application generated creates the login page and the registration page for performing authentication. They are as shown below –

Login

The screenshot shows a web browser window with a light gray header bar. On the left of the header is the word "Laravel". To its right are two links: "Home" and "Logout". On the far right of the header are two more links: "Login" and "Register". Below the header is a white rectangular form area with a thin gray border. At the top left of this area is the word "Login" in a small, dark font. Below it are two input fields: one labeled "E-Mail Address" and another labeled "Password", both with placeholder text. Underneath these fields is a small checkbox labeled "Remember Me". At the bottom left is a blue rectangular button with a white play icon and the word "Login". To the right of this button is a link with the text "Forgot Your Password?".

## Registration

A screenshot of a web application's registration page. The page has a header with "Laravel" and "Home" on the left, and "Login" and "Register" on the right. The main content area is titled "Register". It contains four input fields: "Name", "E-Mail Address", "Password", and "Confirm Password". Below these fields is a blue "Register" button with a user icon.

Laravel Home

Login Register

Register

Name

E-Mail Address

Password

Confirm Password

Register

- 
- Manually Authenticating Users
  - Laravel uses the **Auth** façade which helps in manually authenticating the users. It includes the **attempt** method to verify their email and password.
  - Consider the following lines of code for **LoginController** which includes all the functions for authentication –



```
<?php

// Authentication mechanism
namespace App\Http\Controllers;

use Illuminate\Support\Facades\Auth;

class LoginController extends Controller{
    /**
     * Handling authentication request
     *
     * @return Response
     */

    public function authenticate() {
        if (Auth::attempt(['email' => $email, 'password' => $password])) {

            // Authentication passed...
            return redirect()->intended('dashboard');
        }
    }
}
```



# Laravel - Authorization

- Difference between Authentication and Authorization
- Before proceeding further into learning about the authorization process in Laravel, let us understand the difference between authentication and authorization.
- In **authentication**, the system or the web application identifies its users through the credentials they provide. If it finds that the credentials are valid, they are authenticated, or else they are not.
- In **authorization**, the system or the web application checks if the authenticated users can access the resources that they are trying to access or make a request for. In other words, it checks their rights and permissions over the requested resources. If it finds that they can access the resources, it means that they are authorized.
- Thus, **authentication** involves checking the validity of the user credentials, and **authorization** involves checking the rights and permissions over the resources that an authenticated user has.

- 
- Authorization Mechanism in Laravel
  - Laravel provides a simple mechanism for authorization that contains two primary ways, namely **Gates** and **Policies**.
  - Writing Gates and Policies
  - Gates are used to determine if a user is authorized to perform a specified action. They are typically defined in **App/Providers/AuthServiceProvider.php** using Gate facade. Gates are also functions which are declared for performing authorization mechanism.
  - Policies are declared within an array and are used within classes and methods which use authorization mechanism.
- 



- 
- The following lines of code explain you how to use Gates and Policies for authorizing a user in a Laravel web application. Note that in this example, the `boot` function is used for authorizing the users.



```
<?php

namespace App\Providers;

use Illuminate\Contracts\Auth\Access\Gate as GateContract;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;

class AuthServiceProvider extends ServiceProvider{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        'App\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Register any application authentication / authorization services.
     *
     * @param \Illuminate\Contracts\Auth\Access\Gate $gate
     * @return void
     */
    public function boot(GateContract $gate) {
        $this->registerPolicies($gate);
    }
}
```



# Laravel - Pagination Customizations

---

- Laravel includes a feature of pagination which helps a user or a developer to include a pagination feature. Laravel paginator is integrated with the query builder and Eloquent ORM. The paginate method automatically takes care of setting the required limit and the defined offset. It accepts only one parameter to paginate i.e. the number of items to be displayed in one page.
- Laravel 5.7 includes a new pagination method to customize the number of pages on each side of the paginator. The new method no longer needs a custom pagination view.
- The custom pagination view code demonstration is mentioned below –

```
<?php  
namespace App\Http\Controllers;  
use Illuminate\Support\Facades\DB;  
use App\Http\Controllers\Controller;  
class UserController extends Controller{  
    /**  
     * Show all of the users for the application.  
     *  
     * @return Response  
    */  
    public function index() {  
        $users = DB::table('users')->paginate(15);  
        return view('user.index', ['users' => $users]);  
    }  
}
```



- 
- The new pagination customization as per Laravel standards is mentioned below –
  - <?php User::paginate(10)->onEachSide(5);
  
  - Note that **onEachSide** refers to the subdivision of each pagination records with 10 and subdivision of 5.
-

# Laravel-Migration

---

- Laravel Migrations
- Creation of table in database in laravel is an easy task with laravel. Database schema can be easily modified and shared. Modification can be addition of new column or deletion of existing columns.
- Laravel migrations can add new column or delete records in DB without deleting actual records. This is helpful in projects having different teams working on different modules. No need to pass on the SQL file to the different systems. It act like version control of the database.



- 
- Make Migratoins
  - To make migrations
  - **make:migrations**
  - **command** is used. The general syntax of making migrations is:
  - `php artisan make:migration message_you_want_to_remember`



□ `php artisan make:migration create_blogs_table --create=blogs //php artisan make:migration add_votes_to_blogs_table --table=blogs`

```
Command Prompt - php artisan make:migration create_users_table --create=users
Microsoft Windows [Version 10.0.18362.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Sanjeev>E:

E:\>cd xampp

E:\xampp>cd htdocs blog
The system cannot find the path specified.

E:\xampp>cd htdocs blogss
The system cannot find the path specified.

E:\xampp>cd htdocs

E:\xampp\htdocs>cd blogss

E:\xampp\htdocs\blogss>php artisan make:migration create_users_table --create=users
Created Migration: 2020_01_07_053408_create_users_table
```

- 
- Migration Structure
  - Migrations includes two methods inside the file: **up** and **down**. Up method helps to add the new updates to the database i.e. creating new tables or columns whereas down reverse the operation performed by up. To create and modify tables laravel schema builder is used, below shown the structure of a migration file:
-

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateBlogsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('blogs', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('title');
            $table->string('desciption');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('blogs');
    }
}
```

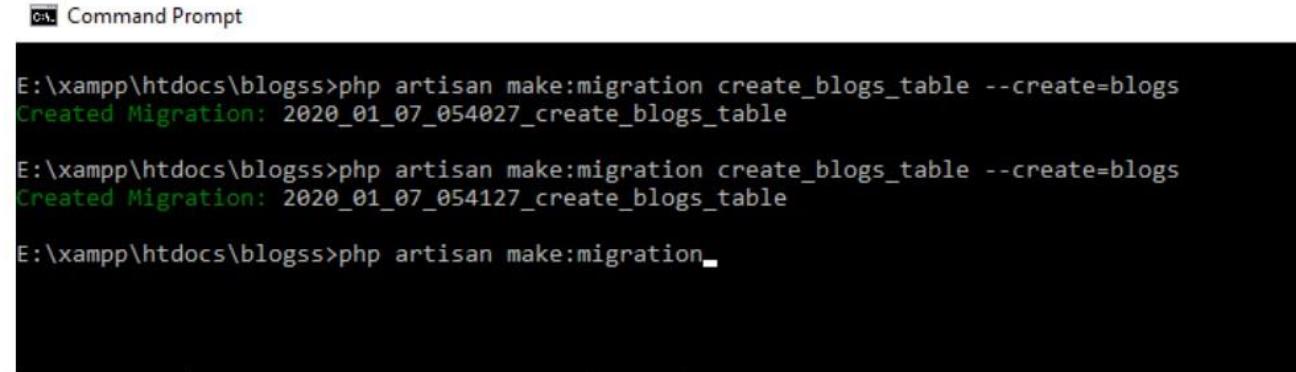


```
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('blogs');
    }
?>
```



- Migrations can be run by the command
- `php artisan make:migration`
- For some destructive migrations --force flag is used to run without prompt of confirmation like so:
- `php artisan migrate --force`



```
Command Prompt

E:\xampp\htdocs\blogss>php artisan make:migration create_blogs_table --create=blogs
Created Migration: 2020_01_07_054027_create_blogs_table

E:\xampp\htdocs\blogss>php artisan make:migration create_blogs_table --create=blogs
Created Migration: 2020_01_07_054127_create_blogs_table

E:\xampp\htdocs\blogss>php artisan make:migration
```

- 
- After Doing Migration Check Your database Blogs Table created



# Laravel database

---

- Laravel Databases
  - Database interaction in laravel is easy among all other server side scripting framework with variery of database backends either SQL, query builder or eloquent ORM.
  - **Various DB supported by laravel are:**
  - MySQL 5.6+
  - PostgreSQL 9.4+
  - SQLite 3.8.8+
  - SQL Server 2017+
- 



- 
- Configuration of DB in Laravel
  - The directory having database connections are located inside config folder.
  - config/database.php
  - All DB connections are defined and default connections can be set. Laravel uses Laravel Homestead virtual machine for development on local machine which can be modified according to local database.
-

- 
- SQLite Configuration:
  - A SQLite DB can be created with the command touch database/database.sqlite
  - And can be configured by setting up by using database absolute path:
    - DB\_CONNECTION=sqlite  
DB\_DATABASE=/absolute/path/to/database.sqlite



- 
- Multiple DB Connections:
  - Laravel supports multiple database connections and handle easily with **connection()** method on DB facade. Connections can be accessed in such a way as shown below:
    - \$users = DB::connection('DIFF\_DB')->select(...);
    - Make sure the name passed in connection method should correspond to connection listed in **config/database.php** file. PDO instance can also be used on the connection instance by syntax shown below by using **getPdo()** method:
      - \$pdo = DB::connection()->getPdo();



- 
- Running RAW SQL Queries
  - After DB configuration you may simply run queries as used in php using **DB facade** which provides method for each type of query: select, insert, update, delete and other statements. Below shown example and screenshot of the **select** query and other queries will work in the same manner.

**Make UserController.php file and paste the following code:**



```
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    public function index()
    {
        $users = DB::select('select * from users where id = ?', [1]);

        return view('user.index', ['users' => $users]);
    }
}
```



- 
- In web.php file add code mentioned below:
  - Route::get('/database','UserController@index');
  - You must create a **users** DB in your phpmyadminpannel.
  - Above query the select() method consist of two parameters: select query and other is parameter binding that is to be bound with query. Instead of using ? in above query named binding can be used as shown below:
  - \$results = DB::select('select \* from users where id = :id', ['id' => 1]);
- 



Screenshot of structure of the working is mentioned below:

### Output :

The screenshot shows the phpMyAdmin interface for a database named 'laravel'. The left sidebar lists databases, schemas, and tables under the 'laravel' schema. The 'users' table is selected. The main area displays the table structure with columns 'id' and 'name', and a single row of data: id 1, name phptpoint. Navigation tabs at the top include Browse, Structure, SQL, Search, and Import.

id	name
1	phptpoint

### Output :

#### Output :

A screenshot of a browser window with the address bar showing the URL: 127.0.0.1:8000/database. The page content area displays the text 'phptpoint'.

- Insert Query:
- DB::insert('insert into users (id, name) values (?,?)', [2, 'Believ Master']);

Output :

phpMyAdmin

Server: 127.0.0.1 » Databases

Browse Structure

Show all Number of rows:

+ Options

	id	name
1	1	phptpoint
2	2	Believe Master

Show all Number of rows:

The screenshot shows the 'Output' section of phpMyAdmin. On the left, there's a sidebar with 'Recent' and 'Favorites' tabs, a search bar, and a tree view of databases ('laravel', 'migrations', 'users') and tables ('New'). The main area shows the 'Structure' tab for the 'users' table. It has two rows of data: one with id 1 and name 'phptpoint', and another with id 2 and name 'Believe Master'. There are also 'Show all' and 'Number of rows:' dropdowns.

- Update Query:
- \$update = DB::update("update users set name = 'PHPTPOINT' where id = ?", [1]);

**Output :**

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
- Database:** laravel
- Table:** users
- Browse Tab:** Active
- Search Bar:** Show all, Number of rows: All
- Table Data:**

id	name
1	PHPTPOINT
2	Believe Master
- Left Sidebar:** Shows the database structure with 'information\_schema', 'laravel' schema, and tables 'laravel', 'migrations', and 'users'.

- Delete Query:
- `$delete = DB::delete('delete from users');`

**Output :**

The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible with the 'laravel' schema selected, containing tables 'users', 'migrations', and 'New'. The main panel shows the results of a query: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0009 seconds.)'. The SQL query listed is 'SELECT \* FROM `users`'. Below the results, there is a table header with columns 'id' and 'name', and a 'Query results operations' button.

- 
- Drop Query:
  - DB::statement('drop table users');

