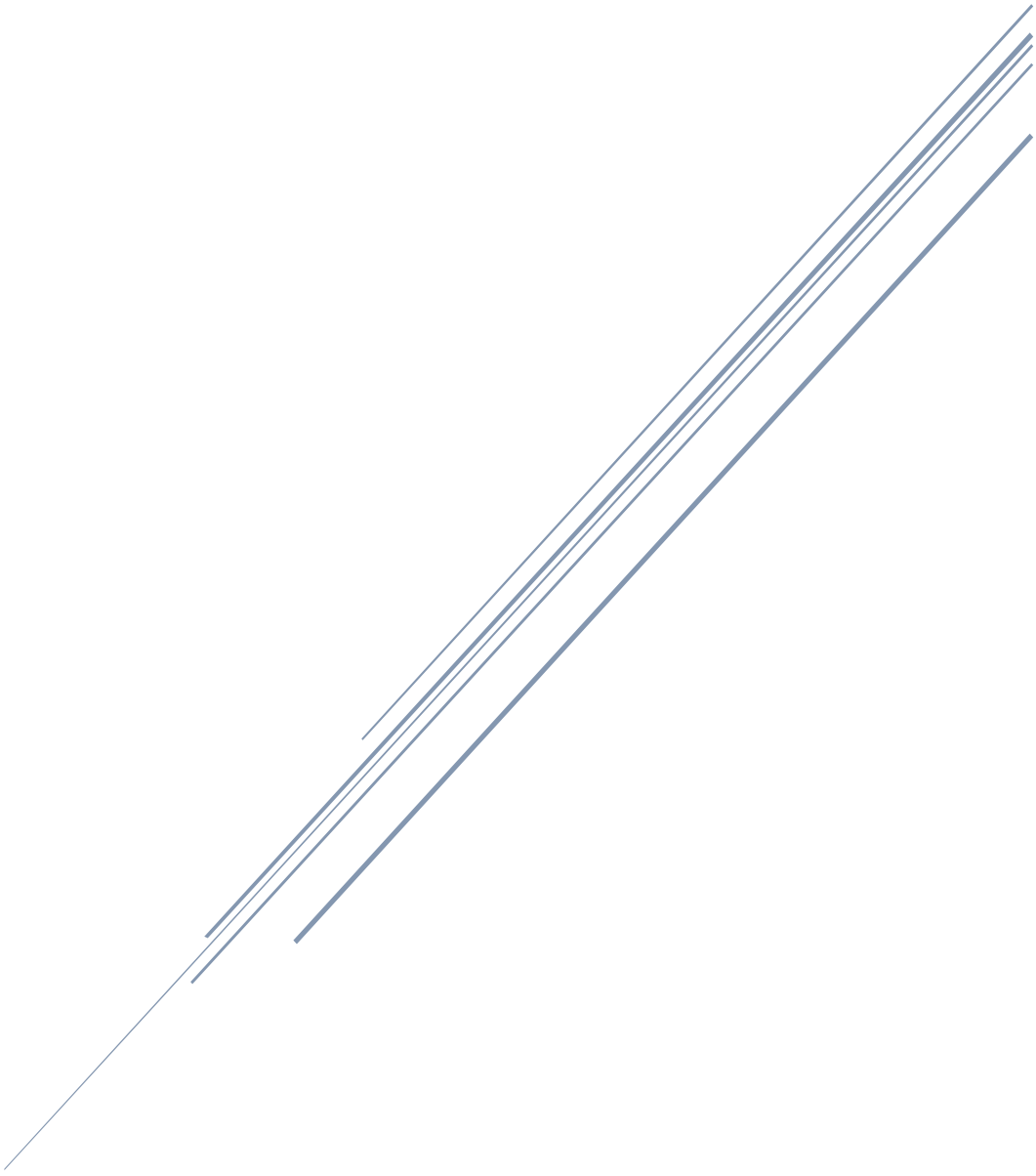
ADVANCED PHP

Trainer

MOHAMMED SHOUKAT ALI



Sai Pali Institute of Technology & Management

Advanced PHP Notes

How to Install Composer on Windows

Composer is an application that is used for *dependency management* in standard format in **PHP**. Composer lets you declare the libraries on which your PHP project depends and manage it for you. It installs and updates the libraries that you need for your project.

By default, Composer doesn't install anything globally; it manages dependencies based on per-project and installs them in the directory (*vendor*) inside your project.

Composer offers two things for our projects:

- Helps in dependency management for PHP.
- Helps in file auto-loading in the project.

The Composer is highly energized by Node's NPM and Ruby's bundle.

Suppose a scenario:

1. You have a PHP project which depends on several libraries.
2. And some of these libraries also depend on other libraries.

Composer fulfils the requirement of the above scenario as:

1. It lets you declare the libraries your project depends on.
2. It finds out which versions of the package you require to install in your project, and it downloads and installs for you.
3. It also allows updating all your dependencies in a single command.

System requirements to install Composer:

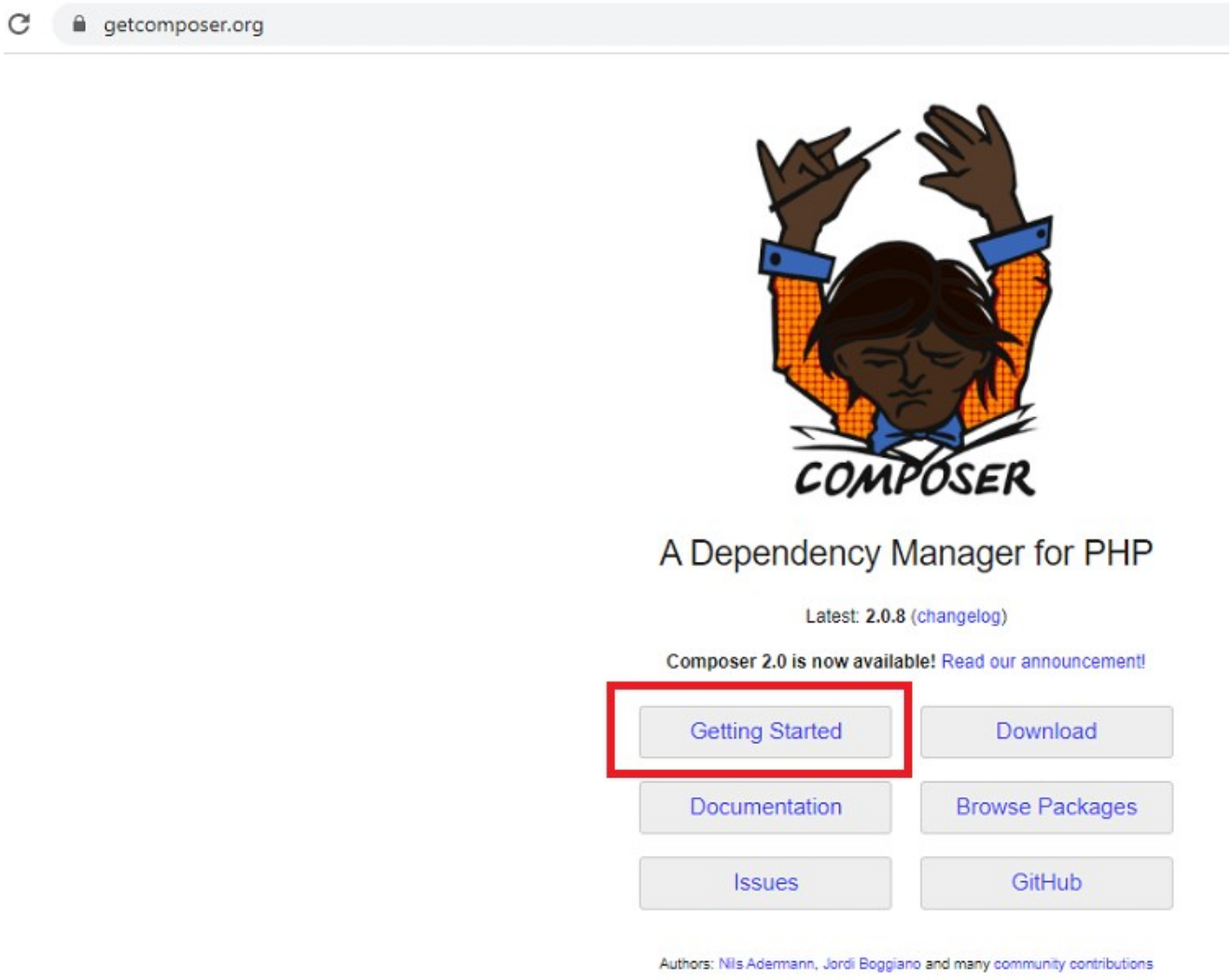
Composer requires installed and successfully running ***PHP 5.3.2+*** on the machine. It also requires some sensitive PHP settings and compiles flags. Installing the Composer packages from source rather than zip archive, you will require git, svn, fossil or hg based on the package version-controlled. A Composer is a multi-platform tool that runs equally on Windows OS, Linux and macOS.

Install Composer on Windows

There are two ways to install Composer on the Windows operating system: the first one is using Composer installer setup, and the other is by manual installation using script.

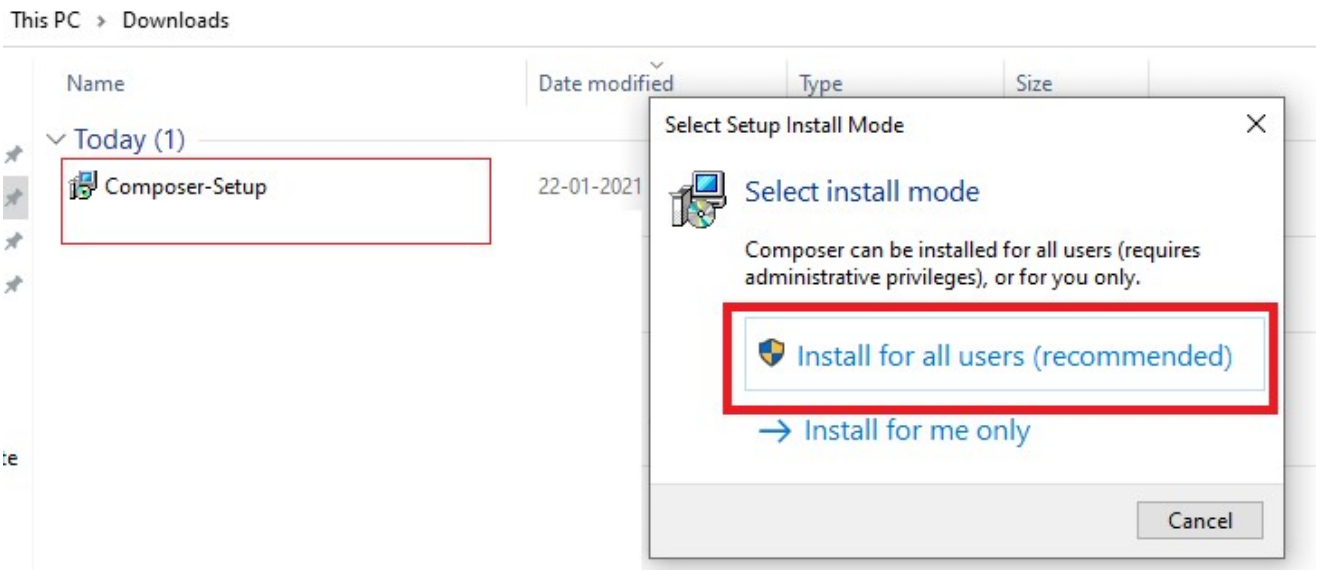
1. Using Installer

a) Installing Composer using Composer installer setup is the easiest way to install it on Windows operating system. Launch your default browser and visit https://getcomposer.org and click on the "Getting Started" button. Under the "*Installation - Windows*" section, click on the "*Using the Installer*" option; it will take you to the "Using the Installer" section.



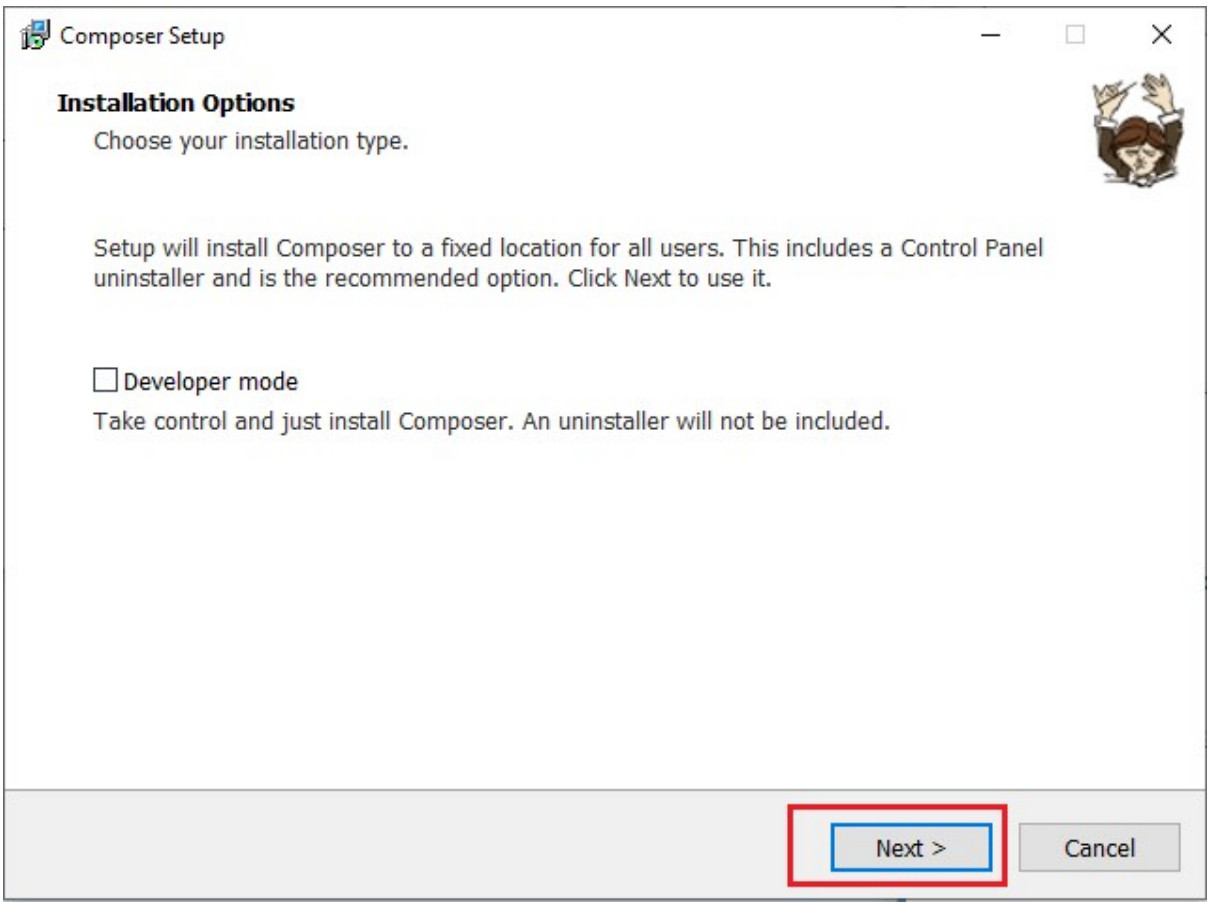
b) Click on the **Composer-Setup.exe** link to download Composer setup on your device. After downloading the setup, run it to install and follow the instructions.

c) Open the downloaded Composer-Setup and click on the "Install for all users", which is recommended option to install Composer setup.

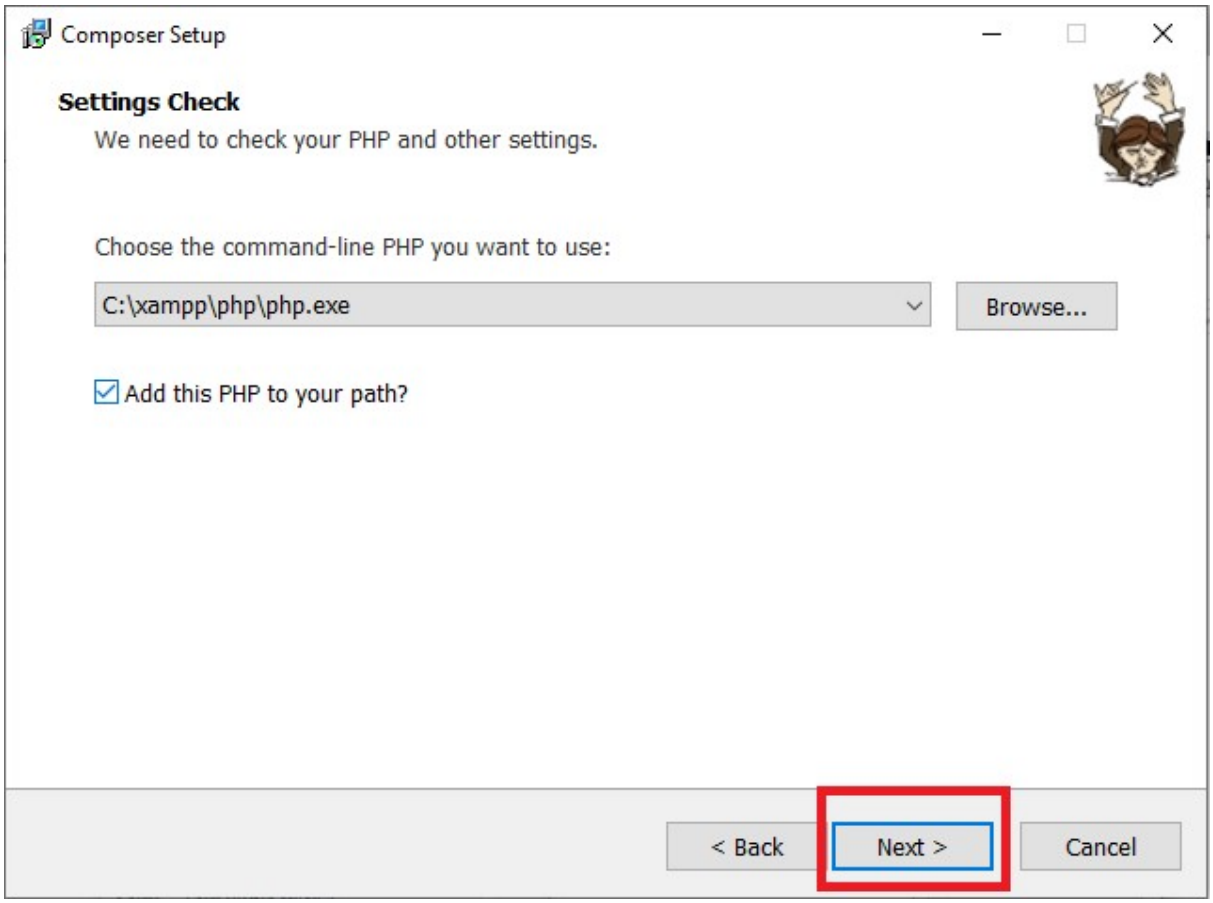


d) On the pop-up screen, click on **YES** to allow installation.

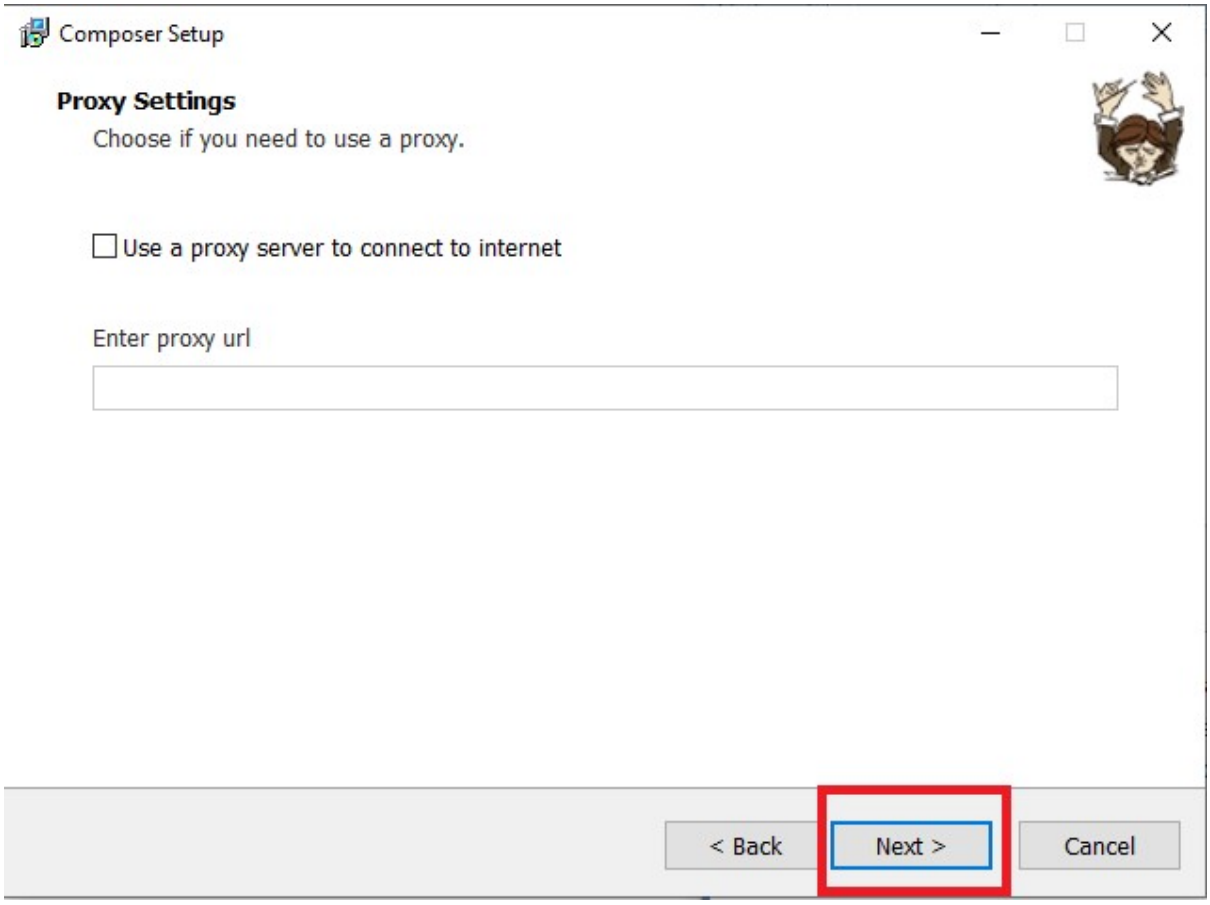
e) Now, choose your installation type and click on the **Next**



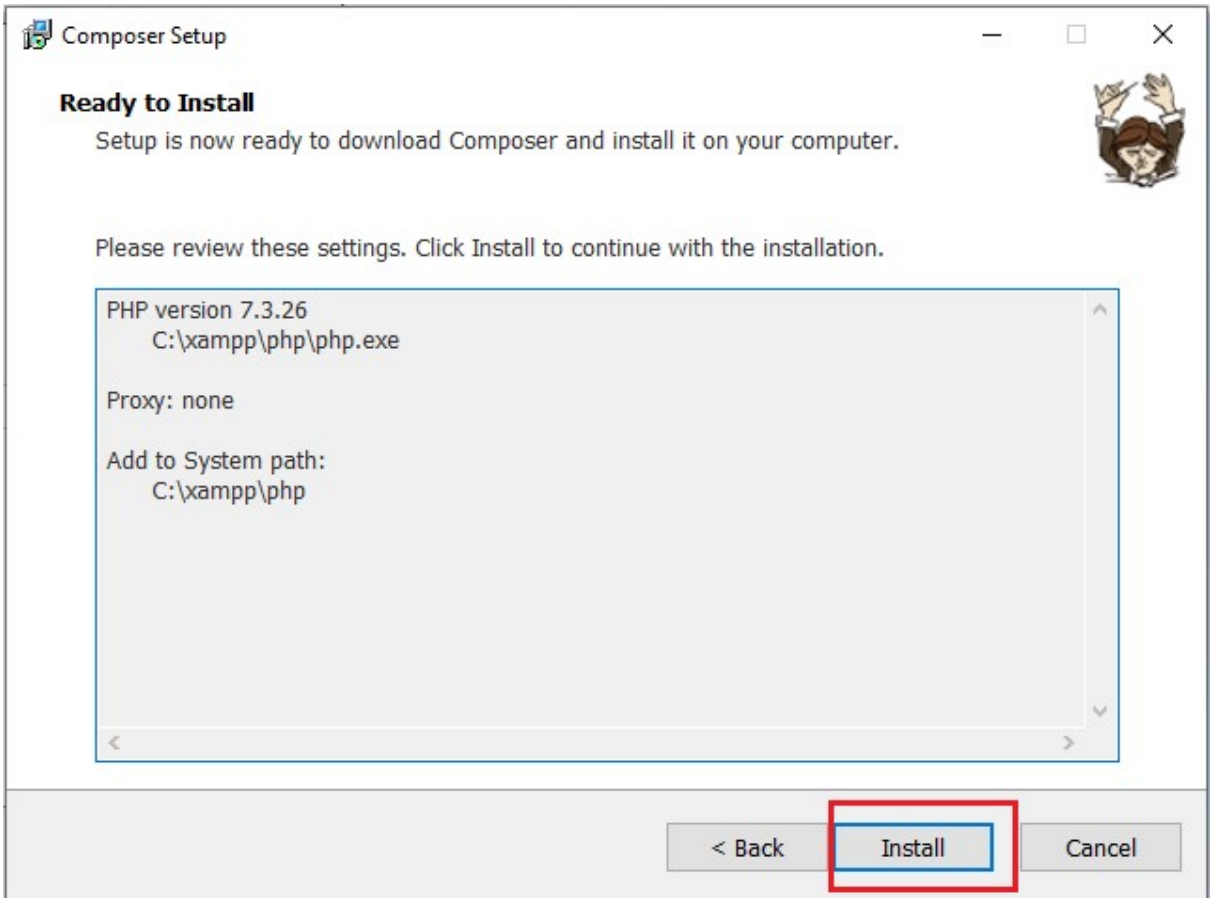
f) Now, choose the command-line PHP path you want to use, checkmark the box to add the PHP path, and click **Next**.



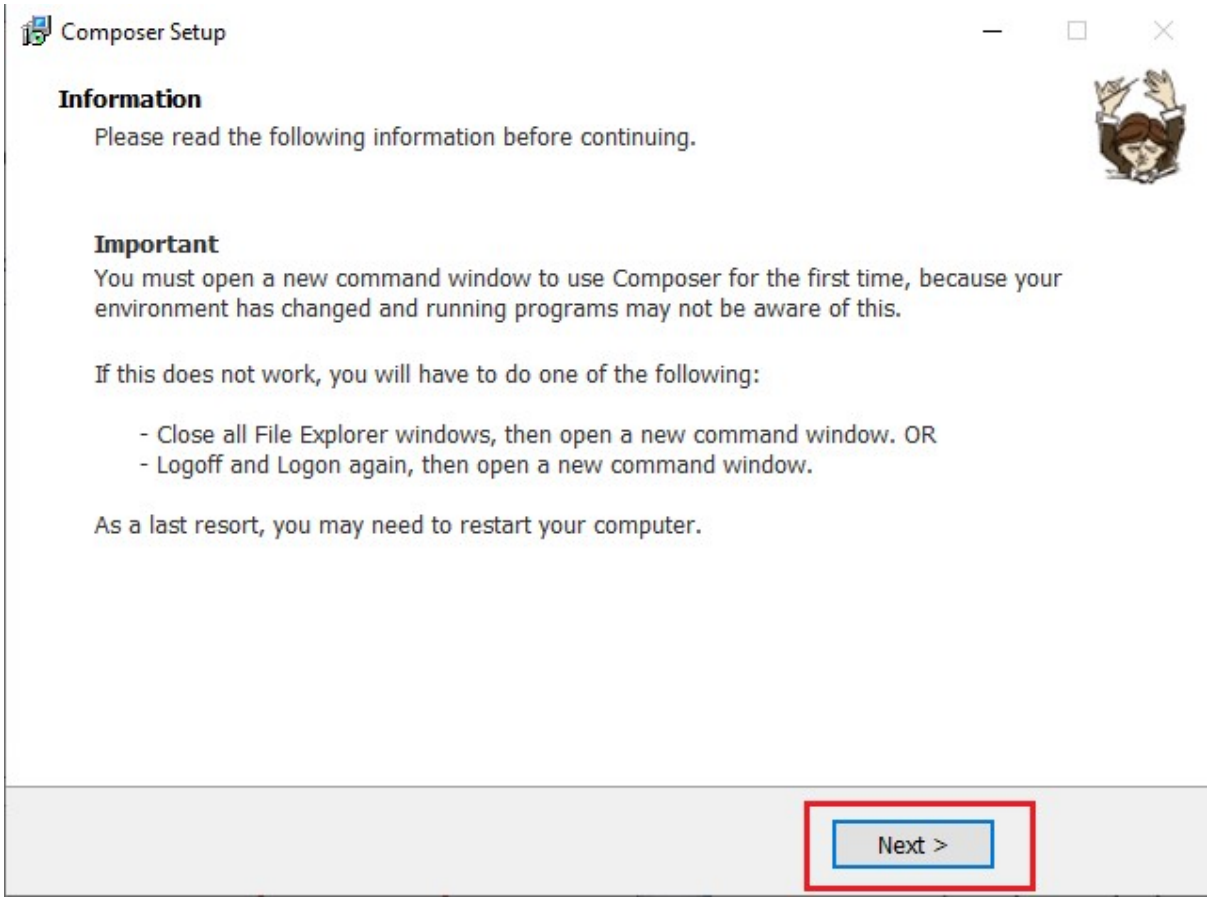
g) Composer setup pop-up a screen that provides an option to use a proxy server to connect to the internet. If you want to use a proxy server, checkmark the box and enter the proxy URL; if not, leave it and click on the **Next** We are skipping this as we are not using any proxy server to connect internet.



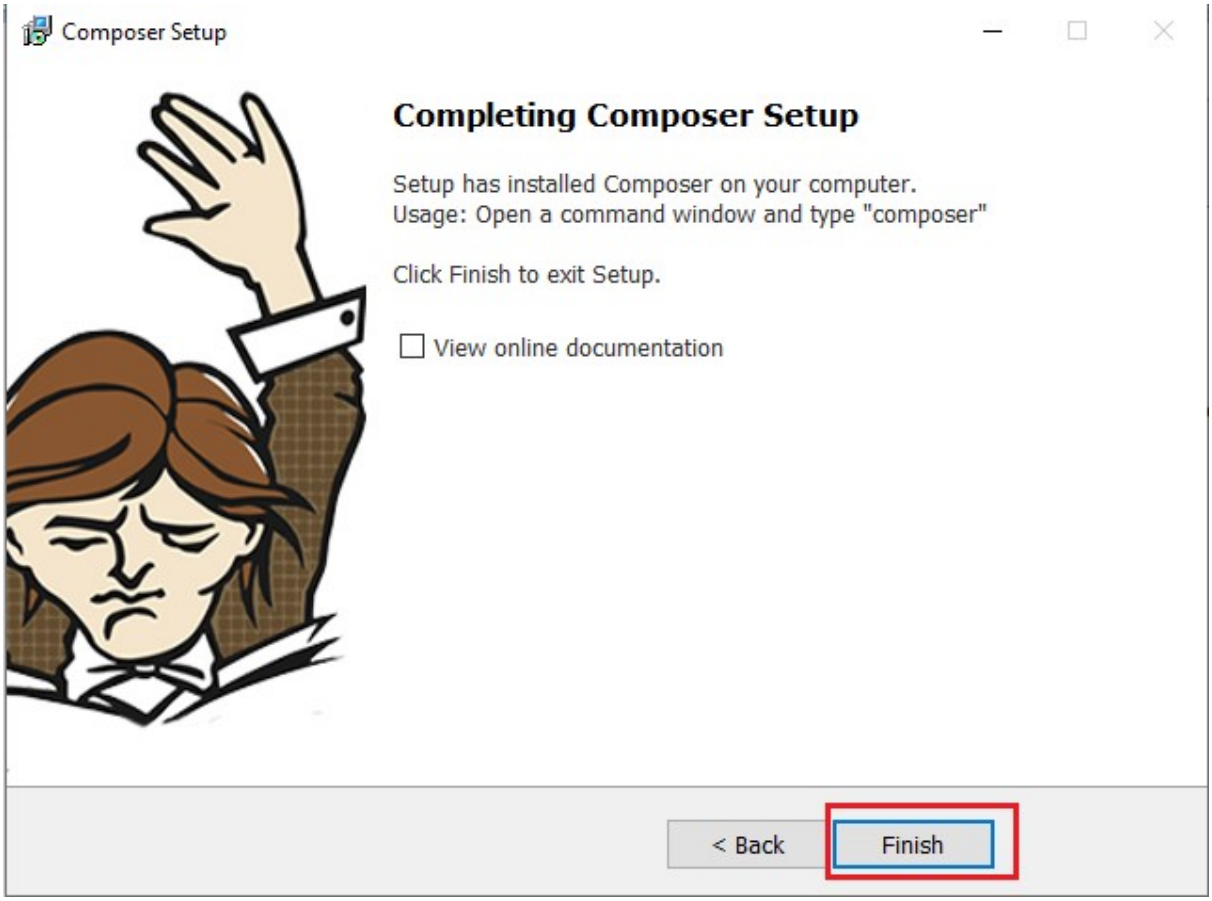
h) The Composer setup is ready to install on your computer; review your settings and click on the Install button.



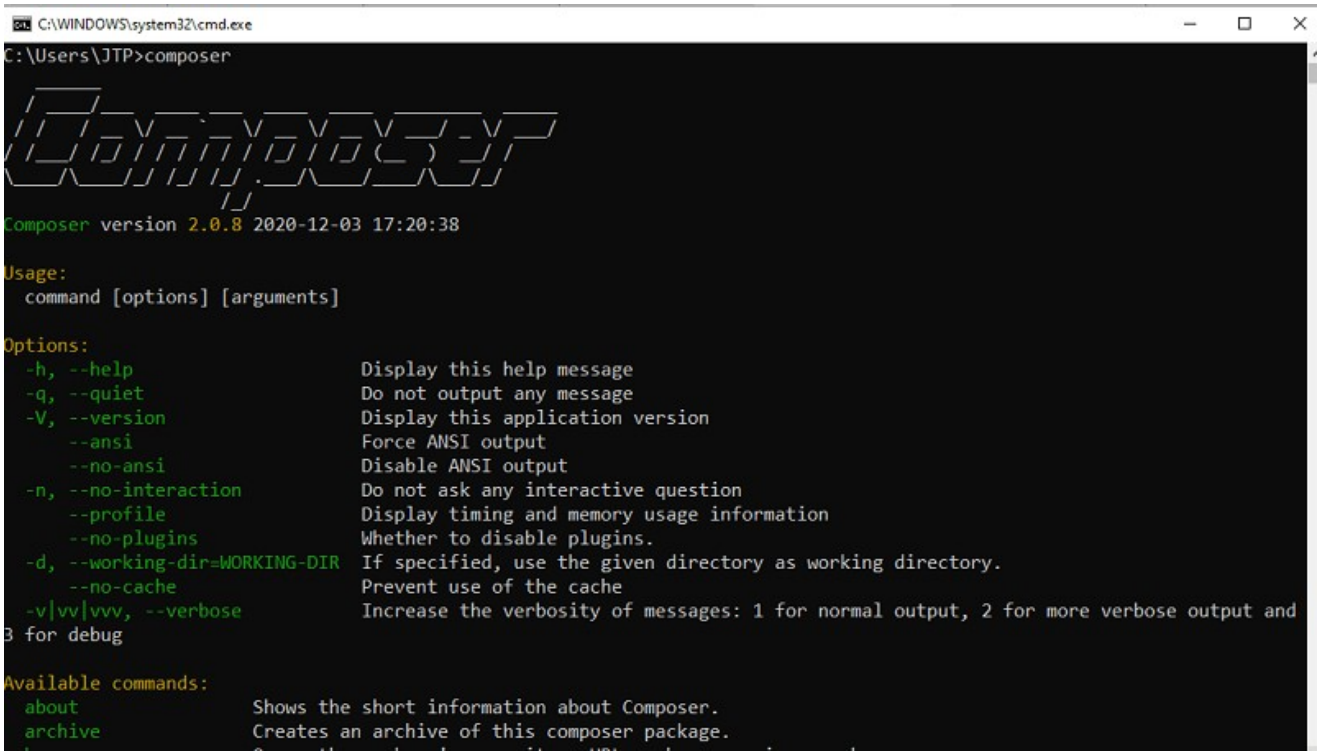
i) After installation of Composer setup, it pop-up important information about how to open it. Read the information, click on Next and do accordingly after installation.



j) Click on the Finish button to complete the installation.



When the Composer gets installed on your machine, open command (cmd) windows, type ***composer*** and press **Enter** key. If it displays a list of commands, that means Composer is successfully installed on your computer.



PHPMyAdmin Login

Introduction

phpMyAdmin is the free software tool specified in ***PHP***, designed for handling MySQL administration on the web. This software tool supports a huge operation's range on ***MariaDB*** and ***MySQL***. Mostly used operations are permissions, users, indexes, relations, columns, tables, managing databases, etc. It can be implemented by a user interface. Still, we have the capability to run the SQL statement directly.

phpMyAdmin uses a huge documentation range and users will be able for updating wiki pages and distribute ideas for several operations. It will support us in case we face any issue. We can apply the support channel's variety to get support.

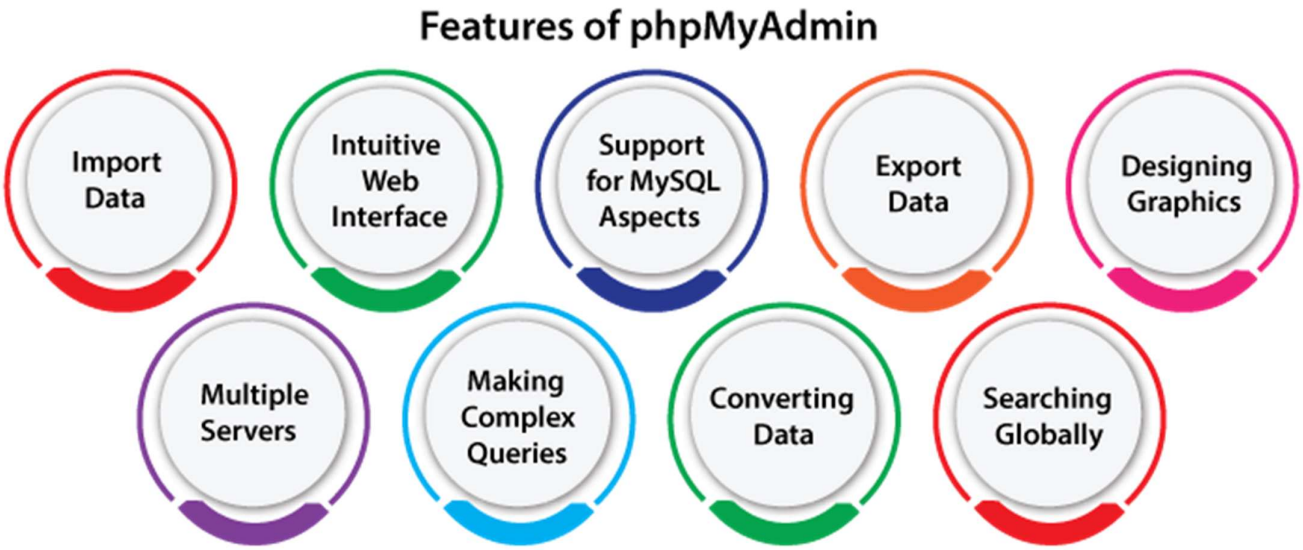
Also, ***phpMyAdmin*** is very well documented inside a book explained by the developers- ***Mastering phpMyAdmin for Effective MySQL Management***, available in ***Spanish*** and ***English***.

Key points:

- ***phpMyAdmin*** supports both RTL and LTR languages and converted into ***72 languages***.

- ***phpMyAdmin*** is one of the most widely used mature projects along with a flexible and stable code base. We can find more about its history, project, and awards it collected.
- This project is the Software Freedom Conservancy member. The Software Freedom Conservancy is not-for-profit enterprise that supports improve, promote, develop, open-source Software, Libre, and defend free projects.

Features of phpMyAdmin:



- Import data through SQL and CSV
- Intuitive web interface
- Support for MySQL aspects:
    - Drop and browse databases, indexes, fields, views, and tables.
    - Alter, rename, drop, copy, and create databases, indexes, fields, and tables.
    - Maintain tables, databases, and server with proposals on the server
    - Bookmark, edit, and execute batch queries and SQL statement.
    - Manage MySQL privileges and user accounts.
    - Manage stored triggers and procedures.
- Export data for several formats: **PDF, XML, SQL, JEC/JSO, CSV**.
- Designing graphics of our database layout within several formats.
- Multiple servers.

- Making complex queries with QBE (Query-by-example).
- Converting stored data into a format with predefined functions set, such as showing BLOB- data as download-link or image.
- Searching globally within the database or any subset of it.

phpMyAdmin Connection

phpMyAdmin can be accessible when using the hostname 127.0.0.1 for various security reasons. We should make the SSH tunnel that directs the requests to any Web-server through 127.0.0.1 for accessing it through the remote system. It indicates that we should be able for connecting to our server on SSH to access the applications remotely.

> *Note: Make sure that our database server and Web are running before going to proceed with the following steps.*

For accessing phpMyAdmin by the SSH tunnel, we need the SSH client. Inside the instructions below we've selected PuTTY. PuTTY is the SSH client (free) for LINUX and Windows platforms. The initial step is the PuTTY configuration.

Once we've our SSH client configured correctly and we've confirmed that we can access our instance with SSH successfully, we need to create the SSH tunnel for accessing phpMyAdmin. Consider the following steps:

- Within the section "Connection -> SSH -> Tunnels", include the new port via introducing the below values:
  - Source Port: 8888
  - Destination: localhost:80

> *Note: If we are redirecting requests of HTTP to an HTTP port, we must consider port 443 rather than 80.*

- Access phpMyAdmin console from the protected tunnel we created, via browsing to hhtp://127.0.0.1:8888/phpmyadmin.
- Now, log-in to the phpMyAdmin via using the below information:
  - Username: root

- Password: application password

PHP MySQL Database

What is MySQL?

- 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
- MySQL is named after co-founder Monty Widenius's daughter: My

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.

Databases are useful for storing information categorically. A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

PHP Connect to MySQL

Since PHP 5.5, **mysql_connect()** extension is *deprecated*.

PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

Should I Use MySQLi or PDO?

If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

MySQL Examples in Both MySQLi and PDO Syntax

In this, and in the following chapters we demonstrate three ways of working with PHP and MySQL:

- MySQLi (object-oriented)
- MySQLi (procedural)
- PDO

MySQLi Installation

- For Linux and Windows: The MySQLi extension is automatically installed in most cases, when php5 mysql package is installed.

Open a Connection to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the server:

Example (MySQLi Object-Oriented)

```php
<?php
$servername = "localhost";  OR $servername=”localhost:3306”;
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

Example (PDO)

```php
<?php
$servername = "localhost";
```

```
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username,
$password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo "Connected successfully";
} catch(PDOException $e) {
  echo "Connection failed: " . $e->getMessage();
}
?>
```

Note: In the PDO example above we have also **specified a database (myDB)**. PDO require a valid database to connect to. If no database is specified, an exception is thrown.

Tip: A great benefit of PDO is that it has an exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

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;
```

PHP Create a MySQL Database

A database consists of one or more tables.

You will need special CREATE privileges to create or to delete a MySQL database.

Create a MySQL Database Using MySQLi and PDO

The CREATE DATABASE statement is used to create a database in MySQL.

The following examples create a database named "myDB":

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . $conn->error;
}

$conn->close();
?>
```

Note: When you create a new database, you must only specify the first three arguments to the mysqli object (servername, username and password).

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Note: The following PDO example create a database named "myDBPDO":

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```
try {
  $conn = new PDO("mysql:host=$servername", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "CREATE DATABASE myDBPDO";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Database created successfully<br>";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

Tip: A great benefit of PDO is that it has exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block. In the catch block above we echo the SQL statement and the generated error message.

PHP MySQL Create Table

A database table has its own unique name and consists of columns and rows.

Create a MySQL Table Using MySQLi and PDO

The CREATE TABLE statement is used to create a table in MySQL.

We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
```

```
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)
```

Notes on the table above:

The data type specifies what type of data the column can hold.

After the data type, you can specify other optional attributes for each column:

- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

The following examples shows how to create the table in PHP:

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
```

```
  die("Connection failed: " . $conn->connect_error);
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
  echo "Table MyGuests created successfully";
} else {
  echo "Error creating table: " . $conn->error;
}

$conn->close();
?>
```

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
```

```
  die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
  echo "Table MyGuests created successfully";
} else {
  echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // set the PDO error mode to exception
```

```
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // sql to create table
  $sql = "CREATE TABLE MyGuests (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstname VARCHAR(30) NOT NULL,
  lastname VARCHAR(30) NOT NULL,
  email VARCHAR(50),
  reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
  )";

  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Table MyGuests created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

DATA TYPES

In MySQL there are three main data types: string, numeric, and date and time.

String Data Types

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column

	length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes

BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them

SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Numeric Data Types

Data type	**Description**
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to

	65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated

	in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Note: All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

Date and Time Data Types

Data type	**Description**
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition

TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

PHP MySQL Insert Data

Insert Data Into MySQL Using MySQLi and PDO

After a database and a table have been created, we can start adding data in them.

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

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,...)

In the previous chapter we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.

Note: If a column is AUTO_INCREMENT (like the "id" column) or TIMESTAMP with default update of current_timesamp (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

The following examples add a new record to the "MyGuests" table:

Example (MySQLi Object-oriented)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
```

```php
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
```

```
  VALUES ('John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo "New record created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Get Last Inserted ID

Get ID of The Last Inserted Record

If we perform an INSERT or UPDATE on a table with an AUTO_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

In the table "MyGuests", the "id" column is an AUTO_INCREMENT field:

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)
```

The following examples are equal to the examples from the previous page (PHP Insert Data Into MySQL), except that we have added one single line of code to retrieve the ID of the last inserted record. We also echo the last inserted ID:

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
```

```php
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
  $last_id = $conn->insert_id;
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
```

```php
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
  $last_id = mysqli_insert_id($conn);
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  $last_id = $conn->lastInsertId();
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Insert Multiple Records

Insert Multiple Records Into MySQL Using MySQLi and PDO

Multiple SQL statements must be executed with the mysqli_multi_query() function.

The following examples add three new records to the "MyGuests" table:

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if ($conn->multi_query($sql) === TRUE) {
  echo "New records created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
```

```
$conn->close();
?>
```

Note that each SQL statement must be separated by a semicolon.

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if (mysqli_multi_query($conn, $sql)) {
  echo "New records created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

The PDO way is a little bit different:

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  // begin the transaction
  $conn->beginTransaction();
  // our SQL statements
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')");
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Mary', 'Moe', 'mary@example.com')");
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Julie', 'Dooley', 'julie@example.com')");

  // commit the transaction
  $conn->commit();
  echo "New records created successfully";
} catch(PDOException $e) {
  // roll back the transaction if something failed
  $conn->rollback();
  echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

INSERT RECORDS:

```
<!—Insert Record using MySQLi Procedural -->

<?php

/* Attempt MySQL server connection. Assuming you are running MySQL

server with default setting (user 'root' with no password) */

require "connection/config.php";

$conn = mysqli_connect($host, $username, $password, $db);

// Check connection

if ($conn === false) {

        die("ERROR: Could not connect. " . mysqli_connect_error());

}

// Attempt insert query execution

$sql = "INSERT INTO EMP (firstname, lastname, email) VALUES ('Allan', 'Parker',
'allanparker@gmail.com')";

if (mysqli_query($conn, $sql)) {

        $last_id = $conn->insert_id;

        echo "<h1>New record inserted successfully. Last inserted ID is: " . $last_id .
"</h1>";

        // echo "<h1>Records inserted successfully.</h1>";

} else {

        echo "ERROR: Could not able to execute $sql. " . mysqli_error($conn);

}

// Close connection

mysqli_close($conn);

?>
```

```
<!-- PHP code to insert records using MySQLi Object Oriented -->
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
require "connection/config.php";

$conn = new mysqli($host, $username, $password, $db);

// Check connection
if ($conn === false) {
	die("ERROR: Could not connect. " . $conn->connect_error);
}

// Attempt insert query execution
$sql = "INSERT INTO EMP (firstname, lastname, email) VALUES ('Smith', 'James',
'james@gmail.com')";
if ($conn->query($sql) === true) {
	echo "<h1>Record inserted successfully.</h1>";
} else {
	echo "ERROR: Could not able to execute $sql. " . $conn->error;
}

// Close connection
$conn->close();
?>
```

```
<!-- Insert record using PDO -->
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
require "connection/config.php";
try {
	$conn = new PDO("mysql:host=localhost;dbname=myDB3", "root", "admin");
	// Set the PDO error mode to exception
	$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
	die("ERROR: Could not connect. " . $e->getMessage());
}

// Attempt insert query execution
try {
	$sql = "INSERT INTO EMP (firstname, lastname, email) VALUES ('Mark',
'Taylor', 'mark@gmail.com')";
	$conn->exec($sql);
	echo "Records inserted successfully.";
} catch (PDOException $e) {
	die("ERROR: Could not able to execute $sql. " . $e->getMessage());
}

// Close connection
unset($conn);
?>
```

INSERT BULK RECORDS:

```
<!-- MySQLi Procedural -->
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
require "connection/config.php";

$conn = mysqli_connect($host, $username, $password, $db);
// Check connection
if ($conn === false) {
	die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt insert query execution
$sql = "INSERT INTO EMP (firstname, lastname, email) VALUES
		('Ricky', 'Ponting', 'ricky@gmail.com'),
		('Pat', 'Cummings', 'pat@gmail.com'),
		('Cathy', 'Smith', 'cathy@gmail.com'),
		('Sara', 'Huges', 'sara@gmail.com')";
if (mysqli_query($conn, $sql)) {
	echo "<h1>Records inserted successfully.</h1>";
} else {
	echo "ERROR: Could not able to execute $sql. " . mysqli_error($conn);
}

// Close connection
mysqli_close($conn);
?>
```

https://www.tutorialrepublic.com/php-tutorial/php-mysql-prepared-statements.php

PHP MySQL Prepared Statements

Prepared statements are very useful against SQL injections.

Prepared Statements and Bound Parameters

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Compared to executing SQL statements directly, prepared statements have three main advantages:

- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

Prepared Statements in MySQLi

The following example uses prepared statements and bound parameters in MySQLi:

Example (MySQLi with Prepared Statements)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();
```

```
echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

Code lines to explain from the example above:

"INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)"

In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double or blob value.

Then, have a look at the bind_param() function:

$stmt->bind_param("sss", $firstname, $lastname, $email);

This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.

The argument may be one of four types:

- i - integer
- d - double
- s - string
- b - BLOB

We must have one of these for each parameter.

By telling mysql what type of data to expect, we minimize the risk of SQL injections.

Note: If we want to insert any data from external sources (like user input), it is very important that the data is sanitized and validated.

Prepared Statements in PDO

The following example uses prepared statements and bound parameters in PDO:

Example (PDO with Prepared Statements)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // prepare sql and bind parameters
  $stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES (:firstname, :lastname, :email)");
  $stmt->bindParam(':firstname', $firstname);
  $stmt->bindParam(':lastname', $lastname);
  $stmt->bindParam(':email', $email);

  // insert a row
  $firstname = "John";
  $lastname = "Doe";
  $email = "john@example.com";
  $stmt->execute();

  // insert another row
  $firstname = "Mary";
  $lastname = "Moe";
  $email = "mary@example.com";
  $stmt->execute();

  // insert another row
  $firstname = "Julie";
```

```
  $lastname = "Dooley";
  $email = "julie@example.com";
  $stmt->execute();

  echo "New records created successfully";
} catch(PDOException $e) {
  echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

https://www.w3schools.com/php/php_mysql_select.asp

Reusable script for connecting the database:

https://www.phptutorial.net/php-pdo/php-pdo-mysql/

PHP Date/Time Introduction

The date/time functions allow you to get the date and time from the server where your PHP script runs. You can then use the date/time functions to format the date and time in several ways.

Note: These functions depend on the locale settings of your server. Remember to take daylight saving time and leap years into consideration when working with these functions.

PHP Date/Time Functions

Function	Description
checkdate()	Validates a Gregorian date
date_add()	Adds days, months, years, hours, minutes, and seconds to a date
date_create_from_format()	Returns a new DateTime object formatted according to a specified format
date_create()	Returns a new DateTime object
date_date_set()	Sets a new date
date_default_timezone_get()	Returns the default timezone used by all date/time functions
date_default_timezone_set()	Sets the default timezone used by all date/time functions
date_diff()	Returns the difference between two dates

date_format()	Returns a date formatted according to a specified format
date_get_last_errors()	Returns the warnings/errors found in a date string
date_interval_create_from_date_string()	Sets up a DateInterval from the relative parts of the string
date_interval_format()	Formats the interval
date_isodate_set()	Sets the ISO date
date_modify()	Modifies the timestamp
date_offset_get()	Returns the timezone offset
date_parse_from_format()	Returns an associative array with detailed info about a specified date, according to a specified format
date_parse()	Returns an associative array with detailed info about a specified date
date_sub()	Subtracts days, months, years, hours, minutes, and seconds from a date
date_sun_info()	Returns an array containing info about sunset/sunrise and twilight begin/end, for a specified day and location
date_sunrise()	Returns the sunrise time for a specified day and location

date_sunset()	Returns the sunset time for a specified day and location
date_time_set()	Sets the time
date_timestamp_get()	Returns the Unix timestamp
date_timestamp_set()	Sets the date and time based on a Unix timestamp
date_timezone_get()	Returns the time zone of the given DateTime object
date_timezone_set()	Sets the time zone for the DateTime object
date()	Formats a local date and time
getdate()	Returns date/time information of a timestamp or the current local date/time
gettimeofday()	Returns the current time
gmdate()	Formats a GMT/UTC date and time
gmmktime()	Returns the Unix timestamp for a GMT date
gmstrftime()	Formats a GMT/UTC date and time according to locale settings
idate()	Formats a local time/date as integer
localtime()	Returns the local time

microtime()	Returns the current Unix timestamp with microseconds
mktime()	Returns the Unix timestamp for a date
strftime()	Formats a local time and/or date according to locale settings
strptime()	Parses a time/date generated with strftime()
strtotime()	Parses an English textual datetime into a Unix timestamp
time()	Returns the current time as a Unix timestamp
timezone_abbreviations_list()	Returns an associative array containing dst, offset, and the timezone name
timezone_identifiers_list()	Returns an indexed array with all timezone identifiers
timezone_location_get()	Returns location information for a specified timezone
timezone_name_from_abbr()	Returns the timezone name from abbreviation
timezone_name_get()	Returns the name of the timezone
timezone_offset_get()	Returns the timezone offset from GMT

timezone_open()	Creates new DateTimeZone object
timezone_transitions_get()	Returns all transitions for the timezone
timezone_version_get()	Returns the version of the timezonedb

PHP Date and Time

The PHP date() function is used to format a date and/or a time.

The PHP Date() Function

The PHP date() function formats a timestamp to a more readable date and time.

Syntax

date(format,timestamp)

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

Get a Date

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)

- l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

The example below formats today's date in three different ways:

```
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```

PHP Tip - Automatic Copyright Year

Use the date() function to automatically update the copyright year on your website:

Example

© 2010-<?php echo date("Y");?>

Get a Time

Here are some characters that are commonly used for times:

- H - 24-hour format of an hour (00 to 23)
- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

Example

```
<?php
echo "The time is " . date("h:i:sa");
?>
```

Note that the PHP date() function will return the current date/time of the server!

Get Your Time Zone

If the time you got back from the code is not correct, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set the timezone you want to use.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

Example

```
<?php
date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>
```

```
<!DOCTYPE html>
<html>
<body>


<?php
date_default_timezone_set("Asia/Calcutta");
echo "The time is " . date("H:i:s");   // H: is for 24 Hour
echo "The time is " . date("h:i:sa");  // h: is for 12 Hour
?>
</body>
</html>


<?php
$datetime = new DateTime( "now", new DateTimeZone( "Africa/Nairobi" ) );
```

```
echo $datetime->format( 'Y-m-d H:i:s' );

echo $datetime->format( 'm-d-Y H:i:s' );

?>
```

Create a Date With mktime()

The optional *timestamp* parameter in the date() function specifies a timestamp. If omitted, the current date and time will be used (as in the examples above).

The PHP mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Syntax

```
mktime(hour, minute, second, month, day, year)
```

The example below creates a date and time with the date() function from a number of parameters in the mktime() function:

Example

```
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

Create a Date From a String With strtotime()

The PHP strtotime() function is used to convert a human readable date string into a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).

Syntax

strtotime(*time, now*)

The example below creates a date and time from the strtotime() function:

Example

```
<?php
$d=strtotime("10:30pm April 15 2014");
```

```
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

PHP is quite clever about converting a string to a date, so you can put in various values:

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";
$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";
$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

However, strtotime() is not perfect, so remember to check the strings you put in there.

The example below outputs the dates for the next six Saturdays:

```
<?php
$startdate = strtotime("Saturday");
$enddate = strtotime("+6 weeks", $startdate);

while ($startdate < $enddate) {
  echo date("M d", $startdate) . "<br>";
  $startdate = strtotime("+1 week", $startdate);
}
?>
```

Getting the Time Stamp with time()

PHP's **time()** function gives you all the information that you need about the current date and time. It requires no arguments but returns an integer.

The integer returned by time() represents the number of seconds elapsed since midnight GMT on January 1, 1970. This moment is known as the UNIX epoch, and the number of seconds that have elapsed since then is referred to as a time stamp.

```
<?php
  print time();
```

```
?>
```

This will produce the following result –

```
1480930103
```

Converting a Time Stamp with getdate()

The function **getdate()** optionally accepts a time stamp and returns an associative array containing information about the date. If you omit the time stamp, it works with the current time stamp as returned by time().

Following table lists the elements contained in the array returned by getdate().

Sr.No	Key & Description	Example
1	seconds Seconds past the minutes (0-59)	20
2	minutes Minutes past the hour (0 - 59)	29
3	hours Hours of the day (0 - 23)	22
4	mday Day of the month (1 - 31)	11
5	wday Day of the week (0 - 6)	4
6	mon Month of the year (1 - 12)	7
7	year Year (4 digits)	1997
8	yday	19

	Day of year (0 - 365)	
9	weekday Day of the week	Thursday
10	month Month of the year	January
11	0 Timestamp	948370048

Now you have complete control over date and time. You can format this date and time in whatever format you want.

```
<?php
  $date_array = getdate();

  foreach ( $date_array as $key => $val ){
    print "$key = $val<br />";
  }

  $formated_date  = "Today's date: ";
  $formated_date .= $date_array['mday'] . "/";
  $formated_date .= $date_array['mon'] . "/";
  $formated_date .= $date_array['year'];

  print $formated_date;
?>
```

Output:

seconds = 7

minutes = 14

hours = 8

mday = 24

wday = 5

mon = 12

year = 2021

yday = 357

weekday = Friday

month = December

0 = 1640333647

Today's date: 24/12/2021

Converting a Time Stamp with date()

The **date()** function returns a formatted string representing a date. You can exercise an enormous amount of control over the format that date() returns with a string argument that you must pass to it.

date(format,timestamp)

The date() optionally accepts a time stamp if omitted then current date and time will be used. Any other data you include in the format string passed to date() will be included in the return value.

Following table lists the codes that a format string can contain −

Sr.No	Format & Description	Example
1	a 'am' or 'pm' lowercase	pm
2	A 'AM' or 'PM' uppercase	PM
3	d Day of month, a number with leading zeroes	20
4	D Day of week (three letters)	Thu
5	F Month name	January
6	h Hour (12-hour format - leading zeroes)	12

7	H Hour (24-hour format - leading zeroes)	22
8	g Hour (12-hour format - no leading zeroes)	12
9	G Hour (24-hour format - no leading zeroes)	22
10	i Minutes (0 - 59)	23
11	j Day of the month (no leading zeroes	20
12	l (Lower 'L') Day of the week	Thursday
13	L Leap year ('1' for yes, '0' for no)	1
14	m Month of year (number - leading zeroes)	1
15	M Month of year (three letters)	Jan
16	r The RFC 2822 formatted date	Thu, 21 Dec 2000 16:01:07 +0200
17	n Month of year (number - no leading zeroes)	2
18	s Seconds of hour	20
19	U Time stamp	948372444
20	y Year (two digits)	06
21	Y Year (four digits)	2006
22	z Day of year (0 - 365)	206

23	Z Offset in seconds from GMT	+5

```
<?php
   print date("m/d/y G.i:s<br>", time());
   echo "<br>";
   print "Today is ";
   print date("j of F Y, \a\\t g.i a", time());
?>
```

This will produce following result −

12/05/16 9:29:47

Today is 5 2016f December 2016 at 9:29 am

PHP checkdate() Function

The checkdate() function is used to validate a Gregorian date.

Syntax

checkdate(*month, day, year)*

Parameter Values

Parameter	Description
month	Required. Specifies the month as a number between 1 and 12
day	Required. Specifies the day as a number between 1 and 31
year	Required. Specifies the year as a number between 1 and 32767

Return Value: TRUE if the date is valid. FALSE otherwise

Check if several dates are valid Gregorian dates:

```
<?php
var_dump(checkdate(12,31,-400));
echo "<br>";
var_dump(checkdate(2,29,2003));
echo "<br>";
var_dump(checkdate(2,29,2004));
?>
```

PHP date_add() Function:

The date_add() function adds some days, months, years, hours, minutes, and seconds to a date.

Syntax

date_add(*object, interval)*

Parameter Values

Parameter	Description
object	Required. Specifies a DateTime object returned by date_create()
interval	Required. Specifies a DateInterval object

```
<?php
$date=date_create("2013-03-15");
date_add($date,date_interval_create_from_date_string("40 days"));
echo date_format($date,"Y-m-d");
?>
```

PHP date_create_from_format() Function

The date_create_from_format() function returns a new DateTime object formatted according to the specified format.

Syntax

date_create_from_format(*format, time, timezone)*

Parameter Values

Parameter	**Description**
format	Required. Specifies the format to use. The following characters can be used in the *format* parameter string: • d - Day of the month; with leading zeros • j - Day of the month; without leading zeros • D - Day of the month (Mon - Sun) • l - Day of the month (Monday - Sunday) • S - English suffix for day of the month (st, nd, rd, th) • F - Monthname (January - December) • M - Monthname (Jan-Dec) • m - Month (01-12) • n - Month (1-12) • Y - Year (e.g 2013) • y - Year (e.g 13) • a and A - am or pm • g - 12 hour format without leading zeros • G - 24 hour format without leading zeros • h - 12 hour format with leading zeros • H - 24 hour format with leading zeros • i - Minutes with leading zeros • s - Seconds with leading zeros • u - Microseconds (up to six digits) • e, O, P and T - Timezone identifier • U - Seconds since Unix Epoch • (space) • # - One of the following separation symbol: ;,:,/,.,,,-,(,) • ? - A random byte • * - Rondom bytes until next separator/digit • ! - Resets all fields to Unix Epoch • | - Resets all fields to Unix Epoch if they have not been parsed yet • + - If present, trailing data in the string will cause a warning, not an error
time	Required. Specifies a date/time string. NULL indicates the current date/time

timezone	Optional. Specifies the timezone of *time*. Default is the current timezone

```
<!DOCTYPE html>
<html>
<body>
<?php
$date=date_create_from_format("j-M-Y","15-Mar-2013");
echo date_format($date,"Y/m/d");
?>
</body>
</html>
```

PHP date_create() Function

The date_create() function returns a new DateTime object.

Syntax

date_create(*time, timezone)*

Parameter Values

Parameter	Description
time	Optional. Specifies a date/time string. NULL indicates the current time
timezone	Optional. Specifies the timezone of time. Default is the current timezone.

```
<!DOCTYPE html>
<html>
```

```
<body>

<?php

$date=date_create(date());

echo date_format($date,"Y/m/d");

?>

</body>

</html>
```

PHP Supported Timezones

Below is a complete list of the timezones supported by PHP, which are useful with several PHP date functions.

- Africa
- America
- Antarctica
- Arctic
- Asia
- Atlantic
- Australia
- Europe
- Indian
- Pacific

Africa

Africa/Abidjan	Africa/Accra	Africa/Addis_Ababa	Africa/Algiers	Africa/Asmara
Africa/Asmera	Africa/Bamako	Africa/Bangui	Africa/Banjul	Africa/Bissau
Africa/Blantyre	Africa/Brazzaville	Africa/Bujumbura	Africa/Cairo	Africa/Casablanca
Africa/Ceuta	Africa/Conakry	Africa/Dakar	Africa/Dar_es_Salaam	Africa/Djibouti

Africa/Douala	Africa/El_Aaiun	Africa/Freetown	Africa/Gaborone	Africa/Harare
Africa/Johannesburg	Africa/Juba	Africa/Kampala	Africa/Khartoum	Africa/Kigali
Africa/Kinshasa	Africa/Lagos	Africa/Libreville	Africa/Lome	Africa/Luanda
Africa/Lubumbashi	Africa/Lusaka	Africa/Malabo	Africa/Maputo	Africa/Maseru
Africa/Mbabane	Africa/Mogadishu	Africa/Monrovia	Africa/Nairobi	Africa/Ndjamena
Africa/Niamey	Africa/Nouakchott	Africa/Ouagadougou	Africa/Porto-Novo	Africa/Sao_Tome
Africa/Timbuktu	Africa/Tripoli	Africa/Tunis	Africa/Windhoek	

America

America/Adak	America/Anchorage	America/Anguilla
America/Antigua	America/Araguaina	America/Argentina/Buenos_Aires
America/Argentina/Catamarca	America/Argentina/ComodRivadavia	America/Argentina/Cordoba
America/Argentina/Jujuy	America/Argentina/La_Rioja	America/Argentina/Mendoza
America/Argentina/Rio_Gallegos	America/Argentina/Salta	America/Argentina/San_Juan
America/Argentina/San_Luis	America/Argentina/Tucuman	America/Argentina/Ushuaia
America/Aruba	America/Asuncion	America/Atikokan
America/Atka	America/Bahia	America/Bahia_Banderas
America/Barbados	America/Belem	America/Belize
America/Blanc-Sablon	America/Boa_Vista	America/Bogota

America/Boise	America/Buenos_Aires	America/Cambridge_Bay
America/Campo_Grande	America/Cancun	America/Caracas
America/Catamarca	America/Cayenne	America/Cayman
America/Chicago	America/Chihuahua	America/Coral_Harbour
America/Cordoba	America/Costa_Rica	America/Creston
America/Cuiaba	America/Curacao	America/Danmarkshavn
America/Dawson	America/Dawson_Creek	America/Denver
America/Detroit	America/Dominica	America/Edmonton
America/Eirunepe	America/El_Salvador	America/Ensenada
America/Fort_Wayne	America/Fortaleza	America/Glace_Bay
America/Godthab	America/Goose_Bay	America/Grand_Turk
America/Grenada	America/Guadeloupe	America/Guatemala
America/Guayaquil	America/Guyana	America/Halifax
America/Havana	America/Hermosillo	America/Indiana/Indianapolis
America/Indiana/Knox	America/Indiana/Marengo	America/Indiana/Petersburg
America/Indiana/Tell_City	America/Indiana/Vevay	America/Indiana/Vincennes
America/Indiana/Winamac	America/Indianapolis	America/Inuvik
America/Iqaluit	America/Jamaica	America/Jujuy
America/Juneau	America/Kentucky/Louisville	America/Kentucky/Monticello
America/Knox_IN	America/Kralendijk	America/La_Paz
America/Lima	America/Los_Angeles	America/Louisville
America/Lower_Princes	America/Maceio	America/Managua

America/Manaus	America/Marigot	America/Martinique
America/Matamoros	America/Mazatlan	America/Mendoza
America/Menominee	America/Merida	America/Metlakatla
America/Mexico_City	America/Miquelon	America/Moncton
America/Monterrey	America/Montevideo	America/Montreal
America/Montserrat	America/Nassau	America/New_York
America/Nipigon	America/Nome	America/Noronha
America/North_Dakota/Beulah	America/North_Dakota/Center	America/North_Dakota/New_Salem
America/Ojinaga	America/Panama	America/Pangnirtung
America/Paramaribo	America/Phoenix	America/Port-au-Prince
America/Port_of_Spain	America/Porto_Acre	America/Porto_Velho
America/Puerto_Rico	America/Rainy_River	America/Rankin_Inlet
America/Recife	America/Regina	America/Resolute
America/Rio_Branco	America/Rosario	America/Santa_Isabel
America/Santarem	America/Santiago	America/Santo_Domingo
America/Sao_Paulo	America/Scoresbysund	America/Shiprock
America/Sitka	America/St_Barthelemy	America/St_Johns
America/St_Kitts	America/St_Lucia	America/St_Thomas
America/St_Vincent	America/Swift_Current	America/Tegucigalpa
America/Thule	America/Thunder_Bay	America/Tijuana
America/Toronto	America/Tortola	America/Vancouver

America/Virgin	America/Whitehorse	America/Winnipeg
America/Yakutat	America/Yellowknife	

Antarctica

Antarctica/Casey	Antarctica/Davis	Antarctica/DumontDUrville	Antarctica/Macquarie	Antarctica/Mawson
Antarctica/McMurdo	Antarctica/Palmer	Antarctica/Rothera	Antarctica/South_Pole	Antarctica/Syowa
Antarctica/Vostok				

Arctic

Arctic/Longyearbyen

Asia

Asia/Aden	Asia/Almaty	Asia/Amman	Asia/Anadyr	Asia/Aqtau
Asia/Aqtobe	Asia/Ashgabat	Asia/Ashkhabad	Asia/Baghdad	Asia/Bahrain
Asia/Baku	Asia/Bangkok	Asia/Beirut	Asia/Bishkek	Asia/Brunei
Asia/Calcutta	Asia/Choibalsan	Asia/Chongqing	Asia/Chungking	Asia/Colombo
Asia/Dacca	Asia/Damascus	Asia/Dhaka	Asia/Dili	Asia/Dubai
Asia/Dushanbe	Asia/Gaza	Asia/Harbin	Asia/Hebron	Asia/Ho_Chi_Minh
Asia/Hong_Kong	Asia/Hovd	Asia/Irkutsk	Asia/Istanbul	Asia/Jakarta
Asia/Jayapura	Asia/Jerusalem	Asia/Kabul	Asia/Kamchatka	Asia/Karachi

Asia/Kashgar	Asia/Kathmandu	Asia/Katmandu	Asia/Khandyga	Asia/Kolkata
Asia/Krasnoyarsk	Asia/Kuala_Lumpur	Asia/Kuching	Asia/Kuwait	Asia/Macao
Asia/Macau	Asia/Magadan	Asia/Makassar	Asia/Manila	Asia/Muscat
Asia/Nicosia	Asia/Novokuznetsk	Asia/Novosibirsk	Asia/Omsk	Asia/Oral
Asia/Phnom_Penh	Asia/Pontianak	Asia/Pyongyang	Asia/Qatar	Asia/Qyzylorda
Asia/Rangoon	Asia/Riyadh	Asia/Saigon	Asia/Sakhalin	Asia/Samarkand
Asia/Seoul	Asia/Shanghai	Asia/Singapore	Asia/Taipei	Asia/Tashkent
Asia/Tbilisi	Asia/Tehran	Asia/Tel_Aviv	Asia/Thimbu	Asia/Thimphu
Asia/Tokyo	Asia/Ujung_Pandang	Asia/Ulaanbaatar	Asia/Ulan_Bator	Asia/Urumqi
Asia/Ust-Nera	Asia/Vientiane	Asia/Vladivostok	Asia/Yakutsk	Asia/Yekaterinburg
Asia/Yerevan				

Atlantic

Atlantic/Azores	Atlantic/Bermuda	Atlantic/Canary	Atlantic/Cape_Verde	Atlantic/Faeroe
Atlantic/Faroe	Atlantic/Jan_Mayen	Atlantic/Madeira	Atlantic/Reykjavik	Atlantic/South_Georgia
Atlantic/St_Helena	Atlantic/Stanley			

Australia

Australia/ACT	Australia/Adelaide	Australia/Brisbane	Australia/Broken_Hill	Australia/Canberra
Australia/Currie	Australia/Darwin	Australia/Eucla	Australia/Hobart	Australia/LHI

Australia/Lindeman	Australia/Lord_Howe	Australia/Melbourne	Australia/North	Australia/NSW
Australia/Perth	Australia/Queensland	Australia/South	Australia/Sydney	Australia/Tasmania
Australia/Victoria	Australia/West	Australia/Yancowinna		

Europe

Europe/Amsterdam	Europe/Andorra	Europe/Athens	Europe/Belfast	Europe/Belgrade
Europe/Berlin	Europe/Bratislava	Europe/Brussels	Europe/Bucharest	Europe/Budapest
Europe/Busingen	Europe/Chisinau	Europe/Copenhagen	Europe/Dublin	Europe/Gibraltar
Europe/Guernsey	Europe/Helsinki	Europe/Isle_of_Man	Europe/Istanbul	Europe/Jersey
Europe/Kaliningrad	Europe/Kiev	Europe/Lisbon	Europe/Ljubljana	Europe/London
Europe/Luxembourg	Europe/Madrid	Europe/Malta	Europe/Mariehamn	Europe/Minsk
Europe/Monaco	Europe/Moscow	Europe/Nicosia	Europe/Oslo	Europe/Paris
Europe/Podgorica	Europe/Prague	Europe/Riga	Europe/Rome	Europe/Samara
Europe/San_Marino	Europe/Sarajevo	Europe/Simferopol	Europe/Skopje	Europe/Sofia
Europe/Stockholm	Europe/Tallinn	Europe/Tirane	Europe/Tiraspol	Europe/Uzhgorod
Europe/Vaduz	Europe/Vatican	Europe/Vienna	Europe/Vilnius	Europe/Volgograd
Europe/Warsaw	Europe/Zagreb	Europe/Zaporozhye	Europe/Zurich	

Indian

Indian/Antananarivo	Indian/Chagos	Indian/Christmas	Indian/Cocos	Indian/Comoro
Indian/Kerguelen	Indian/Mahe	Indian/Maldives	Indian/Mauritius	Indian/Mayotte
Indian/Reunion				

Pacific

Pacific/Apia	Pacific/Auckland	Pacific/Chatham	Pacific/Chuuk	Pacific/Easter
Pacific/Efate	Pacific/Enderbury	Pacific/Fakaofo	Pacific/Fiji	Pacific/Funafuti
Pacific/Galapagos	Pacific/Gambier	Pacific/Guadalcanal	Pacific/Guam	Pacific/Honolulu
Pacific/Johnston	Pacific/Kiritimati	Pacific/Kosrae	Pacific/Kwajalein	Pacific/Majuro
Pacific/Marquesas	Pacific/Midway	Pacific/Nauru	Pacific/Niue	Pacific/Norfolk
Pacific/Noumea	Pacific/Pago_Pago	Pacific/Palau	Pacific/Pitcairn	Pacific/Pohnpei
Pacific/Ponape	Pacific/Port_Moresby	Pacific/Rarotonga	Pacific/Saipan	Pacific/Samoa
Pacific/Tahiti	Pacific/Tarawa	Pacific/Tongatapu	Pacific/Truk	Pacific/Wake
Pacific/Wallis	Pacific/Yap			

PHP date_default_timezone_get() Function

```
<!DOCTYPE html>
<html>
<body>

<?php
echo date_default_timezone_get();
?>

</body>
```

```
</html>
```

PHP date_default_timezone_set() Function

The date_default_timezone_set() function sets the default timezone used by all date/time functions in the script.

Syntax

```
date_default_timezone_set(timezone)
```

```
<!DOCTYPE html>
<html>
<body>

<?php
date_default_timezone_set("Asia/Calcutta");
echo date_default_timezone_get();
echo date('H:m:s');
?>

</body>
</html>
```

PHP date_diff() Function

The date_diff() function returns the difference between two DateTime objects.

Syntax

```
date_diff(datetime1, datetime2, absolute)
```

Parameter Values

Parameter	Description
datetime1	Required. Specifies a DateTime object

datetime2	Required. Specifies a DateTime object
absolute	Optional. Specifies a Boolean value. TRUE indicates that the interval/difference MUST be positive. Default is FALSE

Calculate the difference between two dates:

```
<?php
$date1=date_create("2013-03-15");
$date2=date_create("2013-12-12");
$diff=date_diff($date1,$date2);
?>
```

```
<!DOCTYPE html>
<html>
<body>

<?php
// Prints the day
echo date("l") . "<br>";

// Prints the day, date, month, year, time, AM or PM
echo date("l jS \of F Y h:i:s A") . "<br>";

// Prints October 3, 1975 was on a Friday
echo "Oct 3,1975 was on a ".date("l", mktime(0,0,0,10,3,1975)) . "<br>";

// Use a constant in the format parameter
echo date(DATE_RFC822) . "<br>";

// prints something like: 1975-10-03T00:00:00+00:00
echo date(DATE_ATOM,mktime(0,0,0,10,3,1975));
?>

</body>
</html>
```

PHP Global Variables - Superglobals

Super global variables are built-in variables that are always available in all scopes.

Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

PHP $GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

The example below shows how to use the super global variable $GLOBALS:

Example

```
<?php
$x = 75;
$y = 25;

function addition() {
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
```

```
}

addition();
echo $z;
?>
```

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function!

PHP $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

The example below shows how to use some of the elements in $_SERVER:

Example

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

The following table lists the most important elements that can go inside $_SERVER:

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request

$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script

$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP $_REQUEST

PHP $_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // collect value of input field
  $name = $_REQUEST['fname'];
  if (empty($name)) {
    echo "Name is empty";
  } else {
    echo $name;
  }
}
?>
```

```
</body>
</html>
```

PHP $_POST

PHP $_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // collect value of input field
  $name = $_POST['fname'];
  if (empty($name)) {
    echo "Name is empty";
  } else {
    echo $name;
  }
```

```
}
?>
</body>
</html>
```

PHP $_GET

PHP $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".

$_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>
<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>
</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "test_get.php", and you can then access their values in "test_get.php" with $_GET.

The example below shows the code in "test_get.php":

Example

```
<html>
<body>
<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>
</body>
</html>
```

PHP Classes and Objects

What is Object Oriented Programming

Object-Oriented Programming (OOP) is a programming model that is based on the concept of classes and objects. As opposed to procedural programming where the focus is on writing procedures or functions that perform operations on the data, in object-oriented programming the focus is on the creations of objects which contain both data and functions together.

Object-oriented programming has several advantages over conventional or procedural style of programming. The most important ones are listed below:

- It provides a clear modular structure for the programs.
- It helps you adhere to the "don't repeat yourself" (DRY) principle, and thus make your code much easier to maintain, modify and debug.
- It makes it possible to create more complicated behaviour with less code and shorter development time and high degree of reusability.

Note: The idea behind Don't Repeat Yourself (DRY) principle is reducing the repetition of code by abstracting out the code that are common for the application and placing them at a single place and reuse them instead of repeating it.

Understanding Classes and Objects

Classes and objects are the two main aspects of object-oriented programming. A class is a self-contained, independent collection of variables and functions which work together to perform one or more specific tasks, while objects are individual instances of a class.

A class acts as a template or blueprint from which lots of individual objects can be created. When individual objects are created, they inherit the same generic properties and behaviors, although each object may have different values for certain properties.

For example, think of a class as a blueprint for a house. The blueprint itself is not a house, but is a detailed plan of the house. While, an object is like an actual house built according to that blueprint. We can build several identical houses from the same

blueprint, but each house may have different paints, interiors and families inside, as shown in the illustration below.

Blueprint *Houses built according to the blueprint*

A class can be declared using the `class` keyword, followed by the name of the class and a pair of curly braces (`{}`), as shown in the following example.
Let's create a PHP file named Rectangle.php and put the following example code inside it so that our class code should be separated from rest of the program. We can then use it wherever it's needed by simply including the Rectangle.php file.

```
<?php
class Rectangle {
	// Declare  properties
	public $length = 0;
	public $width = 0;

	// Method to get the perimeter
	public function getPerimeter() {
		return (2 * ($this->length + $this->width));
	}

	// Method to get the area
	public function getArea() {
		return ($this->length * $this->width);
	}
}
?>
```

The public keyword before the properties and methods in the example above, is an access modifier, which indicates that this property or method is accessible from anywhere.

Note: Syntactically, variables within a class are called properties, whereas functions are called methods. Also class names conventionally are written in PascalCase i.e. each concatenated word starts with an uppercase letter (e.g. MyClass).

Once a class has been defined, objects can be created from the class with the `new` keyword. Class methods and properties can directly be accessed through this object instance.

The arrow symbol (`->`) is an OOP construct that is used to access contained properties and methods of a given object. Whereas, the pseudo-variable `$this` provides a reference to the calling object i.e. the object to which the method belongs.

The real power of object-oriented programming becomes evident when using multiple instances of the same class, as shown in the following example:

```
<?php

// Include class definition

require "Rectangle.php";


// Create multiple objects from the Rectangle class

$obj1 = new Rectangle;

$obj2 = new Rectangle;


// Call the methods of both the objects

echo "<br />Area1: " . $obj1->getArea(); // Output: 0

echo "<br />Area2: " . $obj2->getArea(); // Output: 0
```

```
// Set $obj1 properties values

$obj1->length = 30;

$obj1->width = 20;


// Set $obj2 properties values

$obj2->length = 35;

$obj2->width = 50;


// Call the methods of both the objects again

echo "<br />Area1: " . $obj1->getArea(); // Output: 600

echo "<br />Area2: " . $obj2->getArea(); // Output: 1750

?>
```

As you can see in the above example, calling the getArea() method on different objects causes that method to operate on a different set of data. Each object instance is completely independent, with its own properties and methods, and thus can be manipulated independently, even if they're of the same class.

Creating Constructors

A constructor is a special kind of method that differs from standard ones in three ways:

Its name is always __construct().

It is automatically and immediately called whenever an object of that class is created.

It cannot have a return statement.

The syntax for defining a constructor is therefore

```
class ClassName {
  public $var;
  function __construct() {
    // Function code.
```

```
  }
}
```

A constructor could be used to connect to a database, set cookies, or establish initial values. Basically, you'll use constructors to do whatever should always be done—and done first—when an object of this class is made.

Using Constructors and Destructors

To make the object-oriented programming easier, PHP provides some magic methods that are executed automatically when certain actions occur within an object.

For example, the magic method __construct() (known as constructor) is executed automatically whenever a new object is created. Similarly, the magic method __destruct() (known as destructor) is executed automatically when the object is destroyed. A destructor function cleans up any resources allocated to an object once the object is destroyed.

```
<?php
class MyClass {
        // Constructor
        public function __construct() {
                echo 'The class "' . __CLASS__ . '" was initiated!<br>';
        }


        // Destructor
        public function __destruct() {
                echo '<br />The class "' . __CLASS__ . '" was destroyed.<br />';
        }
}
```

```
// Create a new object
$obj = new MyClass;

// Output a message at the end of the file
echo "<br />The end of the file is reached.";
?>
```

The PHP code in the above example will produce the following output:

```
The class "MyClass" was initiated!
The end of the file is reached.
The class "MyClass" was destroyed.
```

A destructor is called automatically when a scripts ends. However, to explicitly trigger the destructor, you can destroy the object using the PHP `unset()` function, as follow:

Note:

PHP automatically clean up all resources allocated during execution when the script is finished, e.g. closing database connections, destroying objects, etc.

The `__CLASS__` is a magic constant which contains the name of the class in which it is occur. It is empty, if it occurs outside of the class.

https://www.tutorialrepublic.com/php-tutorial/php-classes-and-objects.php

https://www.peachpit.com/articles/article.aspx?p=1949760&seqNum=5

https://t4tutorials.com/php-program-to-find-the-area-of-rectangle-with-form-and-database/

https://t4tutorials.com/php-progam-to-find-the-even-odd-number-with-form-and-database/

https://t4tutorials.com/search-a-record-from-database-in-php-mysql/

CONSTRUCTOR

- **PHP** 5 allows developers to declare **constructor methods for classes**.
- Constructor is suitable for any **initialization that the object may need before it is used**.
- We can design constructor using **"__construct" or same name as class name**.
- Parent constructors are not called implicitly if the child class defines a constructor. In order to run a parent constructor, a call to **parent::__construct()**.

Example1:

```
<?php
   class Example
   {
      public function __construct()
      {
         echo "Object Oriented Programming";
      }
   }
   $obj = new Example();
   $obj = new Example();
?>
```

Example2:

```
<?php
   class demo
   {
      public function demo()
      {
         echo "constructor1...";
      }
   }

   class demo1 extends demo
   {
      public function __construct()
      {
         echo parent::demo();
         echo "constructor2...";
```

```
        }
    }
    $obj= new demo1();
?>
```

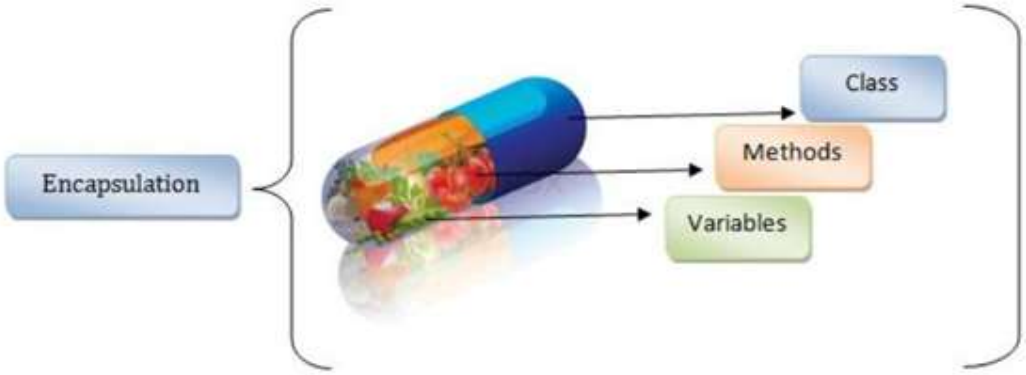
DESTRUCTOR

- **PHP 5** introduces a destructor concept similar to that of other object-oriented languages, such as C++.
- The destructor method will be called as soon as all references to a particular object are removed or when the object is **explicitly destroyed in any order in shutdown sequence.**
- We create destructor by using **"__destruct" function.**

```
<?php
    class demo
    {
        public function demo()
        {
            echo "constructor1...";
        }
    }

    class demo1 extends demo
    {
        public function __construct()
        {
            echo parent::demo();
            echo "constructor2...";
        }
        public function __destruct()
        {
            echo "destroy.....";
        }
    }
    $obj= new demo1();
?>
```

Encapsulation in PHP



- Encapsulation is a concept where we encapsulate all the **data and member functions** together to form an object.
- **Wrapping up data member and method together** into a single unit is called Encapsulation.
- Encapsulation also allows a **class to change its internal implementation** without hurting the overall functioning of the system.
- Binding the data with the code that manipulates it.
- It keeps the data and the code **safe from external interference**.

```
<?php
class person
{
public $name;
public $age;
function __construct($n, $a)
{
$this->name=$n;
$this->age=$a;
}
public function setAge($ag)
 {

$this->ag=$ag;

}

public function display()

{

echo  "welcome ".$this->name."<br/>";

return $this->age-$this->ag;
```

```
}
}
$person=new person("James Smith",28);
$person->setAge(1);
echo "You are ".$person->display()." years old";
?>
```

PHP Error Handling

Handling Errors

Sometimes your application will not run as it supposed to do, resulting in an error. There are a number of reasons that may cause errors, for example:

- The Web server might run out of disk space
- A user might have entered an invalid value in a form field
- The file or database record that you were trying to access may not exist
- The application might not have permission to write to a file on the disk
- A service that the application needs to access might be temporarily unavailable

These types of errors are known as runtime errors, because they occur at the time the script runs. They are distinct from syntax errors that need to be fixed before the script will run.

A professional application must have the capabilities to handle such runtime error gracefully. Usually this means informing the user about the problem more clearly and precisely.

Understanding Error Levels

Usually, when there's a problem that prevents a script from running properly, the PHP engine triggers an error. Each error is represented by an integer value and an associated constant. The following table list some of the common error levels:

Error Level	**Value**	**Description**
E_ERROR	1	A fatal run-time error, that can't be recovered from. The execution of the script is stopped immediately.
E_WARNING	2	A run-time warning. It is non-fatal and most errors tend to fall into this category. The execution of the script is not stopped.

E_NOTICE	8	A run-time notice. Indicate that the script encountered something that could possibly an error, although the situation could also occur when running a script normally.
E_USER_ERROR	256	A fatal user-generated error message. This is like an E_ERROR, except it is generated by the PHP script using the function trigger_error() rather than the PHP engine.
E_USER_WARNING	512	A non-fatal user-generated warning message. This is like an E_WARNING, except it is generated by the PHP script using the function trigger_error() rather than the PHP. engine
E_USER_NOTICE	1024	A user-generated notice message. This is like an E_NOTICE, except it is generated by the PHP script using the function trigger_error() rather than the PHP engine.
E_STRICT	2048	Not strictly an error, but triggered whenever PHP encounters code that could lead to problems or forward incompatibilities
E_ALL	8191	All errors and warnings, except of E_STRICT prior to PHP 5.4.0.

The PHP engine triggers an error whenever it encounters a problem with your script, but you can also trigger errors yourself to generate more user-friendly error messages. This way you can make your application more sophisticated. The following section describes some of common methods used for handling errors in PHP:

Basic Error Handling Using the die() Function

Consider the following example that simply tries to open a text file for reading only.

Example:

```
<?php
// Try to open a non-existent file
$file = fopen("sample.txt", "r");
?>
```

If the file does not exist you might get an error like this:

Warning: fopen(sample.txt) [function.fopen]: failed to open stream: No such file or directory in C:\wamp\www\project\test.php on line 2

If we follow some simple steps we can prevent the users from getting such error message.

Example:

```
<?php
if(file_exists("sample.txt")){
    $file = fopen("sample.txt", "r");
} else{
    die("Error: The file you are trying to access doesn't exist.");
}
?>
```

Now if you run the above script you will get the error message like this:

Error: The file you are trying to access doesn't exist.

As you can see by implementing a simple check whether the file exist or not before trying to access it, we can generate an error message that is more meaningful to the user.

The die() function used above simply display the custom error message and terminate the current script if 'sample.txt' file is not found.

Creating a Custom Error Handler

You can create your own error handler function to deal with the run-time error generated by PHP engine. The custom error handler provides you greater flexibility and better control over the errors, it can inspect the error and decide what to do with the error, it might display a message to the user, log the error in a file or database or send by e-mail, attempt to fix the problem and carry on, exit the execution of the script or ignore the error altogether.

The custom error handler function must be able to handle at least two parameters (errno and errstr), however it can optionally accept an additional three parameters (errfile, errline, and errcontext), as described below:

Parameter	**Description**
Required — The following parameters are required	
errno	Specifies the level of the error, as an integer. This corresponds to the appropriate error level constant (E_ERROR, E_WARNING, and so on)
errstr	Specifies the error message as a string
Optional — The following parameters are optional	
errfile	Specifies the filename of the script file in which the error occurred, as a string
errline	Specifies the line number on which the error occurred, as a string
errcontext	Specifies an array containing all the variables and their values that existed at the time the error occurred. Useful for debugging

Here's an example of a simple custom error handling function. This handler, `customError()` is triggered whenever an error occurred, no matter how trivial. It then outputs the details of the error to the browser and stops the execution of the script.

Example:

```
<?php
// Error handler function
function customError($errno, $errstr) {
    echo "<b>Error:</b> [$errno] $errstr";
}
```

You need to tell the PHP to use your custom error handler function — just call the built-in `set_error_handler()` function, passing in the name of the function.

Example:

```
<?php
// Error handler function
function customError($errno, $errstr) {
    echo "<b>Error:</b> [$errno] $errstr";
}

// Set error handler
set_error_handler("customError");

// Trigger error
echo ($test);
?>
```

Error Logging

Log Error Messages in a Text File

You can also logs details of the error to the log file, like this:

```
<?php
function calcDivision($dividend, $divisor) {
	if ($divisor == 0) {
		trigger_error("calcDivision(): The divisor cannot be zero", E_USER_WARNING);
		return false;
	} else {
		return ($dividend / $divisor);
	}
}
function customError($errno, $errstr, $errfile, $errline, $errcontext) {
	$message = date("Y-m-d H:i:s - ");
	$message .= "Error: [" . $errno . "], " . "$errstr in $errfile on line $errline, ";
	$message .= "Variables:" . print_r($errcontext, true) . "\r\n";

	error_log($message, 3, "logs/app_errors.log");
	die("There was a problem, please try again.");
}
set_error_handler("customError");
echo calcDivision(10, 0);
echo "This will never be printed.";
?>
```

Trigger an Error

Although the PHP engine triggers an error whenever it encounters a problem with your script, however you can also trigger errors yourself. This can help to make your application more robust, because it can flag potential problems before they turn into serious errors.

To trigger an error from within your script, call the trigger_error() function, passing in the error message that you want to generate:

```
trigger_error("There was a problem.");
```

Consider the following function that calculates division of the two numbers.

```
<?php
function calcDivision($dividend, $divisor){
   return($dividend / $divisor);
}

// Calling the function
echo calcDivision(10, 0);
?>
```

If a value of zero (0) is passed as the `$divisor` parameter, the error generated by the PHP engine will look something like this:

```
Warning: Division by zero in C:\wamp\www\project\test.php on line 3
```

This message doesn't look very informative. Consider the following example that uses the `trigger_error()` function to generate the error.

```
<?php
function calcDivision($dividend, $divisor) {
    if ($divisor == 0) {
        trigger_error("The divisor cannot be zero", E_USER_WARNING);
```

```
        return false;
    } else {
        return ($dividend / $divisor);
    }
}

// Calling the function
echo calcDivision(10, 0);
?>
```

Now the script generates this error message:

Warning: The divisor cannot be zero in C:\wamp\www\project\error.php on line 4

As you can see the error message generated by the second example explains the problem more clearly as compared to the previous one.

PHP Exception Handling

What is an Exception

An exception is a signal that indicates some sort of exceptional event or error has occurred. Exceptions can be caused due to various reasons, for example, database connection or query fails, file that you're trying to access doesn't exist, and so on.

Exceptions are thrown by many PHP functions and classes.

PHP provides a powerful exception handling mechanism that allows you to handle exceptions in a graceful way. As opposed to PHP's traditional error-handling system, exception handling is the object-oriented method for handling errors, which provides more controlled and flexible form of error reporting. Exception model was first introduced in PHP 5.

Using Throw and Try...Catch Statements

In exception-based approach, program code is written in a `try` block, an exception can be thrown using the `throw` statement when an exceptional event occurs during the execution of code in a `try` block. It is then caught and resolved by one or more `catch` blocks.

Syntax

```
try {
  code that can throw exceptions
} catch(Exception $e) {
  code that runs when an exception is caught
}
```

Example1:

```
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}

try {
```

```
  echo divide(5, 0);
} catch(Exception $e) {
  echo "Unable to divide.";
}
?>
```

The catch block indicates what type of exception should be caught and the name of the variable which can be used to access the exception. In the example above, the type of exception is Exception and the variable name is $e.

The following example demonstrates how exception handling works:

```
<?php

function division($dividend, $divisor) {

        // Throw exception if divisor is zero

        if ($divisor == 0) {

                throw new Exception('Division by zero.');

        } else {

                $quotient = $dividend / $divisor;

                echo "<p>$dividend / $divisor = $quotient</p>";

        }

}


try {

        division(10, 2);

        division(30, -4);

        division(15, 0);


        // If exception is thrown following line won't execute

        echo '<p>All divisions performed successfully.</p>';
```

```
} catch (Exception $e) {
    // Handle the exception
    echo "<p>Caught exception: " . $e->getMessage() . "</p>";
}

// Continue execution
echo "<p>Hello World!</p>";
?>
```

You might be wondering what this code was all about. Well, let's go through each part of this code one by one for a better understanding.

Explanation of Code

The PHP's exception handling system has basically four parts: `try`, `throw`, `catch`, and the Exception class. The following list describes how each part exactly works.

- The `division()` function in the example above checks if a divisor is equal to zero. If it is, an exception is thrown via PHP's `throw` statement. Otherwise this function perform the division using given numbers and display the result.
- Later, the `division()` function is called within a `try` block with different arguments. If an exception is generated while executing the code within the `try` block, PHP stops execution at that point and attempt to find the corresponding `catch` block. If it is found, the code within that `catch` block is executed, if not, a fatal error is generated.
- The `catch` block typically catch the exception thrown within the `try` block and creates an object (`$e`) containing the exception information. The error message from this object can be retrieved using the Exception's `getMessage()` method.

The try...catch...finally Statement

The try...catch...finally statement can be used to catch exceptions. Code in the finally block will always run regardless of whether an exception was caught. If finally is present, the catch block is optional.

Syntax

```
try {
  code that can throw exceptions
} catch(Exception $e) {
  code that runs when an exception is caught
} finally {
  code that always runs regardless of whether an exception was caught
}
```

Example1:

Show a message when an exception is thrown and then indicate that the process has ended:

```
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}

try {
  echo divide(5, 0);
} catch(Exception $e) {
  echo "Unable to divide. ";
} finally {
  echo "Process complete.";
}
?>
```

Example2:

Output a string even if an exception was not caught:

```
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
```

```
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}

try {
  echo divide(5, 0);
} finally {
  echo "Process complete.";
}
?>
```

The Exception Object

The Exception Object contains information about the error or unexpected behaviour that the function encountered.

Syntax

```
new Exception(message, code, previous)
```

Parameter Values

Parameter	Description
message	Optional. A string describing why the exception was thrown
code	Optional. An integer that can be used used to easily distinguish this exception from others of the same type
previous	Optional. If this exception was thrown in a catch block of another exception, it is recommended to pass that exception into this parameter

Methods

When catching an exception, the following table shows some of the methods that can be used to get information about the exception:

Method	Description
getMessage()	Returns a string describing why the exception was thrown
getPrevious()	If this exception was triggered by another one, this method returns the previous exception. If not, then it returns null
getCode()	Returns the exception code
getFile()	Returns the full path of the file in which the exception was thrown
getLine()	Returns the line number of the line of code which threw the exception

Example

Output information about an exception that was thrown:

```
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero", 1);
  }
  return $dividend / $divisor;
}

try {
  echo divide(5, 0);
} catch(Exception $ex) {
  $code = $ex->getCode();
  $message = $ex->getMessage();
  $file = $ex->getFile();
  $line = $ex->getLine();
  echo "Exception thrown in $file on line $line: [Code $code]
  $message";
```

```
}
?>
```

The PHP's Exception class also provides getCode(), getFile(), getLine() and getTraceAsString() methods that can be used to generate detailed debugging information.

```
<?php

// Turn off default error reporting

error_reporting(0);


try {

	$file = "somefile.txt";


	// Attempt to open the file

	$handle = fopen($file, "r");

	if (!$handle) {

		throw new Exception("Cannot open the file!", 5);

	}


	// Attempt to read the file contents

	$content = fread($handle, filesize($file));

	if (!$content) {

		throw new Exception("Could not read file!", 10);

	}


	// Closing the file handle
```

```
        fclose($handle);


        // Display file contents
        echo $content;
} catch (Exception $e) {
        echo "<h3>Caught Exception!</h3>";
        echo "<p>Error message: " . $e->getMessage() . "</p>";
        echo "<p>File: " . $e->getFile() . "</p>";
        echo "<p>Line: " . $e->getLine() . "</p>";
        echo "<p>Error code: " . $e->getCode() . "</p>";
        echo "<p>Trace: " . $e->getTraceAsString() . "</p>";
}
?>
```

The Exception's constructor optionally takes an exception message and an exception code. While the exception message is typically used to display generic information on what went wrong, the exception code can be used to categorize the errors. The exception code provided can be retrieved later via Exception's getCode() method.

Tip: Exception should only be used to denote exceptional conditions; they should not be used to control normal application flow e.g., jump to another place in the script at a particular point. Doing that would adversely affect your application's performance.

Proper exception code should include:

1. try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
2. throw - This is how you trigger an exception. Each "throw" must have at least one "catch"

3. catch - A "catch" block retrieves an exception and creates an object containing the exception information

```
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}

//trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below';
}

//catch exception
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
?>
```

The code above will get an error like this:

Message: Value must be 1 or below

Example explained:

The code above throws an exception and catches it:

1. The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown
2. The checkNum() function is called in a "try" block

3. The exception within the checkNum() function is thrown
4. The "catch" block retrieves the exception and creates an object ($e) containing the exception information
5. The error message from the exception is echoed by calling $e->getMessage() from the exception object

However, one way to get around the "every throw must have a catch" rule is to set a top level exception handler to handle errors that slip through.

Creating a Custom Exception Class

To create a custom exception handler you must create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.

The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.

Lets create an exception class:

```
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
    return $errorMsg;
  }
}

$email = "someone@example...com";

try {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throw exception if email is not valid
```

```
    throw new customException($email);
  }
}

catch (customException $e) {
  //display custom message
  echo $e->errorMessage();
}
?>
```

The new class is a copy of the old exception class with an addition of the errorMessage() function. Since it is a copy of the old class, and it inherits the properties and methods from the old class, we can use the exception class methods like getLine() and getFile() and getMessage().

Example explained:

The code above throws an exception and catches it with a custom exception class:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is not a valid e-mail address
4. The "try" block is executed and an exception is thrown since the e-mail address is invalid
5. The "catch" block catches the exception and displays the error message

Multiple Exceptions

It is possible for a script to use multiple exceptions to check for multiple conditions.

It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages:

```php
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
    return $errorMsg;
  }
}

$email = "someone@example.com";

try {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throw exception if email is not valid
    throw new customException($email);
  }
  //check for "example" in mail address
  if(strpos($email, "example") !== FALSE) {
    throw new Exception("$email is an example e-mail");
  }
}

catch (customException $e) {
  echo $e->errorMessage();
}

catch(Exception $e) {
  echo $e->getMessage();
}
?>
```

Example explained:

The code above tests two conditions and throws an exception if any of the conditions are not met:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block is executed and an exception is not thrown on the first condition
5. The second condition triggers an exception since the e-mail contains the string "example"
6. The "catch" block catches the exception and displays the correct error message

If the exception thrown were of the class customException and there were no customException catch, only the base exception catch, the exception would be handled there.

Re-throwing Exceptions

Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way. It is possible to throw an exception a second time within a "catch" block.

A script should hide system errors from users. System errors may be important for the coder, but are of no interest to the user. To make things easier for the user you can re-throw the exception with a user friendly message:

```
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = $this->getMessage().' is not a valid E-Mail address.';
    return $errorMsg;
  }
```

```
}

$email = "someone@example.com";

try {
  try {
    //check for "example" in mail address
    if(strpos($email, "example") !== FALSE) {
      //throw exception if email is not valid
      throw new Exception($email);
    }
  }
  catch(Exception $e) {
    //re-throw exception
    throw new customException($email);
  }
}

catch (customException $e) {
  //display custom message
  echo $e->errorMessage();
}
?>
```

Example explained:

The code above tests if the email-address contains the string "example" in it, if it does, the exception is re-thrown:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid

3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block contains another "try" block to make it possible to re-throw the exception
5. The exception is triggered since the e-mail contains the string "example"
6. The "catch" block catches the exception and re-throws a "customException"
7. The "customException" is caught and displays an error message

Rules for exceptions

- Code may be surrounded in a try block, to help catch potential exceptions
- Each try block or "throw" must have at least one corresponding catch block
- Multiple catch blocks can be used to catch different classes of exceptions
- Exceptions can be thrown (or re-thrown) in a catch block within a try block

PHP Regular Expressions

What is a Regular Expression?

A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of text search and text replace operations.

Syntax

In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

```
$exp = "/w3schools/i";
```

In the example above, / is the **delimiter**, *w3schools* is the **pattern** that is being searched for, and i is a **modifier** that makes the search case-insensitive.

The delimiter can be any character that is not a letter, number, backslash or space. The most common delimiter is the forward slash (/), but when your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

Regular Expression Functions

PHP provides a variety of functions that allow you to use regular expressions. The preg_match(), preg_match_all() and preg_replace() functions are some of the most commonly used ones:

Function	Description
preg_match()	Returns 1 if the pattern was found in the string and 0 if not

preg_match_all()	Returns the number of times the pattern was found in the string, which may also be 0
preg_replace()	Returns a new string where matched patterns have been replaced with another string

Using preg_match()

The preg_match() function will tell you whether a string contains matches of a pattern.

Example

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
<?php
$str = "Visit W3Schools";
$pattern = "/w3schools/i";
echo preg_match($pattern, $str); // Outputs 1
?>
```

Example 2:

```
<?php
$pattern = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
if(preg_match($pattern, $text)){
    echo "Match found!";
} else{
    echo "Match not found.";
}
?>
```

Using preg_match_all()

The preg_match_all() function will tell you how many matches were found for a pattern in a string.

Example

Use a regular expression to do a case-insensitive count of the number of occurrences of "ain" in a string:

```
<?php
$str = "The rain in SPAIN falls mainly on the plains.";
$pattern = "/ain/i";
echo preg_match_all($pattern, $str); // Outputs 4
?>
```

Example 2:

```
<?php
$pattern = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";
?>
```

Using preg_replace()

The preg_replace() function will replace all of the matches of the pattern in a string with another string.

Example

Use a case-insensitive regular expression to replace Microsoft with W3Schools in a string:

```php
<?php
$str = "Visit Microsoft!";
$pattern = "/microsoft/i";
echo preg_replace($pattern, "W3Schools", $str); // Outputs "Visit W3Schools!"
?>
```

Predefined Character Classes

Some character classes such as digits, letters, and whitespaces are used so frequently that there are shortcut names for them. The following table lists those predefined character classes:

Shortcut	What it Does
.	Matches any single character except newline \n.
\d	matches any digit character. Same as [0-9]
\D	Matches any non-digit character. Same as [^0-9]
\s	Matches any whitespace character (space, tab, newline or carriage return character). Same as [\t\n\r]
\S	Matches any non-whitespace character. Same as [^ \t\n\r]
\w	Matches any word character (definned as a to z, A to Z,0 to 9, and the underscore). Same as [a-zA-Z_0-9]
\W	Matches any non-word character. Same as [^a-zA-Z_0-9]

The following example will show you how to find and replace space with a hyphen character in a string using regular expression and PHP preg_replace() function:

Example:

```php
<?php
$pattern = "/\s/";
$replacement = "-";
```

```
$text = "Earth revolves around\nthe\tSun";

// Replace spaces, newlines and tabs

echo preg_replace($pattern, $replacement, $text);

echo "<br>";

// Replace only spaces

echo str_replace(" ", "-", $text);

?>
```

Repetition Quantifiers:

Example:

```
<?php

$pattern = "/[\s,]+/";

$text = "My favourite colors are red, green and blue";

$parts = preg_split($pattern, $text);


// Loop through parts array and display substrings

foreach($parts as $part){

   echo $part . "<br>";

}

?>
```

The regular expression in the following example will display only those names from the names array which start with the letter "J" using the PHP `preg_grep()` function:

```
<?php

$pattern = "/^J/";

$names = array("Jhon Carter", "Clark Kent", "John Rambo");

$matches = preg_grep($pattern, $names);
```

```
// Loop through matches array and display matched names
foreach($matches as $match){
    echo $match . "<br>";
}
?>
```

The following example will show you how to perform a global case-insensitive search using the `i` modifier and the PHP `preg_match_all()` function.

```
<?php
$pattern = "/color/i";
$text = "Color red is more visible than color blue in daylight.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";
?>
```

PHP Filters

PHP filters are used to validate and filter data coming from insecure sources, like user input.

To test, validate, and filter user input or custom data is an important part of any web application.

The PHP filter extension is designed to make data filtering easier and quicker.

Validating data = Determine if the data is in proper form.

Sanitizing data = Remove any illegal character from the data.

Why Use Filters?

Many web applications receive external input. External input/data can be:

- User input from a form
- Cookies
- Web services data
- Server variables
- Database query results

Almost all web applications depend on external input. Usually, this comes from a user or another application (like a web service). By using filters you can be sure the application gets the correct input.

You should always validate external data!

Invalid submitted data can lead to security problems and break your webpage!

By using PHP filters you can be sure your application gets the correct input!

Validating and Sanitizing Data with Filters

Sanitizing and validating user input is one of the most common tasks in a web application. To make this task easier PHP provides native filter extension that you can use to sanitize or validate data such as e-mail addresses, URLs, IP addresses, etc.

To validate data using filter extension you need to use the PHP's filter_var() function. The basic syntax of this function can be given with:

```
filter_var (variable, filter, options)
```

This function takes three parameters out of which the last two are optional. The first parameter is the value to be filtered, the second parameter is the ID of the filter to apply, and the third parameter is the array of options related to filter. Let's see how it works.

Filter Functions:

The filter function is used to filter the data coming from insecure source.

- **filter_var():** Filters a specific variable
- **filter_var_array():**Filters multiple variable i.e. array of variable
- **filter_has_var():** Check if the variable of specific input type exists or not
- **filter_id():**helps to get filter id of the specified filter name
- **filter_list():**Returns a list of supported filter name in the form of array.
- **filter_input():**Gets an external variable and filters it if set to do so.
- **filter_input_array():**same as filter_input() but here Gets multiple variables i.e. array of variable and filters them if set to do so.

Predefined Filter Constants:

There are many predefined filter constants which are listed below:

- **Validate filter constants:**
  - **FILTER_VALIDATE_BOOLEAN:** Validates a boolean
  - **FILTER_VALIDATE_INT:** Validates an integer
  - **FILTER_VALIDATE_FLOAT:** Validates a float
  - **FILTER_VALIDATE_REGEXP:** Validates a regular expression
  - **FILTER_VALIDATE_IP:** Validates an IP address
  - **FILTER_VALIDATE_EMAIL:** Validates an e-mail address
  - **FILTER_VALIDATE_URL:** Validates an URL

- **Sanitize filter constants:**
  - **FILTER_SANITIZE_EMAIL:** Removes all illegal characters from an e-mail address
  - **FILTER_SANITIZE_ENCODED:** Removes/Encodes special characters
  - **FILTER_SANITIZE_MAGIC_QUOTES:** Apply addslashes() function
  - **FILTER_SANITIZE_NUMBER_FLOAT:** Remove all characters, except digits, +- and optionally ., eE
  - **FILTER_SANITIZE_NUMBER_INT:** Removes all characters except digits and + –
  - **FILTER_SANITIZE_SPECIAL_CHARS:** Removes special characters
  - **FILTER_SANITIZE_FULL_SPECIAL_CHARS** Encoding quotes can be disabled by using FILTER_FLAG_NO_ENCODE_QUOTES.
  - **FILTER_SANITIZE_STRING :** Removes tags/special characters from a string
  - **FILTER_SANITIZE_STRIPPED :** Alias of FILTER_SANITIZE_STRING
  - **FILTER_SANITIZE_URL:** Removes all illegal character from s URL
- **Other filter constants:**
  - **FILTER_UNSAFE_RAW :**Do nothing, optionally strip/encode special characters
  - **FILTER_CALLBACK :**Call a user-defined function to filter data

Sanitize a String

The following example will sanitize a string by removing all HTML tags from it:

```
<?php

// Sample user comment

$comment = "<h1>Hey there! How are you doing today?</h1>";


// Sanitize and print comment string

$sanitizedComment = filter_var($comment, FILTER_SANITIZE_STRING);

echo $sanitizedComment;

?>
```

The output of the above example will look something like this:

Hey there! How are you doing today?

Validate Integer Values

The following example will validate whether the value is a valid integer or not.

```
<?php

// Sample integer value

$int = 20;


// Validate sample integer value

if (filter_var($int, FILTER_VALIDATE_INT)) {

        echo "The <b>$int</b> is a valid integer";

} else {

        echo "The <b>$int</b> is not a valid integer";

}

?>
```

In the above example, if variable `$int` is set to 0, the example code will display invalid integer message. To fix this problem, you need to explicitly test for the value 0, as follow:

```
<?php

// Sample integer value

$int = 0;


// Validate sample integer value

if (filter_var($int, FILTER_VALIDATE_INT) === 0 || filter_var($int,
FILTER_VALIDATE_INT)) {
```

```
    echo "The <b>$int</b> is a valid integer";
} else {
    echo "The <b>$int</b> is not a valid integer";
}
?>
```

Validate IP Addresses

The following example will validate whether the value is a valid IP address or not.

```
<!-- Validate IP Address -->
<?php
// Sample IP address
$ip = "172.16.254.1";

// Validate sample IP address
if (filter_var($ip, FILTER_VALIDATE_IP)) {
    echo "The <b>$ip</b> is a valid IP address";
} else {
    echo "The <b>$ip</b> is not a valid IP address";
}
?>
```

You can further apply validation for IPV4 or IPV6 IP addresses by using the `FILTER_FLAG_IPV4` or `FILTER_FLAG_IPV6` flags, respectively. Here's an example:

```
<?php
// Sample IP address
```

```
$ip = "172.16.254.1";


// Validate sample IP address
if (filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6)) {
        echo "The <b>$ip</b> is a valid IPV6 address";
} else {
        echo "The <b>$ip</b> is not a valid IPV6 address";
}
?>
```

Sanitize and Validate Email Addresses

The following example will show you how to sanitize and validate an e-mail address.

```
<!-- Sanitize valid email address -->
<?php
// Sample email address
$email = "someone@@example.com";


// Remove all illegal characters from email
$sanitizedEmail = filter_var($email, FILTER_SANITIZE_EMAIL);


// Validate email address
if ($email == $sanitizedEmail && filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "The $email is a valid email address";
} else {
        echo "The $email is not a valid email address";
}
```

```
?>
```

Sanitize and Validate URLs

The following example will show you how to sanitize and validate a url.

```
<?php

// Sample website url

$url = "http:://www.example.com";


// Remove all illegal characters from url

$sanitizedUrl = filter_var($url, FILTER_SANITIZE_URL);


// Validate website url

if ($url == $sanitizedUrl && filter_var($url, FILTER_VALIDATE_URL)) {

        echo "The $url is a valid website url";

} else {

        echo "The $url is not a valid website url";

}

?>
```

You can also check whether a URL contains query string or not by using the flag `FILTER_FLAG_QUERY_REQUIRED`, as shown in the following example:

```
<?php

// Sample website url

$url = "http://www.example.com?topic=filters";


// Validate website url for query string
```

```
if (filter_var($url, FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED)) {
    echo "The <b>$url</b> contains query string";
} else {
    echo "The <b>$url</b> does not contain query string";
}
?>
```

Validate Integers Within a Range

The following example will validate whether the supplied value is an integer or not, as well as whether it lies within the range of 0 to 100 or not.

```
<?php
// Sample integer value
$int = 75;

// Validate sample integer value
if (filter_var($int, FILTER_VALIDATE_INT, array("options" => array("min_range" => 0, "max_range" => 100)))) {
    echo "The <b>$int</b> is within the range of 0 to 100";
} else {
    echo "The <b>$int</b> is not within the range of 0 to 100";
}
?>
```

PHP - AJAX Introduction

AJAX is about updating parts of a web page, without reloading the whole page.

AJAX is a technique for creating fast and dynamic web pages.

What is AJAX?

AJAX = Asynchronous JavaScript and XML.

AJAX is a technique for creating fast and dynamic web pages.

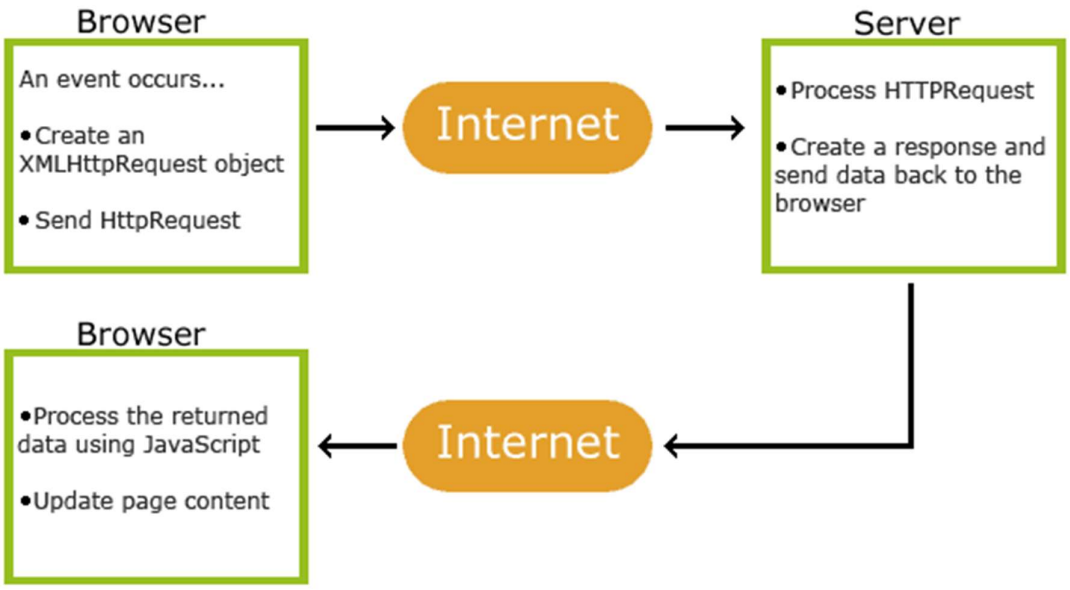
AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.

- AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and JavaScript.
- Conventional web applications transmit information to and from the server 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 submit is pressed, 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.

Examples of applications using AJAX: Google Maps, Gmail, YouTube, and Facebook tabs.

How AJAX Works



AJAX is Based on Internet Standards

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

AJAX applications are browser- and platform-independent!

AJAX - The XMLHttpRequest Object

The keystone of AJAX is the XMLHttpRequest object.

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, Edge (and IE7+), Safari, Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

```
variable = new XMLHttpRequest();
```

Example

```
var xhttp = new XMLHttpRequest();
```

XMLHttpRequest Object Properties:

Property	**Description**
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found"

statusText	Returns the status-text (e.g. "OK" or "Not Found")

AJAX - Send a Request To a Server

The XMLHttpRequest object is used to exchange data with a server.

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Method	**Description**
open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous)
send()	Sends the request to the server (used for GET)
send(*string*)	Sends the request to the server (used for POST)

PHP GET and POST

Methods of Sending Information to Server

A web browser communicates with the server typically using one of the two HTTP (Hypertext Transfer Protocol) methods — GET and POST. Both methods pass the information differently and have different advantages and disadvantages, as described below.

The GET Method

In GET method the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands (&). In general, a URL with GET data will look like this:

http://www.example.com/action.php?**name**=john&**age**=24

The bold parts in the URL are the GET parameters and the italic parts are the value of those parameters. More than one parameter=value can be embedded in the URL by concatenating with ampersands (&). One can only send simple text data via GET method.

Advantages and Disadvantages of Using the GET Method

- Since the data sent by the GET method are displayed in the URL, it is possible to bookmark the page with specific query string values.
- The GET method is not suitable for passing sensitive information such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.
- Because the GET method assigns data to a server environment variable, the length of the URL is limited. So, there is a limitation for the total data to be sent.

PHP provides the superglobal variable `$_GET` to access all the information sent either through the URL or submitted through an HTML form using the `method="get"`.

Example:

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
<title>Example of PHP GET method</title>
</head>
<body>
<?php
if(isset($_GET["name"]))
{
 echo "<p>Hi, " . $_GET["name"] . "</p>";
}
?>
<form method="get" action="<?php echo $_SERVER["PHP_SELF"];?>">
<label for="inputName">Name:</label>
<input type="text" name="name" id="inputName">
<input type="submit" value="Submit">
</form>
</body>
```

The POST Method

In POST method the data is sent to the server as a package in a separate communication with the processing script. Data sent through POST method will not visible in the URL.

Advantages and Disadvantages of Using the POST Method

- It is more secure than GET because user-entered information is never visible in the URL query string or in the server logs.
- There is a much larger limit on the amount of data that can be passed and one can send text data as well as binary data (uploading a file) using POST.

- Since the data sent by the POST method is not visible in the URL, so it is not possible to bookmark the page with specific query.

Like `$_GET`, PHP provide another superglobal variable `$_POST` to access all the information sent via post method or submitted through an HTML form using the `method="post"`.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Example of PHP POST method</title>
</head>
<body>
<?php
if(isset($_POST["name"]))
{
echo "<p>Hi, " . $_POST["name"] . "</p>";
} ?>
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
<label for="inputName">Name:</label>
<input type="text" name="name" id="inputName">
<input type="submit" value="Submit">
</form>
</body>
```

The $_REQUEST Variable

PHP provides another superglobal variable `$_REQUEST` that contains the values of both the `$_GET` and `$_POST` variables as well as the values of the `$_COOKIE` superglobal variable.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<title>Example of PHP $_REQUEST variable</title>
</head>
<body>
<?php
if(isset($_REQUEST["name"]))
{
echo "<p>Hi, " . $_REQUEST["name"] . "</p>";
}
?>
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
<label for="inputName">Name:</label>
<input type="text" name="name" id="inputName">
<input type="submit" value="Submit">
</form>
</body>
```

Note: The superglobal variables $_GET, $_POST and $_REQUEST are built-in variables that are always available in all scopes throughout a script.

PHP Form Validation

```
<?php
// Functions to filter user inputs
function filterName($field) {
        // Sanitize user name
        $field = filter_var(trim($field), FILTER_SANITIZE_STRING);

        // Validate user name
```

```
        if (filter_var($field, FILTER_VALIDATE_REGEXP, array("options" =>
array("regexp" => "/^[a-zA-Z\s]+$/")))) {
                return $field;
        } else {
                return FALSE;
        }
}
function filterEmail($field) {
        // Sanitize e-mail address
        $field = filter_var(trim($field), FILTER_SANITIZE_EMAIL);

        // Validate e-mail address
        if (filter_var($field, FILTER_VALIDATE_EMAIL)) {
                return $field;
        } else {
                return FALSE;
        }
}
function filterString($field) {
        // Sanitize string
        $field = filter_var(trim($field), FILTER_SANITIZE_STRING);
        if (!empty($field)) {
                return $field;
        } else {
                return FALSE;
```

```
        }
}

// Define variables and initialize with empty values
$nameErr = $emailErr = $messageErr = "";
$name = $email = $subject = $message = "";

// Processing form data when form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {

        // Validate user name
        if (empty($_POST["name"])) {
                $nameErr = "Please enter your name.";
        } else {
                $name = filterName($_POST["name"]);
                if ($name == FALSE) {
                        $nameErr = "Please enter a valid name.";
                }
        }

        // Validate email address
        if (empty($_POST["email"])) {
                $emailErr = "Please enter your email address.";
        } else {
                $email = filterEmail($_POST["email"]);
```

```
        if ($email == FALSE) {
                $emailErr = "Please enter a valid email address.";
        }
}

// Validate message subject
if (empty($_POST["subject"])) {
        $subject = "";
} else {
        $subject = filterString($_POST["subject"]);
}

// Validate user comment
if (empty($_POST["message"])) {
        $messageErr = "Please enter your comment.";
} else {
        $message = filterString($_POST["message"]);
        if ($message == FALSE) {
                $messageErr = "Please enter a valid comment.";
        }
}

// Check input errors before sending email
if (empty($nameErr) && empty($emailErr) && empty($messageErr)) {
        // Recipient email address
```

```
            $to = 'webmaster@example.com';

            // Create email headers
            $headers = 'From: ' . $email . "\r\n" .
            'Reply-To: ' . $email . "\r\n" .
            'X-Mailer: PHP/' . phpversion();

            // Sending email
            if (mail($to, $subject, $message, $headers)) {
                echo '<p class="success">Your message has been sent successfully!</p>';
            } else {
                echo '<p class="error">Unable to send email. Please try again!</p>';
            }
        }
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Form</title>
    <style type="text/css">
        .error{ color: red; }
```

```
      .success{ color: green; }
   </style>
</head>
<body>
   <h2>Contact Us</h2>
   <p>Please fill in this form and send us.</p>
   <form action="form.php" method="post">
      <p>
         <label for="inputName">Name:<sup>*</sup></label>
         <input type="text" name="name" id="inputName" value="<?php echo $name; ?>">
         <span class="error"><?php echo $nameErr; ?></span>
      </p>
      <p>
         <label for="inputEmail">Email:<sup>*</sup></label>
         <input type="text" name="email" id="inputEmail" value="<?php echo $email; ?>">
         <span class="error"><?php echo $emailErr; ?></span>
      </p>
      <p>
         <label for="inputSubject">Subject:</label>
         <input type="text" name="subject" id="inputSubject" value="<?php echo $subject; ?>">
      </p>
      <p>
         <label for="inputComment">Message:<sup>*</sup></label>
```

```
        <textarea name="message" id="inputComment" rows="5" cols="30"><?php
echo $message; ?></textarea>
        <span class="error"><?php echo $messageErr; ?></span>
    </p>
    <input type="submit" value="Send">
    <input type="reset" value="Reset">
  </form>
</body>
</html>
```

PHP JSON:

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easily read and written by humans and parsed and generated by machines. The application/json is the official Internet media type for JSON. The JSON filename extension is .json.

The json_encode function returns the JSON representation of the given value. The json_decode takes a JSON encoded string and converts it into a PHP variable.

PHP frameworks such as Symfony and Laravel have built-in methods that work with JSON.

PHP JSON encode

In the following example, we use the json_encode function.

encode.php

```
<?php

$data = ["falcon", "sky", "cloud", "orange", "wood", "forest"];

header('Content-type:application/json;charset=utf-8');
echo json_encode($data);
```

The example transforms a PHP array into a JSON string.

$ php -S localhost:8000 encode.php

We start the server and locate to the localhost:8000.

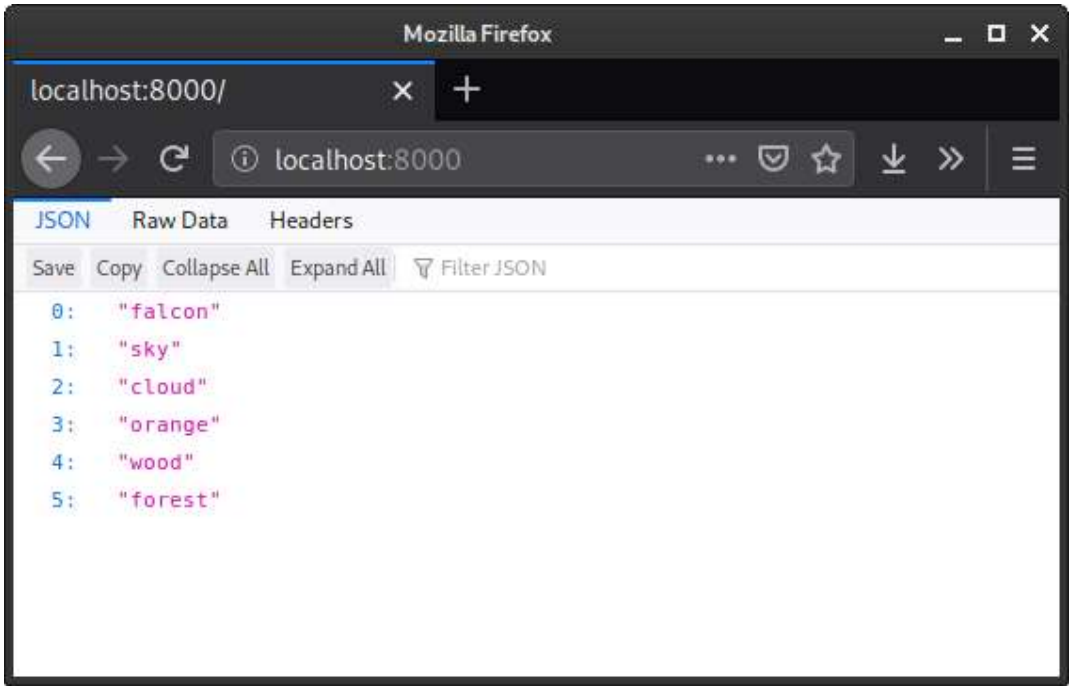


Figure: JSON view in Firefox

Modern web browsers show JSON view for JSON data when they receive an appropriate content type in the header of the response.

PHP JSON decode

In the following example, we use the json_decode function.

decode.php

```
<?php

$data = '{
   "name": "John Doe",
   "occupation": "gardener"
}';

$a = json_decode($data, true);

header('Content-type:text/html;charset=utf-8');
echo "{$a["name"]} is a {$a["occupation"]}";
```

The example transforms a JSON string into a PHP variable.

```
$ php -S localhost:8000 decode.php
```

We start the server.

```
$ curl localhost:8000
John Doe is a gardener
```

We send a GET request with curl.

PHP JSON read from file

In the following example, we read JSON data from a file.

data.json

```
[
   {"name": "John Doe", "occupation": "gardener", "country": "USA"},
   {"name": "Richard Roe", "occupation": "driver", "country": "UK"},
   {"name": "Sibel Schumacher", "occupation": "architect", "country": "Germany"},
   {"name": "Manuella Navarro", "occupation": "teacher", "country": "Spain"},
   {"name": "Thomas Dawn", "occupation": "teacher", "country": "New Zealand"},
   {"name": "Morris Holmes", "occupation": "programmer", "country": "Ireland"}
]
```

This is the JSON data.

readjson.php

```
<?php

$filename = 'data.json';

$data = file_get_contents($filename);
$users = json_decode($data);
?>

<html>
<table>
   <tbody>
      <tr>
         <th>Name</th>
         <th>Occupation</th>
         <th>Country</th>
```

```
      </tr>
      <?php foreach ($users as $user) { ?>
      <tr>
         <td> <?= $user->name; ?> </td>
         <td> <?= $user->occupation; ?> </td>
         <td> <?= $user->country; ?> </td>
      </tr>
      <?php } ?>
   </tbody>
</table>
</html>
```

In the code example, we read the file with file_get_contents and decode it into an PHP array with json_decode. Later, we place the data into a table utilizing PHP foreach loop.

PHP JSON read from database

In the following example, we read data from an SQLite database and return it in JSON.

cities.sql

```
BEGIN TRANSACTION;
DROP TABLE IF EXISTS cities;

CREATE TABLE cities(id INTEGER PRIMARY KEY, name TEXT, population INTEGER);
INSERT INTO cities(name, population) VALUES('Bratislava', 432000);
INSERT INTO cities(name, population) VALUES('Budapest', 1759000);
INSERT INTO cities(name, population) VALUES('Prague', 1280000);
INSERT INTO cities(name, population) VALUES('Warsaw', 1748000);
INSERT INTO cities(name, population) VALUES('Los Angeles', 3971000);
INSERT INTO cities(name, population) VALUES('New York', 8550000);
INSERT INTO cities(name, population) VALUES('Edinburgh', 464000);
INSERT INTO cities(name, population) VALUES('Berlin', 3671000);
COMMIT;
```

This SQL code creates a cities table in SQLite.

```
$ sqlite3 test.db
sqlite> .read cities.sql
sqlite> SELECT * FROM cities;
1|Bratislava|432000
2|Budapest|1759000
3|Prague|1280000
4|Warsaw|1748000
5|Los Angeles|3971000
6|New York|8550000
7|Edinburgh|464000
8|Berlin|3671000
```

With the sqlite3 command line tool, we generate an SQLite database and create the cities table.

fetch_all.php

```
<?php

$db = new SQLite3('test.db');
$res = $db->query('SELECT * FROM cities');
$cities = [];

while ($row = $res->fetchArray()) {
   $cities[] = $row;
}

header('Content-type:application/json;charset=utf-8');
echo json_encode(['cities' => $cities]);
```

In the example, we retrieve the data from the database and return it as JSON.

PHP JSON and JS fetch API

In the following example, we use JavaScript fetch API to get the JSON data from a PHP script.

data.json

```
[
   {"name": "John Doe", "occupation": "gardener", "country": "USA"},
   {"name": "Richard Roe", "occupation": "driver", "country": "UK"},
   {"name": "Sibel Schumacher", "occupation": "architect", "country": "Germany"},
   {"name": "Manuella Navarro", "occupation": "teacher", "country": "Spain"},
   {"name": "Thomas Dawn", "occupation": "teacher", "country": "New Zealand"},
   {"name": "Morris Holmes", "occupation": "programmer", "country": "Ireland"}
]
```

The JSON data is stored in a file.

data.php

```
<?php

$filename = 'data.json';

$data = file_get_contents($filename);
header('Content-type:application/json;charset=utf-8');
echo $data;
```

We read the data and return it to the client.

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Home page</title>
   <style>
      th,
      td {
         font: 15px 'Segoe UI';
      }

      table,
      th,
      td {
         border: solid 1px #ddd;
         border-collapse: collapse;
         padding: 2px 3px;
         text-align: center;
      }
      tr:nth-child(odd) {background: #efefef}
      th {
         font-weight: bold;
      }
   </style>
</head>

<body>

   <button id="getData">Get data</button>
   <br>
   <br>
   <div id="output"></div>


   <script>

      const getBtn = document.getElementById('getData');
      const output = document.getElementById('output');
      const table = document.getElementById('table');

      getBtn.addEventListener('click', () => {

         fetch('/data.php')
            .then((res) => {
               return res.json();
            })
            .then((data) => {
               output.innerHTML = '';
```

```
            let table = createTableAndHeader();

            output.appendChild(table);
            appendRows(table, data);
        })
        .catch((err) => {
            console.log("error fetching data");
            console.log(err);
        })

});

function createTableAndHeader() {

    let table = document.createElement("table");

    let tr = table.insertRow(-1);
    let headers = ["Name", "Occupation", "Country"];

    for (let i = 0; i < 3; i++) {

        let th = document.createElement("th");
        th.innerHTML = headers[i];
        tr.appendChild(th);
    }

    return table;
}

function appendRows(table, data) {

    for (let i = 0; i < data.length; i++) {

        let tr = table.insertRow(-1);

        for (const [_, value] of Object.entries(data[i])) {

            let cell = tr.insertCell(-1);
            cell.innerHTML = value;
        }
    }
}
  </script>
</body>

</html>
```

We have a button in the document. When we click on the button, the fetch function retrieves JSON data from the data.php script. An HTML table is dynamically built and filled with the data.

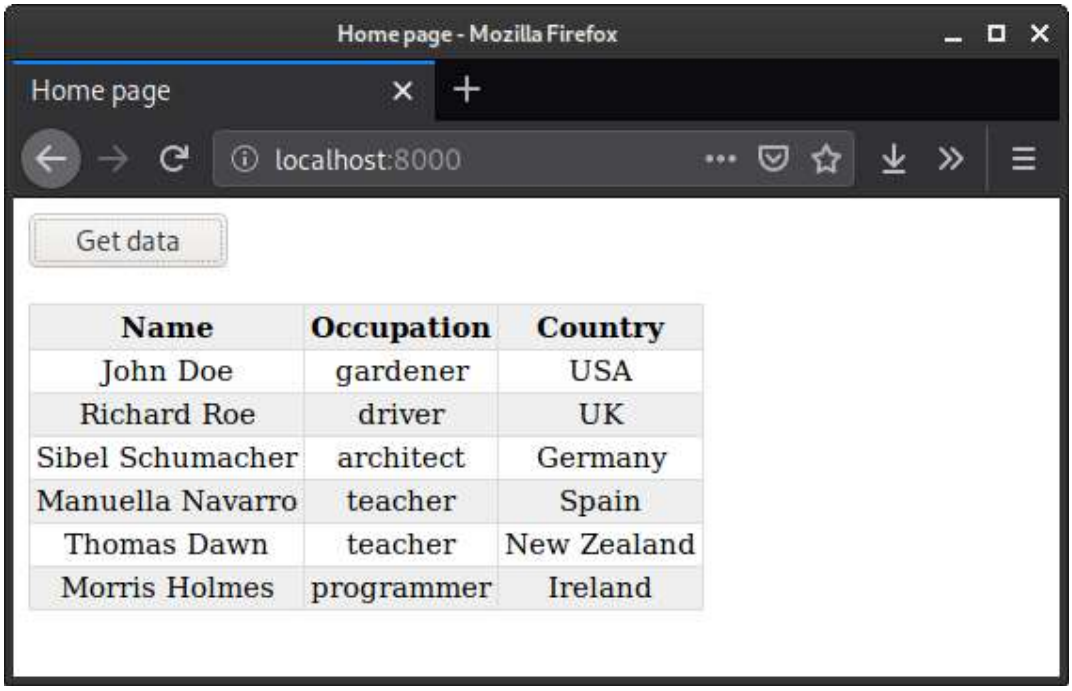


Figure: HTML table filled with JSON data from fetch API

https://zetcode.com/php/json/

https://jsonformatter.org/

https://www.youtube.com/watch?v=iiADhChRriM

VSCODE:

https://www.youtube.com/watch?v=VknMxAIbJj4

PHP - Frame Works

Frame Work is collection of software or program, that trigger off easy coding and implementing the code. It helps to programmer to achieve goals in short period of time. If PHP code is integrated with frame works, you can do anything with php coding skills.

Why use a PHP framework?

A PHP framework provides a basic structure for streamlining the development of web apps. We use them because they speed up the development process.

Above all, the responsiveness of websites and applications built using PHP frameworks helps businesses fulfill their performance needs. So, there’s no doubt that PHP frameworks will continue gaining attention and popularity in 2019.

Some of frame works

FuelPHP

Fuel PHP works based on Model View Control and having innovative plug ins. FuelPHP supports router based theory where you might route directly to a nearer the input uri, making the closure the controller and giving it control of further execution.

CakePHP

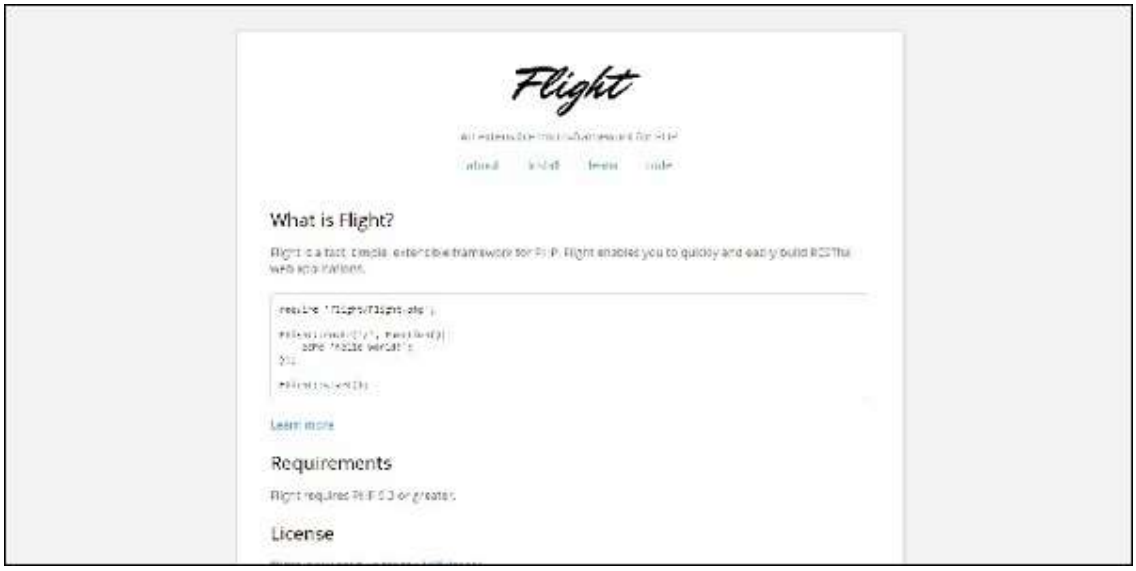
Cake PHP is a great source to build up simple and great web application in an easy way. Some great feature which are inbuilt in php are input validation, SQL injection prevention that keeps you application safe and secure.

Features

- Build Quickly
- No need to configure
- MIT licence
- MVC Model
- Secure

FlightPHP

Flight PHP is very helpful to make RESTful web services and it is under MIT licence.

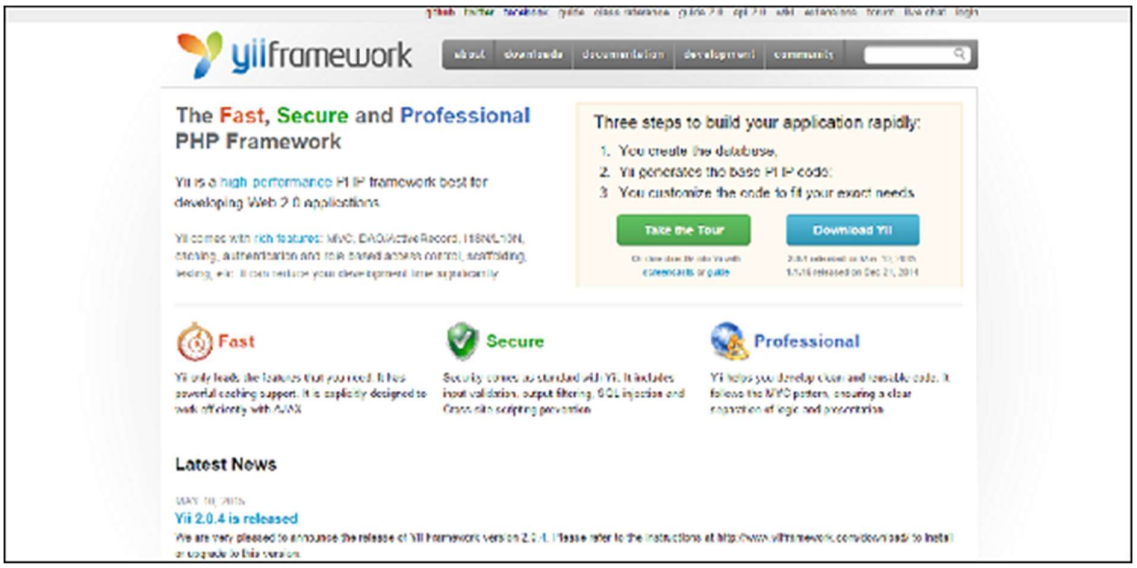


Symfony

Symfony is for highly professional developer to build websites with PHP components such as Drupal, PHPBB, laravel, eX, OROCRM and piwik.

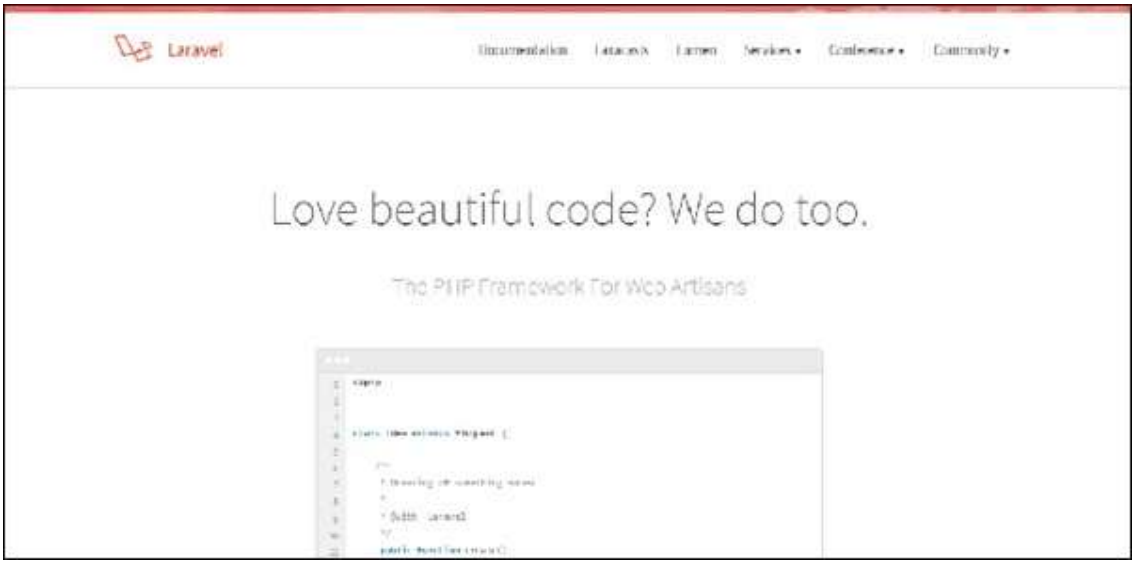
yiiFramework

YiiFramework works based on web 2.0 with high end security. It included input Validation, output filtering, and SQL injection.



Laravel

Laravel is most useful for RESRful Routing and light weight bled tempting engine. Laravel has integrated with some of great components of well tested and reliable code.



Zend

Zend is Modern frame work for performing high end web applications. This works based on Cryptographic and secure coding tools.

Codeigniter

Codeigiter is simple to develop small fool print for developer who need simple and elegant tool kit to create innovative web applications.

Reasons to use CodeIgniter

- CodeIgniter is a lightweight and straightforward PHP framework that's hassle-free to install, unlike other frameworks. Due to a simple setup process and highly illustrated documentation, it's ideal for beginners.
- Key features include an MVC architecture, top-notch error handling, inbuilt security tools, and simple and excellent documentation. In addition, it creates scalable apps.
- Compared with other frameworks, CodeIgniter is considerably faster. Since it also offers solid performance, it's a good choice when you want to develop lightweight applications to run on modest servers. One caveat: CodeIgniter releases are a bit irregular, so the framework isn't a great option for an application that requires high-level security.

Phalcon PHP

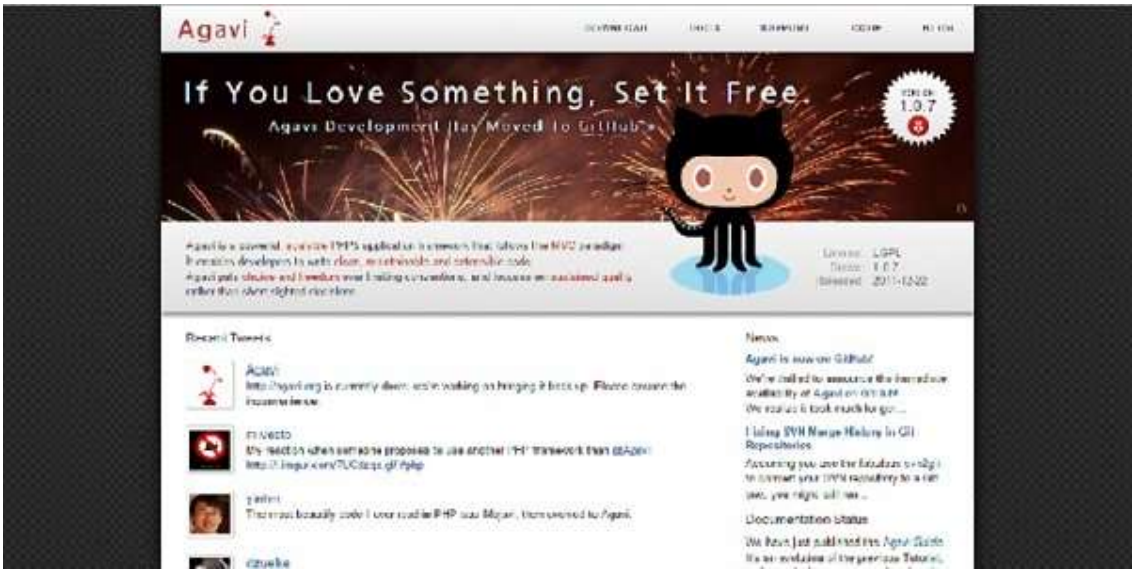
Pholcon PHP works based on MVC and integrated with innovative architecture to do perform faster.

PHPixie

PHPixie works based on MVC and designed for fast and reliability to develop web sites.

Agavi

Agavi is a powerful frame work and follows MVC model. It enables to developer to write clean and maintainable code.


Agavi
If You Love Something, Set It Free.
Agavi Development Has Moved To GitHub!
1.0.7
Recent Tweets
News
Documentation Status


The End