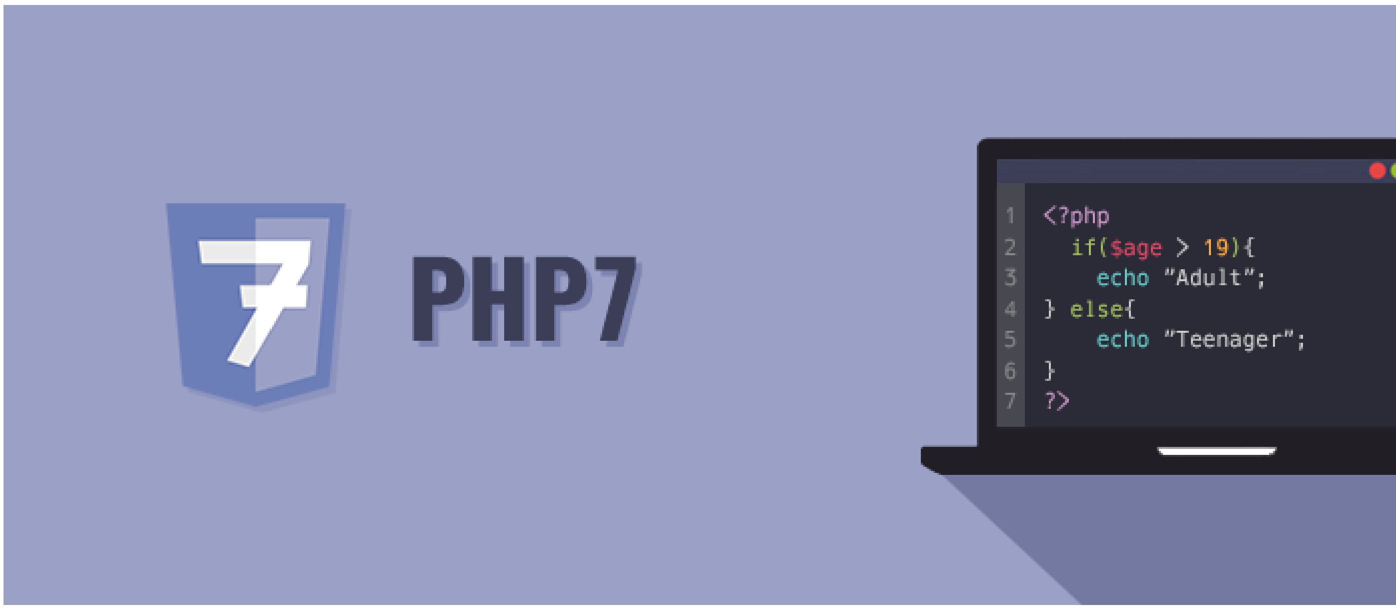
PHP Tutorial

PHP is the most popular server-side scripting language for creating dynamic web pages.



```
<?php
  if($age > 19){
    echo "Adult";
} else{
    echo "Teenager";
}
?>
```

PHP stands for Hypertext Preprocessor. PHP is a very popular and widely-used open source server-side scripting language to write dynamically generated web pages. PHP was originally created by Rasmus Lerdorf in 1994. It was initially known as Personal Home Page.

PHP scripts are executed on the server and the result is sent to the web browser as plain HTML. PHP can be integrated with the number of popular databases, including MySQL, PostgreSQL, Oracle, Microsoft SQL Server, Sybase, and so on. The current major version of PHP is 7. All of the code in this tutorial has been tested and validated against the most recent release of PHP 7.

PHP is very powerful language yet easy to learn and use. So bookmark this website and continued on.

Tip: Our PHP tutorial will help you to learn the fundamentals of the PHP scripting language, from the basic to advanced topics step-by-step. If you're a beginner, start with the basics and gradually move forward by learning a little bit every day.

What You Can Do with PHP

There are lot more things you can do with PHP.

- You can generate pages and files dynamically.
- You can create, open, read, write and close files on the server.
- You can collect data from a web form such as user information, email, phone no, etc.
- You can send emails to the users of your website.
- You can send and receive cookies to track the visitor of your website.
- You can store, delete, and modify information in your database.
- You can restrict unauthorized access to your website.
- You can encrypt data for safe transmission over internet.

The list does not end here, there are many other interesting things that you can do with PHP. You will learn about all of them in detail in upcoming chapters.

Advantages of PHP over Other Languages

If you're familiar with other server-side languages like ASP.NET or Java, you might be wondering what makes PHP so special. There are several advantages why one should choose PHP.

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

Tip: Do you know some huge websites like Facebook, Yahoo, Flickr, and Wikipedia are built using PHP. Most of the major content management systems (CMS), such as WordPress, Drupal, Joomla and Magento are also built in PHP.

What This Tutorial Covers

This PHP tutorial series covers all the fundamental programming concepts, including data types, operators, creating and using variables, generating outputs, structuring your code to make decisions in your programs or to loop over the same block of code multiple times, creating and manipulating strings and arrays, defining and calling functions, and so on.

Once you're comfortable with the basics, you'll move on to next level that explains the concept file system, sessions and cookies, dates and times, as well as how to send email from your script, handling and validating forms, perform data filtration and handling errors in PHP.

Finally, you'll explore some advanced concepts like classes and objects, parsing JSON data, pattern matching with regular expressions, exception handling as well as how to use PHP to manipulate data in MySQL database and create useful features like user login system, Ajax search, etc.

Tip: Every chapter in this tutorial contains lots of real-world examples that you can try and test using an online editor. These examples will help you to better understand the concept or topic. It also contains smart workarounds as well as useful tips and important notes.

PHP Getting Started

Install Wampserver or XAMPP on your PC to quickly create web applications with Apache, PHP and a MySQL database.

Getting Started with PHP

Here, you will learn how easy it is to create dynamic web pages using PHP. Before begin, be sure to have a code editor and some working knowledge of HTML and CSS.

If you're just starting out in web development, start learning from here »

Well, let's get straight into it.

Setting Up a Local Web Server

PHP script execute on a web server running PHP. So before you start writing any PHP program you need the following program installed on your computer.

- The Apache Web server
- The PHP engine
- The MySQL database server

You can either install them individually or choose a pre-configured package for your operating system like Linux and Windows. Popular pre-configured package are XAMPP and WampServer.

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. It will also provide the MySQL administrative tool PhpMyAdmin to easily manage your databases using a web browser.

The official website for downloading and installation instructions for the WampServer: http://www.wampserver.com/en/

Creating Your First PHP Script

Now that you have successfully installed WampServer on your computer. In this section we will create a very simple PHP script that displays the text "Hello, world!" in the browser window.

Ok, click on the WampServer icon somewhere on your Windows task bar and select the "www directory". Alternatively, you can access the "www" directory through navigating the `C:\wamp\www`. Create a subdirectory in "www" directory let's say "project".

Now open up your favorite code editor and create a new PHP file then type the following code:

Example
Run this code »

```
<?php
// Display greeting message
echo "Hello, world!";
?>
```

Now save this file as "hello.php" in your project folder (located at `C:\wamp\www\project`), and view the result in your browser through visiting this URL: `http://localhost/project/hello.php`.

Alternatively, you can access the "hello.php" file through selecting the localhost option and then select the project folder from the WampSever menu on the taskbar.

PHP can be embedded within a normal HTML web page. That means inside your HTML document you can write the PHP statements, as demonstrated in the follwoing example:

Example
Run this code »

```
<!DOCTYPE HTML>
<html>
<head>
    <title>PHP Application</title>
</head>
<body>
<?php
// Display greeting message
echo 'Hello World!';
?>
</body>
</html>
```

You will learn what each of these statements means in upcoming chapters.

PHP Syntax

The PHP script can be embedded within HTML web pages.

Standard PHP Syntax

A PHP script starts with the `<?php` and ends with the `?>` tag.

The PHP delimiter `<?php` and `?>` in the following example simply tells the PHP engine to treat the enclosed code block as PHP code, rather than simple HTML.

Example
Run this code »

```php
<?php
// Some code to be executed
echo "Hello, world!";
?>
```

Every PHP statement end with a semicolon (`;`) — this tells the PHP engine that the end of the current statement has been reached.

Embedding PHP within HTML

PHP files are plain text files with `.php` extension. Inside a PHP file you can write HTML like you do in regular HTML pages as well as embed PHP codes for server side execution.

Example
Run this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>A Simple PHP File</title>
</head>
<body>
    <h1><?php echo "Hello, world!"; ?></h1>
</body>
</html>
```

The above example shows how you can embed PHP codes within HTML to create well-formed dynamic web pages. If you view the source code of the resulting web page in your browser, the only difference you will see is the PHP code `<?php echo "Hello, world!"; ?>` has been replaced with the output "Hello, world!".

What happend here is? when you run this code the PHP engine exacuted the instructions between the `<?php … ?>` tags and leave rest of the thing as it is. At the end the web server send the final output back to your browser which is completely in HTML.

PHP Comments

A comment is simply text that is ignored by the PHP engine. The purpose of comments is to make the code more readable. It may help other developer (or you in the future when you edit the source code) to understand what you were trying to do with the PHP.

PHP support single-line as well as multi-line comments. To write a single-line comment either start the line with either two slashes (`//`) or a hash symbol (`#`). For example:

Example
Run this code »

```
<?php
// This is a single line comment
# This is also a single line comment
echo "Hello, world!";
?>
```

However to write multi-line comments, start the comment with a slash followed by an asterisk (`/*`) and end the comment with an asterisk followed by a slash (`*/`), like this:

Example
Run this code »

```
<?php
/*
This is a multiple line comment block
that spans across more than
one line
*/
echo "Hello, world!";
?>
```

Case Sensitivity in PHP

Variable names in PHP are case-sensitive. As a result the variables `$color`, `$Color` and `$COLOR` are treated as three different variables.

Example
Run this code »

```
<?php
// Assign value to variable
```

```
$color = "blue";

// Try to print variable value
echo "The color of the sky is " . $color . "<br>";
echo "The color of the sky is " . $Color . "<br>";
echo "The color of the sky is " . $COLOR . "<br>";
?>
```

If you try to run the above example code it will only display the value of the variable `$color` and produce the "Undefined variable" warning for the variable `$Color` and `$COLOR`.

However the keywords, function and classes names are case-insensitive. As a result calling the `gettype()` or `GETTYPE()` produce the same result.

Example
Run this code »

```
<?php
// Assign value to variable
$color = "blue";

// Get the type of a variable
echo gettype($color) . "<br>";
echo GETTYPE($color) . "<br>";
?>
```

If you try to run the above example code both the functions `gettype()` and `GETTYPE()` gives the same output, which is: string.

PHP Variables

In this tutorial you will learn how store information in a variable in PHP.

What is Variable in PHP

Variables are used to store data, like string of text, numbers, etc. Variable values can change over the course of a script. Here're some important things to know about variables:

- In PHP, a variable does not need to be declared before adding a value to it. PHP automatically converts the variable to the correct data type, depending on its value.
- After declaring a variable it can be reused throughout the code.
- The assignment operator (`=`) used to assign value to a variable.

In PHP variable can be declared as: `$var_name = value;`

Example
Run this code »

```php
<?php
// Declaring variables
$txt = "Hello World!";
$number = 10;

// Displaying variables value
echo $txt;  // Output: Hello World!
echo $number; // Output: 10
?>
```

In the above example we have created two variables where first one has assigned with a string value and the second has assigned with a number. Later we've displayed the variables values in the browser using the `echo` statement. The PHP `echo` statement is often used to output data to the browser. We will learn more about this in upcoming chapter.

Naming Conventions for PHP Variables

These are the following rules for naming a PHP variable:

- All variables in PHP start with a `$` sign, followed by the name of the variable.

- A variable name must start with a letter or the underscore character `_`.
- A variable name cannot start with a number.
- A variable name in PHP can only contain alpha-numeric characters and underscores (`A-z`, `0-9`, and `_`).
- A variable name cannot contain spaces.

Note: Variable names in PHP are case sensitive, it means `$x` and `$X` are two different variables. So be careful while defining variable names.

PHP Constants

In this tutorial you will learn how use constants for storing fixed values in PHP.

What is Constant in PHP

A constant is a name or an identifier for a fixed value. Constant are like variables, except that once they are defined, they cannot be undefined or changed (except magic constants).

Constants are very useful for storing data that doesn't change while the script is running. Common examples of such data include configuration settings such as database username and password, website's base URL, company name, etc.

Constants are defined using PHP's `define()` function, which accepts two arguments: the name of the constant, and its value. Once defined the constant value can be accessed at any time just by referring to its name. Here is a simple example:

Example
Run this code »

```
<?php
// Defining constant
define("SITE_URL",
"https://www.tutorialrepublic.com/");

// Using constant
echo 'Thank you for visiting - ' . SITE_URL;
?>
```

The output of the above code will be:

```
Thank you for visiting - https://www.tutorialrepublic.com/
```

The PHP `echo` statement is often used to display or output data to the web browser. We will learn more about this statement in the next chapter.

Tip: By storing the value in a constant instead of a variable, you can make sure that the value won't get changed accidentally when your application runs.

Naming Conventions for PHP Constants

Name of constants must follow the same rules as variable names, which means a valid constant name must starts with a letter or underscore, followed by any number of letters, numbers or underscores with one exception: the `$` prefix is not required for constant names.

Note: By convention, constant names are usually written in uppercase letters. This is for their easy identification and differentiation from variables in the source code.

PHP Echo and Print Statements

In this tutorial you will learn how to use the PHP `echo` and `print` statements to display the output in a web browser.

The PHP echo Statement

The echo statement can output one or more strings. In general terms, the echo statement can display anything that can be displayed to the browser, such as string, numbers, variables values, the results of expressions etc.

Since echo is a language construct not actually a function (like `if` statement), you can use it without parentheses e.g. `echo` or `echo()`. However, if you want to pass more than one parameter to echo, the parameters must not be enclosed within parentheses.

Display Strings of Text

The following example will show you how to display a string of text with the echo statement:

Example
Run this code »

```
<?php
// Displaying string of text
echo "Hello World!";
?>
```

The output of the above PHP code will look something like this:

Hello World!

Display HTML Code

The following example will show you how to display HTML code using the echo statement:

Example
Run this code »

```
<?php
// Displaying HTML code
echo "<h4>This is a simple heading.</h4>";
echo "<h4 style='color: red;'>This is heading with
style.</h4>";
?>
```

The output of the above PHP code will look something like this:

This is a simple heading.

This is heading with style.

Display Variables

The following example will show you how to display variable using the echo statement:

Example
Run this code »

```
<?php
// Defining variables
$txt = "Hello World!";
$num = 123456789;
$colors = array("Red", "Green", "Blue");

// Displaying variables
echo $txt;
echo "<br>";
echo $num;
echo "<br>";
echo $colors[0];
?>
```

The output of the above PHP code will look something like this:

```
Hello World!
123456789
Red
```

The PHP print Statement

You can also use the print statement (an alternative to `echo`) to display output to the browser. Like echo the print is also a language construct not a real function. So you can also use it without parentheses like: `print` or `print()`.

Both `echo` and `print` statement works exactly the same way except that the `print` statement can only output one string, and always returns 1. That's why the `echo` statement considered marginally faster than the `print` statement since it doesn't return any value.

Display Strings of Text

The following example will show you how to display a string of text with the print statement:

Example
Run this code »

```php
<?php
// Displaying string of text
print "Hello World!";
?>
```

The output of the above PHP code will look something like this:

```
Hello World!
```

Display HTML Code

The following example will show you how to display HTML code using the print statement:

Example
Run this code »

```
<?php
// Displaying HTML code
print "<h4>This is a simple heading.</h4>";
print "<h4 style='color: red;'>This is heading with
style.</h4>";
?>
```

The output of the above PHP code will look something like this:

This is a simple heading.

This is heading with style.

Display Variables

The following example will show you how to display variable using the print statement:

Example
Run this code »

```
<?php
// Defining variables
$txt = "Hello World!";
$num = 123456789;
$colors = array("Red", "Green", "Blue");

// Displaying variables
print $txt;
print "<br>";
print $num;
print "<br>";
print $colors[0];
?>
```

The output of the above PHP code will look something like this:

```
Hello World!
123456789
Red
```

PHP Data Types

In this tutorial you will learn about the data types available in PHP.

Data Types in PHP

The values assigned to a PHP variable may be of different data types including simple string and numeric types to more complex data types like arrays and objects.

PHP supports total eight primitive data types: Integer, Floating point number or Float, String, Booleans, Array, Object, resource and NULL. These data types are used to construct variables. Now let's discuss each one of them in detail.

PHP Integers

Integers are whole numbers, without a decimal point (..., -2, -1, 0, 1, 2, ...). Integers can be specified in decimal (base 10), hexadecimal (base 16 - prefixed with `0x`) or octal (base 8 - prefixed with `0`) notation, optionally preceded by a sign (`-` or `+`).

Example
Run this code »

```php
<?php
$a = 123; // decimal number
var_dump($a);
echo "<br>";
```

```
$b = -123; // a negative number
var_dump($b);
echo "<br>";

$c = 0x1A; // hexadecimal number
var_dump($c);
echo "<br>";

$d = 0123; // octal number
var_dump($d);
?>
```

Note: Since PHP 5.4+ you can also specify integers in binary (base 2) notation. To use binary notation precede the number with 0b (e.g. `$var = 0b11111111;`).

PHP Strings

Strings are sequences of characters, where every character is the same as a byte.

A string can hold letters, numbers, and special characters and it can be as large as up to 2GB (2147483647 bytes maximum). The simplest way to specify a string is to enclose it in single quotes (e.g. 'Hello world!'), however you can also use double quotes ("Hello world!").

Example

Run this code »

```
<?php
$a = 'Hello world!';
echo $a;
echo "<br>";

$b = "Hello world!";
echo $b;
echo "<br>";

$c = 'Stay here, I\'ll be back.';
echo $c;
?>
```

You will learn more about strings in PHP Strings tutorial.

PHP Floating Point Numbers or Doubles

Floating point numbers (also known as "floats", "doubles", or "real numbers") are decimal or fractional numbers, like demonstrated in the example below.

Example

Run this code »

```
<?php
$a = 1.234;
var_dump($a);
echo "<br>";

$b = 10.2e3;
var_dump($b);
echo "<br>";

$c = 4E-10;
var_dump($c);
?>
```

PHP Booleans

Booleans are like a switch it has only two possible values either `1` (true) or `0` (false).

Example
Run this code »

```
<?php
// Assign the value TRUE to a variable
$show_error = true;
var_dump($show_error);
?>
```

PHP Arrays

An array is a variable that can hold more than one value at a time. It is useful to aggregate a series of related items together, for example a set of country or city names.

An array is formally defined as an indexed collection of data values. Each index (also known as the key) of an array is unique and references a corresponding value.

Example
Run this code »

```
<?php
$colors = array("Red", "Green", "Blue");
var_dump($colors);
echo "<br>";

$color_codes = array(
    "Red" => "#ff0000",
    "Green" => "#00ff00",
    "Blue" => "#0000ff"
);
var_dump($color_codes);
```

```
?>
```

You will learn more about arrays in PHP Array tutorial.

PHP Objects

An object is a data type that not only allows storing data but also information on, how to process that data. An object is a specific instance of a class which serve as templates for objects. Objects are created based on this template via the new keyword.

Every object has properties and methods corresponding to those of its parent class. Every object instance is completely independent, with its own properties and methods, and can thus be manipulated independently of other objects of the same class.

Here's a simple example of a class definition followed by the object creation.

Example
Run this code »

```
<?php
// Class definition
class greeting{
    // properties
    public $str = "Hello World!";

    // methods
    function show_greeting(){
        return $this->str;
    }
}

// Create object from class
$message = new greeting;
var_dump($message);
?>
```

Tip: The data elements stored within an object are referred to as its properties and the information, or code which describing how to process the data is called the methods of the object.

PHP NULL

The special NULL value is used to represent empty variables in PHP. A variable of type NULL is a variable without any data. NULL is the only possible value of type null.

Example

Run this code »

```
<?php
$a = NULL;
var_dump($a);
echo "<br>";

$b = "Hello World!";
$b = NULL;
var_dump($b);
?>
```

When a variable is created without a value in PHP like `$var;` it is automatically assigned a value of null. Many novice PHP developers mistakenly considered both `$var1 = NULL;` and `$var2 = "";` are same, but this is not true. Both variables are different — the `$var1` has null value while `$var2` indicates no value assigned to it.

PHP Resources

A resource is a special variable, holding a reference to an external resource.

Resource variables typically hold special handlers to opened files and database connections.

Example
Run this code »

```
<?php
// Open a file for reading
$handle = fopen("note.txt", "r");
var_dump($handle);
echo "<br>";

// Connect to MySQL database server with default
setting
$link = mysqli_connect("localhost", "root", "");
var_dump($link);
?>
```

PHP Strings

In this tutorial you will learn how to store and manipulate strings in PHP.

What is String in PHP

A string is a sequence of letters, numbers, special characters and arithmetic values or combination of all. The simplest way to create a string is to enclose the string literal (i.e. string characters) in single quotation marks ('), like this:

```
$my_string = 'Hello World';
```

You can also use double quotation marks ("). However, single and double quotation marks work in different ways. Strings enclosed in single-quotes are treated almost literally, whereas the strings delimited by the double quotes replaces variables with the string representations of their values as well as specially interpreting certain escape sequences.

The escape-sequence replacements are:

- `\n` is replaced by the newline character
- `\r` is replaced by the carriage-return character
- `\t` is replaced by the tab character
- `\$` is replaced by the dollar sign itself (`$`)
- `\"` is replaced by a single double-quote (`"`)
- `\\` is replaced by a single backslash (`\`)

Here's an example to clarify the differences between single and double quoted strings:

Example

Run this code »

```
<?php
$my_str = 'World';
echo "Hello, $my_str!<br>";       // Displays: Hello
World!
echo 'Hello, $my_str!<br>';       // Displays: Hello,
$my_str!

echo '<pre>Hello\tWorld!</pre>'; // Displays:
Hello\tWorld!
echo "<pre>Hello\tWorld!</pre>"; // Displays: Hello
World!
echo 'I\'ll be back';             // Displays: I'll be
back
?>
```

Manipulating PHP Strings

PHP provides many built-in functions for manipulating strings like calculating the length of a string, find substrings or characters, replacing part of a string with different characters, take a string apart, and many others. Here are the examples of some of these functions.

Calculating the Length of a String

The `strlen()` function is used to calculate the number of characters inside a string. It also includes the blank spaces inside the string.

Example
Run this code »

```
<?php
$my_str = 'Welcome to Tutorial Republic';

// Outputs: 28
echo strlen($my_str);
?>
```

Counting Number of Words in a String

The `str_word_count()` function counts the number of words in a string.

Example
Run this code »

```
<?php
$my_str = 'The quick brown fox jumps over the lazy
dog.';
```

```
// Outputs: 9
echo str_word_count($my_str);
?>
```

Replacing Text within Strings

The `str_replace()` replaces all occurrences of the search text within the target string.

Example
Run this code »

```
<?php
$my_str = 'If the facts do not fit the theory, change
the facts.';

// Display replaced string
echo str_replace("facts", "truth", $my_str);
?>
```

The output of the above code will be:

If the truth do not fit the theory, change the truth.

You can optionally pass the fourth argument to the `str_replace()` function to know how many times the string replacements was performed, like this.

Example
Run this code »

```
<?php
$my_str = 'If the facts do not fit the theory, change
the facts.';

// Perform string replacement
str_replace("facts", "truth", $my_str, $count);
```

```
// Display number of replacements performed
echo "The text was replaced $count times.";
?>
```

The output of the above code will be:

The text was replaced 2 times.

Reversing a String

The `strrev()` function reverses a string.

Example
Run this code »

```
<?php
$my_str = 'You can do anything, but not everything.';

// Display reversed string
echo strrev($my_str);
?>
```

The output of the above code will be:

.gnihtyreve ton tub ,gnihtyna od nac uoY

PHP String Reference

For a complete list of useful string functions, please check out PHP String Reference.

PHP Operators

In this tutorial you will learn how to manipulate or perform the operations on variables and values using operators in PHP.

What is Operators in PHP

Operators are symbols that tell the PHP processor to perform certain actions. For example, the addition (+) symbol is an operator that tells PHP to add two variables or values, while the greater-than (>) symbol is an operator that tells PHP to compare two values.

The following lists describe the different operators used in PHP.

PHP Arithmetic Operators

The arithmetic operators are used to perform common arithmetical operations, such as addition, subtraction, multiplication etc. Here's a complete list of PHP's arithmetic operators:

Operator	**Description**	**Example**	**Result**
+	Addition	`$x + $y`	Sum of $x and $y
-	Subtraction	`$x - $y`	Difference of $x and $y.
*	Multiplication	`$x * $y`	Product of $x and $y.
/	Division	`$x / $y`	Quotient of $x and $y
%	Modulus	`$x % $y`	Remainder of $x divided by $

The following example will show you these arithmetic operators in action:

Example
Run this code »

```
<?php
$x = 10;
$y = 4;
echo($x + $y); // Outputs: 14
echo($x - $y); // Outputs: 6
echo($x * $y); // Outputs: 40
echo($x / $y); // Outputs: 2.5
echo($x % $y); // Outputs: 2
?>
```

PHP Assignment Operators

The assignment operators are used to assign values to variables.

Operator	Description	Example	Is The S
`=`	Assign	`$x = $y`	`$x =`
`+=`	Add and assign	`$x += $y`	`$x =`
`-=`	Subtract and assign	`$x -= $y`	`$x =`
`*=`	Multiply and assign	`$x *= $y`	`$x =`
`/=`	Divide and assign quotient	`$x /= $y`	`$x =`
`%=`	Divide and assign modulus	`$x %= $y`	`$x =`

The following example will show you these assignment operators in action:

Example
Run this code »

```
<?php
```

```
$x = 10;
echo $x; // Outputs: 10

$x = 20;
$x += 30;
echo $x; // Outputs: 50

$x = 50;
$x -= 20;
echo $x; // Outputs: 30

$x = 5;
$x *= 25;
echo $x; // Outputs: 125

$x = 50;
$x /= 10;
echo $x; // Outputs: 5

$x = 100;
$x %= 15;
echo $x; // Outputs: 10
?>
```

PHP Comparison Operators

The comparison operators are used to compare two values in a Boolean fashion.

Operator	Name	Example	Result
==	Equal	`$x == $y`	True if $x is equal to $y
===	Identical	`$x === $y`	True if $x is equal to $y, and they are
!=	Not equal	`$x != $y`	True if $x is not equal to $y

`<>`	Not equal	`$x <> $y`	True if $x is not equal to $y
`!==`	Not identical	`$x !== $y`	True if $x is not equal to $y, or they a type
`<`	Less than	`$x < $y`	True if $x is less than $y
`>`	Greater than	`$x > $y`	True if $x is greater than $y
`>=`	Greater than or equal to	`$x >= $y`	True if $x is greater than or equal to $
`<=`	Less than or equal to	`$x <= $y`	True if $x is less than or equal to $y

The following example will show you these comparison operators in action:

Example
Run this code »

```
<?php
$x = 25;
$y = 35;
$z = "25";
var_dump($x == $z);   // Outputs: boolean true
var_dump($x === $z);  // Outputs: boolean false
var_dump($x != $y);   // Outputs: boolean true
var_dump($x !== $z);  // Outputs: boolean true
var_dump($x < $y);    // Outputs: boolean true
var_dump($x > $y);    // Outputs: boolean false
var_dump($x <= $y);   // Outputs: boolean true
var_dump($x >= $y);   // Outputs: boolean false
?>
```

PHP Incrementing and Decrementing Operators

The increment/decrement operators are used to increment/decrement a variable's value.

Operator	Name	Effect
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

The following example will show you these increment and decrement operators in action:

Example
Run this code »

```
<?php
$x = 10;
echo ++$x; // Outputs: 11
echo $x;   // Outputs: 11

$x = 10;
echo $x++; // Outputs: 10
echo $x;   // Outputs: 11

$x = 10;
echo --$x; // Outputs: 9
echo $x;   // Outputs: 9

$x = 10;
echo $x--; // Outputs: 10
echo $x;   // Outputs: 9
```

```
?>
```

PHP Logical Operators

The logical operators are typically used to combine conditional statements.

Operator	Name	Example	Result
`and`	And	`$x and $y`	True if both $x and $y are true
`or`	Or	`$x or $y`	True if either $x or $y is true
`xor`	Xor	`$x xor $y`	True if either $x or $y is true, but not both
`&&`	And	`$x && $y`	True if both $x and $y are true
`||`	Or	`$x || $y`	True if either $x or $y is true
`!`	Not	`!$x`	True if $x is not true

The following example will show you these logical operators in action:

Example
Run this code »

```
<?php
$year = 2014;
// Leap years are divisible by 400 or by 4 but not
100
if(($year % 400 == 0) || (($year % 100 != 0) &&
($year % 4 == 0))){
    echo "$year is a leap year.";
} else{
    echo "$year is not a leap year.";
}
?>
```

PHP String Operators

There are two operators which are specifically designed for strings.

Operator	Description	Example	Result
`.`	Concatenation	`$str1 . $str2`	Concatenation of $
`.=`	Concatenation assignment	`$str1 .= $str2`	Appends the $str2

The following example will show you these string operators in action:

Example
Run this code »

```
<?php
$x = "Hello";
$y = " World!";
echo $x . $y; // Outputs: Hello World!

$x .= $y;
echo $x; // Outputs: Hello World!
?>
```

PHP Array Operators

The array operators are used to compare arrays:

Operator	Name	Example	Result
+	Union	`$x + $y`	Union of $x and $y
==	Equality	`$x == $y`	True if $x and $y have the same key/value pairs
===	Identity	`$x === $y`	True if $x and $y have the same key/value pairs in the same types
!=	Inequality	`$x != $y`	True if $x is not equal to $y
<>	Inequality	`$x <> $y`	True if $x is not equal to $y
!==	Non-identity	`$x !== $y`	True if $x is not identical to $y

The following example will show you these array operators in action:

Example
Run this code »

```
<?php
$x = array("a" => "Red", "b" => "Green", "c" =>
"Blue");
$y = array("u" => "Yellow", "v" => "Orange", "w" =>
"Pink");
$z = $x + $y; // Union of $x and $y
var_dump($z);
var_dump($x == $y);    // Outputs: boolean false
var_dump($x === $y);   // Outputs: boolean false
var_dump($x != $y);    // Outputs: boolean true
var_dump($x <> $y);    // Outputs: boolean true
var_dump($x !== $y);   // Outputs: boolean true
?>
```

PHP Spaceship Operator PHP 7

PHP 7 introduces a new spaceship operator (`<=>`) which can be used for comparing two expressions. It is also known as combined comparison operator.

The spaceship operator returns `0` if both operands are equal, `1` if the left is greater, and `-1` if the right is greater. It basically provides three-way comparison as shown in the following table:

Operator	<=> Equivalent
`$x < $y`	`($x <=> $y) === -1`
`$x <= $y`	`($x <=> $y) === -1 \|\| ($x <=> $y) === 0`
`$x == $y`	`($x <=> $y) === 0`
`$x != $y`	`($x <=> $y) !== 0`
`$x >= $y`	`($x <=> $y) === 1 \|\| ($x <=> $y) === 0`
`$x > $y`	`($x <=> $y) === 1`

The following example will show you how spaceship operator actually works:

Example

Run this code »

```php
<?php
// Comparing Integers
echo 1 <=> 1; // Outputs: 0
echo 1 <=> 2; // Outputs: -1
echo 2 <=> 1; // Outputs: 1

// Comparing Floats
echo 1.5 <=> 1.5; // Outputs: 0
echo 1.5 <=> 2.5; // Outputs: -1
echo 2.5 <=> 1.5; // Outputs: 1

// Comparing Strings
echo "x" <=> "x"; // Outputs: 0
echo "x" <=> "y"; // Outputs: -1
echo "y" <=> "x"; // Outputs: 1
?>
```

PHP If...Else Statements

In this tutorial you'll learn how to write decision-making code using if...else...elseif statements in PHP.

PHP Conditional Statements

Like most programming languages, PHP also allows you to write code that perform different actions based on the results of a logical or comparative test conditions at run time. This means, you can create test conditions in the form of expressions that evaluates to either `true` or `false` and based on these results you can perform certain actions.

There are several statements in PHP that you can use to make decisions:

- The **if** statement
- The **if...else** statement
- The **if...elseif....else** statement
- The **switch...case** statement

We will explore each of these statements in the coming sections.

The `if` Statement

The *if* statement is used to execute a block of code only if the specified condition evaluates to true. This is the simplest PHP's conditional statements and can be written like:

```
if(condition){
    // Code to be executed
}
```

The following example will output "Have a nice weekend!" if the current day is Friday:

Example

Run this code »

```
<?php
$d = date("D");
if($d == "Fri"){
    echo "Have a nice weekend!";
}
?>
```

The `if...else` Statement

You can enhance the decision making process by providing an alternative choice through adding an *else* statement to the *if* statement. The *if...else* statement allows you to execute one block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false. It can be written, like this:

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

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!"

Example
Run this code »

```
<?php
$d = date("D");
if($d == "Fri"){
    echo "Have a nice weekend!";
} else{
    echo "Have a nice day!";
}
?>
```

The `if...elseif...else` Statement

The *if...elseif...else* a special statement that is used to combine multiple *if...else* statements.

```
if(condition1){
    // Code to be executed if condition1 is true
} elseif(condition2){
    // Code to be executed if the condition1 is false and condition2 is true
} else{
    // Code to be executed if both condition1 and condition2 are false
}
```

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday, otherwise it will output "Have a nice day!"

Example
Run this code »

```
<?php
$d = date("D");
if($d == "Fri"){
    echo "Have a nice weekend!";
} elseif($d == "Sun"){
    echo "Have a nice Sunday!";
} else{
    echo "Have a nice day!";
}
?>
```

You will learn about PHP switch-case statement in the next chapter.

The Ternary Operator

The ternary operator provides a shorthand way of writing the *if...else* statements. The ternary operator is represented by the question mark (`?`) symbol and it takes three operands: a condition to check, a result for `true`, and a result for `false`.

To understand how this operator works, consider the following examples:

Example
Run this code »

```
<?php
if($age < 18){
    echo 'Child'; // Display Child if age is less
than 18
} else{
    echo 'Adult'; // Display Adult if age is greater
than or equal to 18
}
?>
```

Using the ternary operator the same code could be written in a more compact way:

Example
Run this code »

```
<?php echo ($age < 18) ? 'Child' : 'Adult'; ?>
```

The ternary operator in the example above selects the value on the left of the colon (i.e. 'Child') if the condition evaluates to true (i.e. if `$age` is less than 18), and selects the value on the right of the colon (i.e. 'Adult') if the condition evaluates to false.

Tip: Code written using the ternary operator can be hard to read. However, it provides a great way to write compact if-else statements.

The Null Coalescing Operator PHP 7

PHP 7 introduces a new null coalescing operator (`??`) which you can use as a shorthand where you need to use a ternary operator in conjunction with `isset()` function.

To uderstand this in a better way consider the following line of code. It fetches the value of `$_GET['name']`, if it does not exist or `NULL`, it returns 'anonymous'.

Example
Run this code »

```
<?php
$name = isset($_GET['name']) ? $_GET['name'] :
'anonymous';
?>
```

Using the null coalescing operator the same code could be written as:

Example
Run this code »

```
<?php
$name = $_GET['name'] ?? 'anonymous';
?>
```

As you can see the later syntax is more compact and easy to write.

PHP Switch...Case Statements

In this tutorial you will learn how to use the switch-case statement to test or evaluate an expression with different values in PHP.

PHP If...Else Vs Switch...Case

The switch-case statement is an alternative to the if-elseif-else statement, which does almost the same thing. The switch-case statement tests a variable against a series of values until it finds a match, and then executes the block of code corresponding to that match.

```
switch(n){
    case label1:
        // Code to be executed if n=label1
        break;
    case label2:
        // Code to be executed if n=label2
        break;
    ...
    default:
        // Code to be executed if n is different from all
labels
}
```

Consider the following example, which display a different message for each day.

Example
Run this code »

```
<?php
$today = date("D");
switch($today){
    case "Mon":
        echo "Today is Monday. Clean your house.";
        break;
    case "Tue":
        echo "Today is Tuesday. Buy some food.";
        break;
    case "Wed":
        echo "Today is Wednesday. Visit a doctor.";
        break;
    case "Thu":
        echo "Today is Thursday. Repair your car.";
        break;
    case "Fri":
        echo "Today is Friday. Party tonight.";
        break;
    case "Sat":
        echo "Today is Saturday. Its movie time.";
        break;
    case "Sun":
        echo "Today is Sunday. Do some rest.";
        break;
    default:
        echo "No information available for that
day.";
        break;
}
?>
```

The `switch-case` statement differs from the `if-elseif-else` statement in one important way. The `switch` statement executes line by line (i.e. statement by statement) and once PHP finds a `case` statement that evaluates to true, it's not only executes the code corresponding to that case statement, but also executes all the subsequent `case` statements till the end of the `switch` block automatically.

To prevent this add a `break` statement to the end of each `case` block. The `break` statement tells PHP to break out of the `switch-case` statement block once it executes the code associated with the first true case.

PHP Arrays

In this tutorial you'll learn how to store multiple values in a single variable in PHP.

What is PHP Arrays

Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name. Let's suppose you want to store colors in your PHP script. Storing the colors one by one in a variable could look something like this:

Example
Run this code »

```php
<?php
$color1 = "Red";
$color2 = "Green";
$color3 = "Blue";
?>
```

But what, if you want to store the states or city names of a country in variables and this time this not just three may be hundred. It is quite hard, boring, and bad idea to store each city name in a separate variable. And here array comes into play.

Types of Arrays in PHP

There are three types of arrays that you can create. These are:

- **Indexed array** — An array with a numeric key.
- **Associative array** — An array where each key has its own specific value.
- **Multidimensional array** — An array containing one or more arrays within itself.

Indexed Arrays

An indexed or numeric array stores each array element with a numeric index. The following examples shows two ways of creating an indexed array, the easiest way is:

Example
Run this code »

```
<?php
// Define an indexed array
$colors = array("Red", "Green", "Blue");
?>
```

Note: In an indexed or numeric array, the indexes are automatically assigned and start with 0, and the values can be any data type.

This is equivalent to the following example, in which indexes are assigned manually:

Example
Run this code »

```
<?php
$colors[0] = "Red";
$colors[1] = "Green";
$colors[2] = "Blue";
?>
```

Associative Arrays

In an associative array, the keys assigned to values can be arbitrary and user defined strings. In the following example the array uses keys instead of index numbers:

Example

Run this code »

```
<?php
// Define an associative array
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);
?>
```

The following example is equivalent to the previous example, but shows a different way of creating associative arrays:

Example

Run this code »

```
<?php
$ages["Peter"] = "22";
$ages["Clark"] = "32";
$ages["John"] = "28";
?>
```

Multidimensional Arrays

The multidimensional array is an array in which each element can also be an array and each element in the sub-array can be an array or further contain array within itself and so on. An example of a multidimensional array will look something like this:

Example

Run this code »

```
<?php
// Define a multidimensional array
$contacts = array(
    array(
        "name" => "Peter Parker",
        "email" => "peterparker@mail.com",
    ),
    array(
        "name" => "Clark Kent",
        "email" => "clarkkent@mail.com",
    ),
    array(
        "name" => "Harry Potter",
        "email" => "harrypotter@mail.com",
    )
);
// Access nested value
echo "Peter Parker's Email-id is: " .
$contacts[0]["email"];
?>
```

Viewing Array Structure and Values

You can see the structure and values of any array by using one of two statements — `var_dump()` or `print_r()`.
The `print_r()` statement, however, gives somewhat less information. Consider the following example:

Example
Run this code »

```
<?php
// Define array
$cities = array("London", "Paris", "New York");

// Display the cities array
print_r($cities);
?>
```

The `print_r()` statement gives the following output:

```
Array ( [0] => London [1] => Paris [2] => New York )
```

This output shows the key and the value for each element in the array. To get more information, use the following statement:

Example
Run this code »

```php
<?php
// Define array
$cities = array("London", "Paris", "New York");

// Display the cities array
var_dump($cities);
?>
```

This `var_dump()` statement gives the following output:

```
array(3) { [0]=> string(6) "London" [1]=> string(5) "Paris"
[2]=> string(8) "New York" }
```

This output shows the data type of each element, such as a string of 6 characters, in addition to the key and value. In the next chapter you will learn how to sort array elements.

PHP Sorting Arrays

In this tutorial you will learn how to sort the elements or keys of an array in ascending or descending order in PHP.

PHP Functions For Sorting Arrays

In the previous chapter you've learnt the essentials of PHP arrays i.e. what arrays are, how to create them, how to view their structure, how to access their elements etc. You can do even more things with arrays like sorting the elements in any order you like.

PHP comes with a number of built-in functions designed specifically for sorting array elements in different ways like alphabetically or numerically in ascending or descending order. Here we'll explore some of these functions most commonly used for sorting arrays.

- `sort()` and `rsort()` — For sorting indexed arrays
- `asort()` and `arsort()` — For sorting associative arrays by value
- `ksort()` and `krsort()` — For sorting associative arrays by key

Sorting Indexed Arrays in Ascending Order

The `sort()` function is used for sorting the elements of the indexed array in ascending order (alphabetically for letters and numerically for numbers).

Example
Run this code »

```
<?php
// Define array
$colors = array("Red", "Green", "Blue", "Yellow");

// Sorting and printing array
sort($colors);
print_r($colors);
?>
```

This `print_r()` statement gives the following output:

```
Array ( [0] => Blue [1] => Green [2] => Red [3] => Yellow )
```

Similarly you can sort the numeric elements of the array in ascending order.

Example
Run this code »

```
<?php
// Define array
$numbers = array(1, 2, 2.5, 4, 7, 10);

// Sorting and printing array
sort($numbers);
print_r($numbers);
?>
```

This `print_r()` statement gives the following output:

```
Array ( [0] => 1 [1] => 2 [2] => 2.5 [3] => 4 [4] => 7 [5] => 10 )
```

Sorting Indexed Arrays in Descending Order

The `rsort()` function is used for sorting the elements of the indexed array in descending order (alphabetically for letters and numerically for numbers).

Example
Run this code »

```
<?php
// Define array
$colors = array("Red", "Green", "Blue", "Yellow");

// Sorting and printing array
rsort($colors);
print_r($colors);
?>
```

This `print_r()` statement gives the following output:

Array ([0] => Yellow [1] => Red [2] => Green [3] => Blue)

Similarly you can sort the numeric elements of the array in descending order.

Example
Run this code »

```php
<?php
// Define array
$numbers = array(1, 2, 2.5, 4, 7, 10);

// Sorting and printing array
rsort($numbers);
print_r($numbers);
?>
```

This `print_r()` statement gives the following output:

Array ([0] => 10 [1] => 7 [2] => 4 [3] => 2.5 [4] => 2 [5] => 1)

Sorting Associative Arrays in Ascending Order By Value

The `asort()` function sorts the elements of an associative array in ascending order according to the value. It works just like `sort()`, but it preserves the association between keys and its values while sorting.

Example
Run this code »

```php
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45,
"Clark"=>35);
```

```
// Sorting array by value and print
asort($age);
print_r($age);
?>
```

This `print_r()` statement gives the following output:

```
Array ( [Harry] => 14 [Peter] => 20 [Clark] => 35 [John] => 45 )
```

Sorting Associative Arrays in Descending Order By Value

The `arsort()` function sorts the elements of an associative array in descending order according to the value. It works just like `rsort()`, but it preserves the association between keys and its values while sorting.

Example

Run this code »

```
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45,
"Clark"=>35);

// Sorting array by value and print
arsort($age);
print_r($age);
?>
```

This `print_r()` statement gives the following output:

```
Array ( [John] => 45 [Clark] => 35 [Peter] => 20 [Harry] => 14 )
```

Sorting Associative Arrays in Ascending Order By Key

The `ksort()` function sorts the elements of an associative array in ascending order by their keys. It preserves the association between keys and its values while sorting, same as `asort()` function.

Example
Run this code »

```
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45,
"Clark"=>35);

// Sorting array by key and print
ksort($age);
print_r($age);
?>
```

This `print_r()` statement gives the following output:

Array ([Clark] => 35 [Harry] => 14 [John] => 45 [Peter] => 20)

Sorting Associative Arrays in Descending Order By Key

The `krsort()` function sorts the elements of an associative array in descending order by their keys. It preserves the association between keys and its values while sorting, same as `arsort()` function.

Example
Run this code »

```php
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45,
"Clark"=>35);

// Sorting array by key and print
krsort($age);
print_r($age);
?>
```

This `print_r()` statement gives the following output:

```
Array ( [Peter] => 20 [John] => 45 [Harry] => 14 [Clark] => 35 )
```

PHP Loops

In this tutorial you will learn how to repeat a series of actions using loops in PHP.

Different Types of Loops in PHP

Loops are used to execute the same block of code again and again, as long as a certain condition is met. The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort. PHP supports four different types of loops.

- **while** — loops through a block of code as long as the condition specified evaluates to true.
- **do…while** — the block of code executed once and then condition is evaluated. If the condition is true the statement is repeated as long as the specified condition is true.
- **for** — loops through a block of code until the counter reaches a specified number.
- **foreach** — loops through a block of code for each element in an array.

You will also learn how to loop through the values of array using `foreach()` loop at the end of this chapter.
The `foreach()` loop work specifically with arrays.

PHP while Loop

The `while` statement will loops through a block of code as long as the condition specified in the `while` statement evaluate to true.

```
while(condition){
    // Code to be executed
}
```

The example below define a loop that starts with `$i=1`. The loop will continue to run as long as `$i` is less than or equal to 3. The `$i` will increase by 1 each time the loop runs:

Example
Run this code »

```
<?php
$i = 1;
while($i <= 3){
    $i++;
    echo "The number is " . $i . "<br>";
}
?>
```

PHP do...while Loop

The `do-while` loop is a variant of while loop, which evaluates the condition at the end of each loop iteration. With a `do-while` loop the block of code executed once, and then the condition is evaluated, if the condition is true, the statement is repeated as long as the specified condition evaluated to is true.

```
do{
    // Code to be executed
}
while(condition);
```

The following example define a loop that starts with `$i=1`. It will then increase `$i` with 1, and print the output. Then the condition is evaluated, and the loop will continue to run as long as `$i` is less than, or equal to 3.

Example
Run this code »

```
<?php
$i = 1;
do{
    $i++;
    echo "The number is " . $i . "<br>";
}
while($i <= 3);
?>
```

Difference Between while and do…while Loop

The `while` loop differs from the `do-while` loop in one important way — with a `while` loop, the condition to be evaluated is tested at the beginning of each loop iteration, so if the conditional expression evaluates to false, the loop will never be executed.

With a `do-while` loop, on the other hand, the loop will always be executed once, even if the conditional expression is false, because the condition is evaluated at the end of the loop iteration rather than the beginning.

PHP for Loop

The `for` loop repeats a block of code as long as a certain condition is met. It is typically used to execute a block of code for certain number of times.

```
for(initialization; condition; increment){
    // Code to be executed
}
```

The parameters of `for` loop have following meanings:

- `initialization` — it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.
- `condition` — in the beginning of each iteration, condition is evaluated. If it evaluates to `true`, the loop continues and the nested statements are executed. If it evaluates to `false`, the execution of the loop ends.
- `increment` — it updates the loop counter with a new value. It is evaluate at the end of each iteration.

The example below defines a loop that starts with `$i=1`. The loop will continued until `$i` is less than, or equal to 3. The variable `$i` will increase by 1 each time the loop runs:

Example
Run this code »

```
<?php
for($i=1; $i<=3; $i++){
    echo "The number is " . $i . "<br>";
}
?>
```

PHP foreach Loop

The `foreach` loop is used to iterate over arrays.

```
foreach($array as $value){
    // Code to be executed
}
```

The following example demonstrates a loop that will print the values of the given array:

Example

Run this code »

```
<?php
$colors = array("Red", "Green", "Blue");

// Loop through colors array
foreach($colors as $value){
    echo $value . "<br>";
}
?>
```

There is one more syntax of `foreach` loop, which is extension of the first.

```
foreach($array as $key => $value){
    // Code to be executed
}
```

Example

Run this code »

```
<?php
$superhero = array(
    "name" => "Peter Parker",
    "email" => "peterparker@mail.com",
    "age" => 18
);
```

```
// Loop through superhero array
foreach($superhero as $key => $value){
    echo $key . " : " . $value . "<br>";
}
?>
```

PHP Functions

In this tutorial you will learn how to create your own custom functions in PHP.

PHP Built-in Functions

A function is a self-contained block of code that performs a specific task.

PHP has a huge collection of internal or built-in functions that you can call directly within your PHP scripts to perform a specific task, like `gettype()`, `print_r()`, `var_dump`, etc.

Please check out PHP reference section for a complete list of useful PHP built-in functions.

PHP User-Defined Functions

In addition to the built-in functions, PHP also allows you to define your own functions. It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately form main program. Here are some advantages of using functions:

- **Functions reduces the repetition of code within a program** — Function allows you to extract commonly used block of code into a single component. Now you can perform the same task by calling this function wherever you want within your script without having to copy and paste the same block of code again and again.

- **Functions makes the code much easier to maintain** — Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.
- **Functions makes it easier to eliminate the errors** — When the program is subdivided into functions, if any error occur you know exactly what function causing the error and where to find it. Therefore, fixing errors becomes much easier.
- **Functions can be reused in other application** — Because a function is separated from the rest of the script, it's easy to reuse the same function in other applications just by including the php file containing those functions.

The following section will show you how easily you can define your own function in PHP.

Creating and Invoking Functions

The basic syntax of creating a custom function can be give with:

```
function functionName(){
    // Code to be executed
}
```

The declaration of a user-defined function start with the word `function`, followed by the name of the function you want to create followed by parentheses i.e. `()` and finally place your function's code between curly brackets `{}`.

This is a simple example of an user-defined function, that display today's date:

Example
Run this code »

```
<?php
// Defining function
function whatIsToday(){
    echo "Today is " . date('l', mktime());
}
// Calling function
whatIsToday();
?>
```

Note: A function name must start with a letter or underscore character not with a number, optionally followed by the more letters, numbers, or underscore characters. Function names are case-insensitive.

Functions with Parameters

You can specify parameters when you define your function to accept input values at run time. The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

```
function myFunc($oneParameter, $anotherParameter){
    // Code to be executed
}
```

You can define as many parameters as you like. However for each parameter you specify, a corresponding argument needs to be passed to the function when it is called.

The `getSum()` function in following example takes two integer values as arguments, simply add them together and then display the result in the browser.

Example
Run this code »

```
<?php
// Defining function
function getSum($num1, $num2){
  $sum = $num1 + $num2;
  echo "Sum of the two numbers $num1 and $num2 is :
$sum";
}

// Calling function
getSum(10, 20);
?>
```

The output of the above code will be:

```
Sum of the two numbers 10 and 20 is : 30
```

Tip: An argument is a value that you pass to a function, and a parameter is the variable within the function that receives the argument. However, in common usage these terms are interchangeable i.e. an argument is a parameter is an argument.

Functions with Optional Parameters and Default Values

You can also create functions with optional parameters — just insert the parameter name, followed by an equals (`=`) sign, followed by a default value, like this.

Example
Run this code »

```
<?php
// Defining function
```

```
function customFont($font, $size=1.5){
    echo "<p style=\"font-family: $font; font-size:
{$size}em;\">Hello, world!</p>";
}

// Calling function
customFont("Arial", 2);
customFont("Times", 3);
customFont("Courier");
?>
```

As you can see the third call to `customFont()` doesn't include the second argument. This causes PHP engine to use the default value for the `$size` parameter which is 1.5.

Returning Values from a Function

A function can return a value back to the script that called the function using the return statement. The value may be of any type, including arrays and objects.

Example
Run this code »

```
<?php
// Defining function
function getSum($num1, $num2){
    $total = $num1 + $num2;
    return $total;
}

// Printing returned value
echo getSum(5, 10); // Outputs: 15
?>
```

A function can not return multiple values. However, you can obtain similar results by returning an array, as demonstrated in the following example.

Example
Run this code »

```
<?php
// Defining function
function divideNumbers($dividend, $divisor){
    $quotient = $dividend / $divisor;
    $array = array($dividend, $divisor, $quotient);
    return $array;
}

// Assign variables as if they were an array
list($dividend, $divisor, $quotient) =
divideNumbers(10, 2);
echo $dividend;  // Outputs: 10
echo $divisor;   // Outputs: 2
echo $quotient;  // Outputs: 5
?>
```

Passing Arguments to a Function by Reference

In PHP there are two ways you can pass arguments to a function: *by value* and *by reference*. By default, function arguments are passed by value so that if the value of the argument within the function is changed, it does not get affected outside of the function. However, to allow a function to modify its arguments, they must be passed by reference.

Passing an argument by reference is done by prepending an ampersand (`&`) to the argument name in the function definition, as shown in the example below:

Example
Run this code »

```php
<?php
/* Defining a function that multiply a number
by itself and return the new value */
function selfMultiply(&$number){
    $number *= $number;
    return $number;
}

$mynum = 5;
echo $mynum; // Outputs: 5

selfMultiply($mynum);
echo $mynum; // Outputs: 25
?>
```

Understanding the Variable Scope

However, you can declare the variables anywhere in a PHP script. But, the location of the declaration determines the extent of a variable's visibility within the PHP program i.e. where the variable can be used or accessed. This accessibility is known as *variable scope*.

By default, variables declared within a function are local and they cannot be viewed or manipulated from outside of that function, as demonstrated in the example below:

Example

Run this code »

```php
<?php
// Defining function
function test(){
    $greet = "Hello World!";
    echo $greet;
}
```

```
test(); // Outputs: Hello World!

echo $greet; // Generate undefined variable error
?>
```

Similarly, if you try to access or import an outside variable inside the function, you'll get an undefined variable error, as shown in the following example:

Example
Run this code »

```
<?php
$greet = "Hello World!";

// Defining function
function test(){
    echo $greet;
}

test();  // Generate undefined variable error

echo $greet; // Outputs: Hello World!
?>
```

As you can see in the above examples the variable declared inside the function is not accessible from outside, likewise the variable declared outside of the function is not accessible inside of the function. This separation reduces the chances of variables within a function getting affected by the variables in the main program.

Tip: It is possible to reuse the same name for a variable in different functions, since local variables are only recognized by the function in which they are declared.

The global Keyword

There may be a situation when you need to import a variable from the main program into a function, or vice versa. In such cases, you can use the `global` keyword before the variables inside a function. This keyword turns the variable into a global variable, making it visible or accessible both inside and outside the function, as show in the example below:

Example
Run this code »

```php
<?php
$greet = "Hello World!";

// Defining function
function test(){
    global $greet;
    echo $greet;
}

test(); // Outpus: Hello World!
echo $greet; // Outpus: Hello World!

// Assign a new value to variable
$greet = "Goodbye";

test(); // Outputs: Goodbye
echo $greet; // Outputs: Goodbye
?>
```

You will learn more about visibility and access control in PHP classes and objects chapter.

Creating Recursive Functions

A recursive function is a function that calls itself again and again until a condition is satisfied. Recursive functions are often used to solve complex mathematical calculations, or to process deeply nested structures e.g., printing all the elements of a deeply nested array.

The following example demonstrates how a recursive function works.

Example
Run this code »

```
<?php
// Defining recursive function
function printValues($arr) {
    global $count;
    global $items;

    // Check input is an array
    if(!is_array($arr)){
        die("ERROR: Input is not an array");
    }

    /*
    Loop through array, if value is itself an array
recursively call the
    function else add the value found to the output
items array,
    and increment counter by 1 for each value found
    */
    foreach($arr as $a){
        if(is_array($a)){
            printValues($a);
        } else{
            $items[] = $a;
            $count++;
        }
    }

    // Return total count and values found in array
    return array('total' => $count, 'values' =>
$items);
}

// Define nested array
$species = array(
    "birds" => array(
        "Eagle",
        "Parrot",
        "Swan"
    ),
    "mammals" => array(
        "Human",
        "cat" => array(
            "Lion",
            "Tiger",
            "Jaguar"
        ),
        "Elephant",
        "Monkey"
    ),
    "reptiles" => array(
        "snake" => array(
            "Cobra" => array(
                "King Cobra",
                "Egyptian cobra"
            ),
            "Viper",
            "Anaconda"
        ),
        "Crocodile",
        "Dinosaur" => array(
            "T-rex",
            "Alamosaurus"
        )
    )
);

// Count and print values in nested array
$result = printValues($species);
echo $result['total'] . ' value(s) found: ';
echo implode(', ', $result['values']);
?>
```

Note: Be careful while creating recursive functions, because if code is written improperly it may result in an infinite loop of function calling.

PHP Math Operations

In this tutorial you will learn how to perform mathematical operations in PHP.

Performing Math Operations

PHP has several built-in functions that help you perform anything from simple additions or subtraction to advanced calculations. You've already seen how to perform basic mathematical operations in PHP operators chapter. Let's check out one more example:

Example
Run this code »

```
<?php
echo 7 + 3; // Outputs: 10
echo 7 - 2; // Outputs: 5
echo 7 * 2; // Outputs: 14
echo 7 / 2; // Outputs: 3.5
echo 7 % 2; // Outputs: 1
?>
```

Every math operation has a certain precedence level; generally multiplication and division are performed before addition and subtraction. However, parentheses can alter this precedence; expressions enclosed within parentheses are always evaluated first, regardless of the operation's precedence level, as demonstrated in the following example:

Example
Run this code »

```
<?php
echo 5 + 4 * 10;          // Outputs: 45
echo (5 + 4) * 10;        // Outputs: 90
echo 5 + 4 * 10 / 2;      // Outputs: 25
echo 8 * 10 / 4 - 2;      // Outputs: 18
echo 8 * 10 / (4 - 2);    // Outputs: 40
echo 8 + 10 / 4 - 2;      // Outputs: 8.5
echo (8 + 10) / (4 - 2); // Outputs: 9
?>
```

In the following section we're going to look at some built-in PHP functions that are most frequently used in performing mathematical operations.

Find the Absolute Value of a Number

The absolute value of an integer or a float can be found with the `abs()` function, as demonstrated in the following example:

Example
Run this code »

```
<?php
echo abs(5);    // Outputs: 5 (integer)
echo abs(-5);   // Outputs: 5 (integer)
echo abs(4.2);  // Outputs: 4.2 (double/float)
echo abs(-4.2); // Outputs: 4.2 (double/float)
?>
```

As you can see if the given number is negative, the valued returned is positive. But, if the number is positive, this function simply returns the number.

Round a Fractional Value Up or Down

The `ceil()` function can be used to round a fractional value up to the next highest integer value, whereas the `floor()` function can be used to round a fractional value down to the next lowest integer value, as demonstrated in the following example:

Example
Run this code »

```
<?php
// Round fractions up
echo ceil(4.2);     // Outputs: 5
echo ceil(9.99);    // Outputs: 10
echo ceil(-5.18);   // Outputs: -5

// Round fractions down
echo floor(4.2);     // Outputs: 4
echo floor(9.99);    // Outputs: 9
echo floor(-5.18);   // Outputs: -6
?>
```

Find the Square Root of a Number

You can use the `sqrt()` function to find the square root of a positive number. This function returns a special value `NAN` for negative numbers. Here's an example:

Example
Run this code »

```
<?php
echo sqrt(9);   // Outputs: 3
echo sqrt(25);  // Outputs: 5
echo sqrt(10);  // Outputs: 3.1622776601684
echo sqrt(-16); // Outputs: NAN
?>
```

Generate a Random Number

The `rand()` function can be used to generate a random number. You can optionally specify a range by passing the min, max arguments, as shown in the following example:

Example
Run this code »

```
<?php
// Generate some random numbers
echo rand() . "<br>";
echo rand() . "<br>";

// Generate some random numbers between 1 and 10
(inclusive)
echo rand(1, 10) . "<br>";
echo rand(1, 10) . "<br>";
?>
```

If `rand()` function is called without the optional **`min`**, **`max`** arguments, it returns a pseudo-random number between `0` and `getrandmax()`. The `getrandmax()` function show the largest possible random value, which is only 32767 on Windows platform. So, if you require a range larger than 32767, you may simply specify
the **`min`** and **`max`** arguments.

Convert Decimal Numbers to Binary and Vice Versa

The `decbin()` function is used to convert a decimal number into binary number. Whereas its counterpart the `bindec()` function converts a number from binary to decimal.

Example
Run this code »

```php
<?php
// Convert Decimal to Binary
echo decbin(2);     // Outputs: 10
echo decbin(12);    // Outputs: 1100
echo decbin(100);   // Outputs: 1100100

// Convert Binary to Decimal
echo bindec(10);        // Outputs: 2
echo bindec(1100);      // Outputs: 12
echo bindec(1100100);   // Outputs: 100
?>
```

Convert Decimal Numbers to Hexadecimal and Vice Versa

The `dechex()` function is used to convert a decimal number into hexadecimal representation. Whereas, the `hexdec()` function is used to converts a hexadecimal string to a decimal number.

Example

Run this code »

```php
<?php
// Convert decimal to hexadecimal
echo dechex(255);   // Outputs: ff
echo dechex(196);   // Outputs: c4
echo dechex(0);     // Outputs: 0

// Convert hexadecimal to decimal
echo hexdec('ff');   // Outputs: 255
echo hexdec('c4');   // Outputs: 196
echo hexdec(0);      // Outputs: 0
?>
```

Convert Decimal Numbers to Octal and Vice Versa

The `decoct()` function is used to convert a decimal number into octal representation. Whereas, the `octdec()` function is used to converts a octal number to a decimal number.

Example
Run this code »

```php
<?php
// Convert decimal to octal
echo decoct(12);    // Outputs: 14
echo decoct(256);   // Outputs: 400
echo decoct(77);    // Outputs: 115

// Convert octal to decimal
echo octdec('14');    // Outputs: 12
echo octdec('400');   // Outputs: 256
echo octdec('115');   // Outputs: 77
?>
```

Convert a Number from One Base System to Another

The `base_convert()` function can be used to convert a number from one base system to other. For example, you can convert decimal (*base 10*) to binary (*base 2*), hexadecimal (*base 16*) to octal (*base 8*), octal to hexadecimal, hexadecimal to decimal, and so on.

This function accepts three parameters: the number to convert, the base it's currently in, and the base it's to be converted to. The basic syntax is as follows:

```
base_convert(number, frombase, tobase);
```

Here, the number can be either an integer or a string representing an integer. Both **`frombase`** and **`tobase`** have to be between 2 and 36, inclusive. Digits in numbers with a base higher than 10 will be represented with the letters a-z, where a means 10, b means 11 and z means 35. Here's a simple example to show how this function works:

Example

Run this code »

```
<?php
// Convert decimal to binary
echo base_convert('12', 10, 2);  // Outputs: 1100
// Convert binary to decimal
echo base_convert('1100', 2, 10);  // Outputs: 12

// Convert decimal to hexadecimal
echo base_convert('10889592', 10, 16);  // Outputs:
a62978
// Convert hexadecimal to decimal
echo base_convert('a62978', 16, 10);  // Outputs:
10889592

// Convert decimal to octal
echo base_convert('82', 10, 8);  // Outputs: 122
// Convert octal to decimal
echo base_convert('122', 8, 10);  // Outputs: 82

// Convert hexadecimal to octal
echo base_convert('c2c6a8', 16, 8);  // Outputs:
60543250
// Convert octal to hexadecimal
echo base_convert('60543250', 8, 16);  // Outputs:
c2c6a8

// Convert octal to binary
echo base_convert('42', 8, 2);  // Outputs: 100010
// Convert binary to octal
echo base_convert('100010', 2, 8);  // Outputs: 42
```

```
// Convert hexadecimal to binary
echo base_convert('abc', 16, 2);  // Outputs:
101010111100
// Convert binary to hexadecimal
echo base_convert('101010111100', 2, 16);  //
Outputs: abc
?>
```

Note: The `base_convert()` function will always return a string value. If the returned value is in base 10 the resulting string can be used as a numeric string in calculations and PHP will convert it to a number when the calculation is performed.

PHP GET and POST

In this tutorial you will learn how to send information to the server using HTTP GET and POST methods and retrieve them using PHP.

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
Download

```
<!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"`.

Example
Download

```
<!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.

Example
Download

```
<!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>
```

You will learn more about PHP cookies and form handling in advanced section.

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

PHP Date and Time

In this tutorial you will learn how to extract or format the date and time in PHP.

The PHP `Date()` Function

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

The computer stores dates and times in a format called UNIX Timestamp, which measures time as a number of seconds since the beginning of the Unix epoch (midnight Greenwich Mean Time on January 1, 1970 i.e. January 1, 1970 00:00:00 GMT).

Since this is an impractical format for humans to read, PHP converts a timestamp to a format that is readable to humans and dates from your notation into a timestamp the computer understands. The syntax of the PHP `date()` function can be given with.

```
date(format, timestamp)
```

The *format* parameter in the `date()` function is required which specifies the format of returned date and time. However the *timestamp* is an optional parameter, if not included then current date and time will be used. The following statement displays today's date:

Example
Run this code »

```
<?php
$today = date("d/m/Y");
echo $today;
?>
```

Note: The PHP `date()` function return the current date and time according to the built-in clock of the web server on which the script has been executed.

Formatting the Dates and Times with PHP

The format parameter of the `date()` function is in fact a string that can contain multiple characters allowing you to generate a date string containing various components of the date and time, like day of the week, AM or PM, etc. Here are some the date-related formatting characters that are commonly used in format string:

- d - Represent day of the month; two digits with leading zeros (01 or 31)
- D - Represent day of the week in text as an abbreviation (Mon to Sun)
- m - Represent month in numbers with leading zeros (01 or 12)
- M - Represent month in text, abbreviated (Jan to Dec)

- y - Represent year in two digits (08 or 14)
- Y - Represent year in four digits (2008 or 2014)

The parts of the date can be separated by inserting other characters, like hyphens (`-`), dots (`.`), slashes (`/`), or spaces to add additional visual formatting.

Example
Run this code »

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

Tip: You can use the PHP `date()` function to automatically update the copyright duration on your website, like: `Copyright &copy; 2010-<?php echo date("Y")?>`.

Similarly you can use the following characters to format the time string:

- h - Represent hour in 12-hour format with leading zeros (01 to 12)
- H - Represent hour in in 24-hour format with leading zeros (00 to 23)
- i - Represent minutes with leading zeros (00 to 59)
- s - Represent seconds with leading zeros (00 to 59)
- a - Represent lowercase ante meridiem and post meridiem (am or pm)
- A - Represent uppercase Ante meridiem and Post meridiem (AM or PM)

The PHP code in the following example displays the date in different formats:

Example
Run this code »

```
<?php
echo date("h:i:s") . "<br>";
echo date("F d, Y h:i:s A") . "<br>";
echo date("h:i a");
?>
```

The PHP `time()` Function

The `time()` function is used to get the current time as a Unix timestamp (the number of seconds since the beginning of the Unix epoch: January 1 1970 00:00:00 GMT).

Example
Run this code »

```
<?php
// Executed at March 05, 2014 07:19:18
$timestamp = time();
echo($timestamp);
?>
```

The above example produce the following output.

```
1394003958
```

We can convert this timestamp to a human readable date through passing it to the previously introduce `date()` function.

Example
Run this code »

```
<?php
$timestamp = 1394003958;
echo(date("F d, Y h:i:s", $timestamp));
```

```
?>
```

The above example produce the following output.

March 05, 2014 07:19:18

The PHP `mktime()` Function

The `mktime()` function is used to create the timestamp based on a specific date and time. If no date and time is provided, the timestamp for the current date and time is returned.

The syntax of the `mktime()` function can be given with:

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

The following example displays the timestamp corresponding to 3:20:12 pm on May 10, 2014:

Example
Run this code »

```
<?php
// Create the timestamp for a particular date
echo mktime(15, 20, 12, 5, 10, 2014);
?>
```

The above example produce the following output.

1399735212

Note: You can leave out as many arguments as you like, and the value corresponding to the current time will be used instead. If you omit all the arguments, the `mktime()` function will return the UNIX timestamp corresponding to the current date and time, just like `time()`.

The `mktime()` function can be used to find the weekday name corresponding to a particular date. To do this, simply use the 'l' (lowercase 'L') character with your timestamp, as in the following example, which displays the day that falls on April 1, 2014:

Example
Run this code »

```
<?php
// Get the weekday name of a particular date
echo date('l', mktime(0, 0, 0, 4, 1, 2014));
?>
```

The above example produce the following output.

```
Tuesday
```

The `mktime()` function can also be used to find a particular date in future after a specific time period. As in the following example, which displays the date which falls on after 30 month from the current date?

Example
Run this code »

```
<?php
// Executed at March 05, 2014
$futureDate = mktime(0, 0, 0, date("m")+30,
date("d"), date("Y"));
echo date("d/m/Y", $futureDate);
?>
```

The above example produce the following output.

```
05/09/2016
```

PHP Include and Require Files

In this tutorial you will learn how to include and evaluate the files in PHP.

Including a PHP File into Another PHP File

The `include()` and `require()` statement allow you to include the code contained in a PHP file within another PHP file. Including a file produces the same result as copying the script from the file specified and pasted into the location where it is called.

You can save a lot of time and work through including files — Just store a block of code in a separate file and include it wherever you want using the `include()` and `require()` statements instead of typing the entire block of code multiple times. A typical example is including the header, footer and menu file within all the pages of a website.

The basic syntax of the `include()` and `require()` statement can be given with:

```
include("path/to/filename"); -Or-
 include "path/to/filename";
require("path/to/filename"); -Or-
 require "path/to/filename";
```

Tip: Like the `print` and `echo` statements, you can omit the parentheses while using the `include` and `require` statements as demonstrated above.

The following example will show you how to include the common header, footer and menu codes which are stored in separate 'header.php', 'footer.php' and 'menu.php' files respectively, within all the pages of your website. Using this technique you can update all pages of the website at once by making the changes to just one file, this saves a lot of repetitive work.

Example
Run this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Tutorial Republic</title>
</head>
<body>
<?php include "header.php"; ?>
<?php include "menu.php"; ?>
    <h1>Welcome to Our Website!</h1>
    <p>Here you will find lots of useful
information.</p>
<?php include "footer.php"; ?>
</body>
</html>
```

Difference Between include and require Statements

You might be thinking if we can include files using the `include()` statement then why we need `require()`. Typically the `require()` statement operates like `include()`.

The only difference is — the `include()` statement will only generate a PHP warning but allow script execution to continue if the file to be included can't be found, whereas the `require()` statement will generate a fatal error and stops the script execution.

Example
Run this code »

```
<?php require "my_variables.php"; ?>
<?php require "my_functions.php"; ?>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title><?php displayTitle($home_page); ?></title>
</head>
<body>
<?php include "header.php"; ?>
<?php include "menu.php"; ?>
    <h1>Welcome to Our Website!</h1>
    <p>Here you will find lots of useful
information.</p>
<?php include "footer.php"; ?>
</body>
</html>
```

Tip: It is recommended to use the `require()` statement if you're including the library files or files containing the functions and configuration variables that are essential for running your application, such as database configuration file.

The include_once and require_once Statements

If you accidentally include the same file (typically functions or classes files) more than one time within your code using the `include` or `require` statements, it may cause conflicts. To prevent this situation, PHP provides `include_once` and `require_once` statements. These statements behave in the same way as `include` and `require` statements with one exception.

The `include_once` and `require_once` statements will only include the file once even if asked to include it a second time i.e. if the specified file has already been included in a previous statement, the file is not included again. To better understand how it works, let's check out an example. Suppose we've a 'my_functions.php' file with the following code:

Example
Run this code »

```
<?php
function multiplySelf($var){
    $var *= $var; // multiply variable by itself
    echo $var;
}
?>
```

Here's is the PHP script within which we've included the 'my_functions.php' file.

Example
Run this code »

```
<?php
// Including file
require "my_functions.php";
// Calling the function
multiplySelf(2); // Output: 4
echo "<br>";

// Including file once again
require "my_functions.php";
// Calling the function
multiplySelf(5); // Doesn't execute
?>
```

When you run the above script, you will see the error message something like this: **"Fatal error: Cannot redeclare multiplySelf()"**. This occurs because the 'my_functions.php' included twice, this means the function `multiplySelf()` is defined twice, which caused PHP to stop script execution and generate fatal error. Now rewrite the above example with `require_once`.

Example
Run this code »

```
<?php
// Including file
require_once "my_functions.php";
// Calling the function
multiplySelf(2); // Output: 4
echo "<br>";

// Including file once again
require_once "my_functions.php";
// Calling the function
multiplySelf(5); // Output: 25
?>
```

As you can see, by using `require_once` instead of `require`, the script works as we expected.

PHP File System

In this tutorial you will learn how to create, access (or read) and manipulate files dynamically using the PHP's file system functions.

Working with Files in PHP

Since PHP is a server side programming language, it allows you to work with files and directories stored on the web server. In this tutorial you will learn how to create, access, and manipulate files on your web server using the PHP file system functions.

Opening a File with PHP `fopen()` Function

To work with a file you first need to open the file. The PHP `fopen()` function is used to open a file. The basic syntax of this function can be given with:

```
fopen(filename, mode)
```

The first parameter passed to `fopen()` specifies the name of the file you want to open, and the second parameter specifies in which mode the file should be opened. For example:

Example
Run this code »

```
<?php
$handle = fopen("data.txt", "r");
?>
```

The file may be opened in one of the following modes:

Modes	What it does
`r`	Open the file for reading only.
`r+`	Open the file for reading and writing.
`w`	Open the file for writing only and clears the contents of file. If the file does not exist, Pl create it.
`w+`	Open the file for reading and writing and clears the contents of file. If the file does not to create it.
`a`	Append. Opens the file for writing only. Preserves file content by writing to the end of not exist, PHP will attempt to create it.
`a+`	Read/Append. Opens the file for reading and writing. Preserves file content by writing If the file does not exist, PHP will attempt to create it.
`x`	Open the file for writing only. Return `FALSE` and generates an error if the file already e not exist, PHP will attempt to create it.
`x+`	Open the file for reading and writing; otherwise it has the same behavior as 'x'.

If you try to open a file that doesn't exist, PHP will generate a warning message. So, to avoid these error messages you should always implement a simple check whether a file or directory exists or not before trying to access it, with the PHP `file_exists()` function.

Example
Run this code »

```
<?php
$file = "data.txt";

// Check the existence of file
if(file_exists($file)){
    // Attempt to open the file
    $handle = fopen($file, "r");
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Tip: Operations on files and directories are prone to errors. So it's a good practice to implement some form of error checking so that if an error occurs your script will handle the error gracefully. See the tutorial on PHP error handling.

Closing a File with PHP `fclose()` Function

Once you've finished working with a file, it needs to be closed. The `fclose()` function is used to close the file, as shown in the following example:

Example
Run this code »

```
<?php
```

```
$file = "data.txt";

// Check the existence of file
if(file_exists($file)){
    // Open the file for reading
    $handle = fopen($file, "r") or die("ERROR: Cannot
open the file.");

    /* Some code to be executed */

    // Closing the file handle
    fclose($handle);
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Note: Although PHP automatically closes all open files when script terminates, but it's a good practice to close a file after performing all the operations.

Reading from Files with PHP `fread()` Function

Now that you have understood how to open and close files. In the following section you will learn how to read data from a file. PHP has several functions for reading data from a file. You can read from just one character to the entire file with a single operation.

Reading Fixed Number of Characters

The `fread()` function can be used to read a specified number of characters from a file. The basic syntax of this function can be given with.

```
fread(file handle, length in bytes)
```

This function takes two parameter — A file handle and the number of bytes to read. The following example reads 20 bytes from the "data.txt" file including spaces. Let's suppose the file "data.txt" contains a paragraph of text "The quick brown fox jumps over the lazy dog."

Example
Run this code »

```
<?php
$file = "data.txt";

// Check the existence of file
if(file_exists($file)){
    // Open the file for reading
    $handle = fopen($file, "r") or die("ERROR: Cannot
open the file.");

    // Read fixed number of bytes from the file
    $content = fread($handle, "20");

    // Closing the file handle
    fclose($handle);

    // Display the file content
    echo $content;
} else{
    echo "ERROR: File does not exist.";
}
?>
```

The above example will produce the following output:

```
The quick brown fox
```

Reading the Entire Contents of a File

The `fread()` function can be used in conjugation with the `filesize()` function to read the entire file at once. The `filesize()` function returns the size of the file in bytes.

Example
Run this code »

```
<?php
$file = "data.txt";

// Check the existence of file
if(file_exists($file)){
    // Open the file for reading
    $handle = fopen($file, "r") or die("ERROR: Cannot
open the file.");

    // Reading the entire file
    $content = fread($handle, filesize($file));

    // Closing the file handle
    fclose($handle);

    // Display the file content
    echo $content;
} else{
    echo "ERROR: File does not exist.";
}
?>
```

The above example will produce the following output:

The quick brown fox jumps over the lazy dog.

The easiest way to read the entire contents of a file in PHP is with the `readfile()` function. This function allows you to read the contents of a file without needing to open it. The following example will generate the same output as above example:

Example
Run this code »

```
<?php
$file = "data.txt";

// Check the existence of file
if(file_exists($file)){
    // Reads and outputs the entire file
    readfile($file) or die("ERROR: Cannot open the
file.");
} else{
    echo "ERROR: File does not exist.";
}
?>
```

The above example will produce the following output:

The quick brown fox jumps over the lazy dog.

Another way to read the whole contents of a file without needing to open it is with the `file_get_contents()` function. This function accepts the name and path to a file, and reads the entire file into a string variable. Here's an example:

Example
Run this code »

```
<?php
$file = "data.txt";

// Check the existence of file
if(file_exists($file)){
    // Reading the entire file into a string
    $content = file_get_contents($file) or
die("ERROR: Cannot open the file.");

    // Display the file content
    echo $content;
} else{
    echo "ERROR: File does not exist.";
}
?>
```

One more method of reading the whole data from a file is the PHP's `file()` function. It does a similar job to `file_get_contents()` function, but it returns the file contents as an array of lines, rather than a single string. Each element of the returned array corresponds to a line in the file.

To process the file data, you need to iterate over the array using a foreach loop. Here's an example, which reads a file into an array and then displays it using the loop:

Example
Run this code »

```php
<?php
$file = "data.txt";

// Check the existence of file
if(file_exists($file)){
    // Reading the entire file into an array
    $arr = file($file) or die("ERROR: Cannot open the 
file.");
    foreach($arr as $line){
        echo $line;
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Writing the Files Using PHP `fwrite()` Function

Similarly, you can write data to a file or append to an existing file using the PHP `fwrite()` function. The basic syntax of this function can be given with:

```
fwrite(file handle, string)
```

The `fwrite()` function takes two parameter — A file handle and the string of data that is to be written, as demonstrated in the following example:

Example
Run this code »

```
<?php
$file = "note.txt";

// String of data to be written
$data = "The quick brown fox jumps over the lazy
dog.";

// Open the file for writing
$handle = fopen($file, "w") or die("ERROR: Cannot
open the file.");

// Write data to the file
fwrite($handle, $data) or die ("ERROR: Cannot write
the file.");

// Closing the file handle
fclose($handle);

echo "Data written to the file successfully.";
?>
```

In the above example, if the "note.txt" file doesn't exist PHP will automatically create it and write the data. But, if the "note.txt" file already exist, PHP will erase the contents of this file, if it has any, before writing the new data, however if you just want to append the file and preserve existing contents just use the mode `a` instead of `w` in the above example.

An alternative way is using the `file_put_contents()` function. It is counterpart of `file_get_contents()` function and provides an easy method of writing the data to a file without needing to open it. This function accepts the name and path to a file together with the data to be written to the file. Here's an example:

Example
Run this code »

```
<?php
$file = "note.txt";

// String of data to be written
$data = "The quick brown fox jumps over the lazy
dog.";

// Write data to the file
file_put_contents($file, $data) or die("ERROR: Cannot
write the file.");

echo "Data written to the file successfully.";
?>
```

If the file specified in the `file_put_contents()` function already exists, PHP will overwrite it by default. If you would like to preserve the file's contents you can pass the special `FILE_APPEND` flag as a third parameter to the `file_put_contents()` function. It will simply append the new data to the file instead of overwitting it. Here's an example:

Example

Run this code »

```
<?php
$file = "note.txt";

// String of data to be written
$data = "The quick brown fox jumps over the lazy
dog.";

// Write data to the file
file_put_contents($file, $data, FILE_APPEND) or
die("ERROR: Cannot write the file.");

echo "Data written to the file successfully.";
?>
```

Renaming Files with PHP `rename()` Function

You can rename a file or directory using the PHP's `rename()` function, like this:

Example
Run this code »

```
<?php
$file = "file.txt";

// Check the existence of file
if(file_exists($file)){
    // Attempt to rename the file
    if(rename($file, "newfile.txt")){
        echo "File renamed successfully.";
    } else{
        echo "ERROR: File cannot be renamed.";
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Removing Files with PHP `unlink()` Function

You can delete files or directories using the PHP's `unlink()` function, like this:

Example
Run this code »

```
<?php
$file = "note.txt";

// Check the existence of file
```

```
if(file_exists($file)){
    // Attempt to delete the file
    if(unlink($file)){
        echo "File removed successfully.";
    } else{
        echo "ERROR: File cannot be removed.";
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

In the next chapter we will learn more about parsing directories or folders in PHP.

PHP Filesystem Functions

The following table provides the overview of some other useful PHP filesystem functions that can be used for reading and writing the files dynamically.

Function	Description
`fgetc()`	Reads a single character at a time.
`fgets()`	Reads a single line at a time.
`fgetcsv()`	Reads a line of comma-separated values.
`filetype()`	Returns the type of the file.
`feof()`	Checks whether the end of the file has been reached.
`is_file()`	Checks whether the file is a regular file.

Function	Description
`is_dir()`	Checks whether the file is a directory.
`is_executable()`	Checks whether the file is executable.
`realpath()`	Returns canonicalized absolute pathname.
`rmdir()`	Removes an empty directory.

Please check out the PHP filesystem reference for other useful PHP filesystem functions.

PHP Parsing Directories

In this tutorial you will learn how to process directories or folders using PHP.

Working with Directories in PHP

In the previous chapter you've learned how to work with files in PHP. Similarly, PHP also allows you to work with directories on the file system, for example, you can open a directory and read its contents, create or delete a directory, list all files in the directory, and so on.

Creating a New Directory

You can create a new and empty directory by calling the PHP `mkdir()` function with the path and name of the directory to be created, as shown in the example below:

Example
Run this code »

```
<?php
// The directory path
$dir = "testdir";

// Check the existence of directory
if(!file_exists($dir)){
    // Attempt to create directory
    if(mkdir($dir)){
        echo "Directory created successfully.";
    } else{
        echo "ERROR: Directory could not be
created.";
    }
} else{
    echo "ERROR: Directory already exists.";
}
?>
```

To make the `mkdir()` function work, the parent directories in the directory path parameter has to exist already, for example, if you specify the directory path as `testdir/subdir` than the `testdir` has to exist otherwise PHP will generate an error.

Copying Files from One Location to Another

You can copy a file from one location to another by calling PHP `copy()` function with the file's source and destination paths as arguments. If the destination file already exists it'll be overwritten. Here's an example which creates a copy of "example.txt" file inside backup folder.

Example
Run this code »

```
<?php
// Source file path
$file = "example.txt";

// Destination file path
$newfile = "backup/example.txt";

// Check the existence of file
if(file_exists($file)){
    // Attempt to copy file
    if(copy($file, $newfile)){
        echo "File copied successfully.";
    } else{
        echo "ERROR: File could not be copied.";
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

To make this example work, the target directory which is *backup* and the source file i.e. "example.txt" has to exist already; otherwise PHP will generate an error.

Listing All Files in a Directory

You can use the PHP `scandir()` function to list files and directories inside the specified path.

Now we're going to create a custom function that will recursively list all files in a directory using PHP. This script will be helpful if you're working with deeply nested directory structure.

Example

Run this code »

```
<?php
// Define a function to output files in a directory
function outputFiles($path){
    // Check directory exists or not
    if(file_exists($path) && is_dir($path)){
        // Scan the files in this directory
        $result = scandir($path);

        // Filter out the current (.) and parent (..)
directories
        $files = array_diff($result, array('.',
'..'));

        if(count($files) > 0){
            // Loop through retuned array
            foreach($files as $file){
                if(is_file("$path/$file")){
                    // Display filename
                    echo $file . "<br>";
                } else if(is_dir("$path/$file")){
                    // Recursively call the function
if directories found
                    outputFiles("$path/$file");
                }
            }
        } else{
            echo "ERROR: No files found in the
directory.";
        }
    } else {
        echo "ERROR: The directory does not exist.";
    }
}

// Call the function
outputFiles("mydir");
?>
```

Listing All Files of a Certain Type

While working on directory and file structure, sometimes you might need to find out certain types of files within the directory, for example, listing only `.text` or `.png` files, etc. You can do this easily with the PHP `glob()` function, which matches files based on the pattern.

The PHP code in the following example will search the *documents* directory and list all the files with `.text` extension. It will not search the subdirectories.

Example
Run this code »

```
<?php
/* Search the directory and loop through
returned array containing the matched files */
foreach(glob("documents/*.txt") as $file){
    echo basename($file) . " (size: " .
filesize($file) . " bytes)" . "<br>";
}
?>
```

The `glob()` function can also be used to find all the files within a directory or its subdirectories. The function defined in the following example will recursively list all files within a directory, just like we've done in previous example with the `scandir()` function.

Example
Run this code »

```
<?php
// Define a function to output files in a directory
function outputFiles($path){
    // Check directory exists or not
    if(file_exists($path) && is_dir($path)){
        // Search the files in this directory
        $files = glob($path ."/*");
        if(count($files) > 0){
            // Loop through retuned array
            foreach($files as $file){
                if(is_file("$file")){
                    // Display only filename
                    echo basename($file) . "<br>";
                } else if(is_dir("$file")){
                    // Recursively call the function
if directories found
                    outputFiles("$file");
                }
            }
        } else{
            echo "ERROR: No such file found in the
directory.";
        }
    } else {
        echo "ERROR: The directory does not exist.";
    }
}

// Call the function
outputFiles("mydir");
?>
```

PHP File Upload

In this tutorial you'll learn how to upload a file to the remote web server with PHP.

Uploading Files with PHP

In this tutorial we will learn how to upload files on remote server using a Simple HTML form and PHP. You can upload any kind of file like images, videos, ZIP files, Microsoft Office documents, PDFs, as well as executables files and a wide range of other file types.

Step 1: Creating an HTML form to upload the file

The following example will create a simple HTML form that can be used to upload files.

Example
Download

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>File Upload Form</title>
</head>
<body>
    <form action="upload-manager.php" method="post"
enctype="multipart/form-data">
        <h2>Upload File</h2>
        <label for="fileSelect">Filename:</label>
        <input type="file" name="photo"
id="fileSelect">
        <input type="submit" name="submit"
value="Upload">
        <p><strong>Note:</strong> Only .jpg, .jpeg,
.gif, .png formats allowed to a max size of 5 MB.</p>
    </form>
</body>
</html>
```

Note: In addition to a file-select field the upload form must use the HTTP post method and must contain an `enctype="multipart/form-data"` attribute. This attribute ensures that the form data is encoded as mulitpart MIME data — which is required for uploading the large quantities of binary data such as image, audio, video, etc.

Step 2: Processing the uploaded file

Here's the complete code of our "upload-manager.php" file. It will store the uploaded file in a "upload" folder on permanent basis as well as implement some basic security check like file type and file size to ensure that users upload the correct file type and within the allowed limit.

Example
Download

```
<?php
// Check if the form was submitted
if($_SERVER["REQUEST_METHOD"] == "POST"){
    // Check if file was uploaded without errors
    if(isset($_FILES["photo"]) &&
$_FILES["photo"]["error"] == 0){
        $allowed = array("jpg" => "image/jpg", "jpeg"
=> "image/jpeg", "gif" => "image/gif", "png" =>
"image/png");
        $filename = $_FILES["photo"]["name"];
        $filetype = $_FILES["photo"]["type"];
        $filesize = $_FILES["photo"]["size"];

        // Verify file extension
        $ext = pathinfo($filename,
PATHINFO_EXTENSION);
        if(!array_key_exists($ext, $allowed))
die("Error: Please select a valid file format.");

        // Verify file size - 5MB maximum
        $maxsize = 5 * 1024 * 1024;
        if($filesize > $maxsize) die("Error: File
size is larger than the allowed limit.");

        // Verify MYME type of the file
        if(in_array($filetype, $allowed)){
            // Check whether file exists before
uploading it
            if(file_exists("upload/" . $filename)){
                echo $filename . " is already
exists.";
            } else{

move_uploaded_file($_FILES["photo"]["tmp_name"],
"upload/" . $filename);
```

```
                echo "Your file was uploaded 
successfully.";
            }
        } else{
            echo "Error: There was a problem 
uploading your file. Please try again.";
        }
    } else{
        echo "Error: " . $_FILES["photo"]["error"];
    }
}
?>
```

Note: The above script prevents uploading a file with the same name as an existing file in the same folder. However, if you want to allow this just prepend the file name with a random string or timestamp, like `$filename = time() . '_' . $_FILES["photo"]["name"];`

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

Explanation of Code

Once the form is submitted information about the uploaded file can be accessed via PHP superglobal array called `$_FILES`. For example, our upload form contains a file select field called photo (i.e. `name="photo"`), if any user uploaded a file using this field, we can obtains its details like the name, type, size, temporary name or any error occurred while attempting the upload via the `$_FILES["photo"]` associative array, like this:

- `$_FILES["photo"]["name"]` — This array value specifies the original name of the file, including the file extension. It doesn't include the file path.
- `$_FILES["photo"]["type"]` — This array value specifies the MIME type of the file.

- `$_FILES["photo"]["size"]` — This array value specifies the file size, in bytes.
- `$_FILES["photo"]["tmp_name"]` — This array value specifies the temporary name including full path that is assigned to the file once it has been uploaded to the server.
- `$_FILES["photo"]["error"]` — This array value specifies error or status code associated with the file upload, e.g. it will be 0, if there is no error.

The PHP code in the following example will simply display the details of the uploaded file and stores it in a temporary directory on the web server.

Example
Download

```
<?php
if($_FILES["photo"]["error"] > 0){
    echo "Error: " . $_FILES["photo"]["error"] .
"<br>";
} else{
    echo "File Name: " . $_FILES["photo"]["name"] .
"<br>";
    echo "File Type: " . $_FILES["photo"]["type"] .
"<br>";
    echo "File Size: " . ($_FILES["photo"]["size"] /
1024) . " KB<br>";
    echo "Stored in: " .
$_FILES["photo"]["tmp_name"];
}
?>
```

Tip: Once a file has been successfully uploaded, it is automatically stored in a temporary directory on the server. To store this file on a permanent basis, you need to move it from the temporary directory to a permanent location using the PHP's `move_uploaded_file()` function.

PHP File Download

In this tutorial you will learn how to force download a file using PHP.

Downloading Files with PHP

Normally, you don't necessarily need to use any server side scripting language like PHP to download images, zip files, pdf documents, exe files, etc. If such kind of file is stored in a public accessible folder, you can just create a hyperlink pointing to that file, and whenever a user click on the link, browser will automatically downloads that file.

Example
Try this code »

```
<a href="downloads/test.zip">Download Zip file</a>
<a href="downloads/masters.pdf">Download PDF file</a>
<a href="downloads/sample.jpg">Download Image
file</a>
<a href="downloads/setup.exe">Download EXE file</a>
```

Clicking a link that points to a PDF or an Image file will not cause it to download to your hard drive directly. It will only open the file in your browser. Further you can save it to your hard drive. However, zip and exe files are downloaded automatically to the hard drive by default.

Forcing a Download Using PHP

You can force images or other kind of files to download directly to the user's hard drive using the PHP `readfile()` function. Here we're going to create a simple image gallery that allows users to download the image files from the browser with a single mouse click.

Let's create a file named "image-gallery.php" and place the following code inside it.

Example
Run this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Simple Image Gallery</title>
<style type="text/css">
    .img-box{
        display: inline-block;
        text-align: center;
        margin: 0 15px;
    }
</style>
</head>
<body>
    <?php
    // Array containing sample image file names
    $images = array("kites.jpg", "balloons.jpg");

    // Loop through array to create image gallery
    foreach($images as $image){
        echo '<div class="img-box">';
            echo '<img src="images/' . $image . '"
width="200" alt="' .  pathinfo($image,
PATHINFO_FILENAME) .'">';
            echo '<p><a href="download.php?file=' .
urlencode($image) . '">Download</a></p>';
        echo '</div>';
    }
    ?>
```

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

If you see the above example code carefully, you'll find the download link pints to a "download.php" file, the URL also contains image file name as a query string. Also, we've used PHP `urlencode()` function to encode the image file names so that it can be safely passed as URL parameter, because file names may contain URL unsafe characters.

Here's the complete code of "download.php" file, which force image download.

Example

Run this code »

```
<?php
if(isset($_REQUEST["file"])){
    // Get parameters
    $file = urldecode($_REQUEST["file"]); // Decode
URL-encoded string

    /* Test whether the file name contains illegal
characters
    such as "../" using the regular expression */
    if(preg_match('/^[^.][-a-z0-9_.]+[a-z]$/i',
$file)){
        $filepath = "images/" . $file;

        // Process download
        if(file_exists($filepath)) {
            header('Content-Description: File
Transfer');
            header('Content-Type: application/octet-
stream');
            header('Content-Disposition: attachment;
filename="'.basename($filepath).'"');
            header('Expires: 0');
            header('Cache-Control: must-revalidate');
            header('Pragma: public');
            header('Content-Length: ' .
filesize($filepath));
```

```
                flush(); // Flush system output buffer
                readfile($filepath);
                die();
            } else {
                http_response_code(404);
                  die();
            }
        } else {
            die("Invalid file name!");
        }
}
?>
```

Similarly, you can force download other files formats like word doc, pdf files, etc.

The regular expression in the above example (*line no-8*) will simply not allow those files whose name starts or ends with a dot character (`.`), for example, it allows the file names such as `kites.jpg` or `Kites.jpg`, `myscript.min.js` but do not allow `kites.jpg.` or `.kites.jpg`.

PHP Cookies

In this tutorial you will learn how to store a small amount of information within the user's browser itself using the PHP cookies.

What is a Cookie

A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer. They are typically used to keeping track of information such as username that the site can retrieve to personalize the page when user visit the website next time.

Tip: Each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.

Setting a Cookie in PHP

The `setcookie()` function is used to set a cookie in PHP. Make sure you call the `setcookie()` function before any output generated by your script otherwise cookie will not set. The basic syntax of this function can be given with:

```
setcookie(name, value, expire, path, domain, secure);
```

The parameters of the `setcookie()` function have the following meanings:

Parameter	Description
`name`	The name of the cookie.
`value`	The value of the cookie. Do not store sensitive information since this value is store computer.
`expires`	The expiry date in UNIX timestamp format. After this time cookie will become inac value is 0.
`path`	Specify the path on the server for which the cookie will be available. If set to `/`, the available within the entire domain.
`domain`	Specify the domain for which the cookie is available to e.g www.example.com.
`secure`	This field, if present, indicates that the cookie should be sent only if a secure HTTPS

Tip: If the expiration time of the cookie is set to 0, or omitted, the cookie will expire at the end of the session i.e. when the browser closes.

Here's an example that uses `setcookie()` function to create a cookie named `username` and assign the value value `John Carter` to it. It also specify that the cookie will expire after 30 days (`30 days * 24 hours * 60 min * 60 sec`).

Example
Download

```php
<?php
// Setting a cookie
setcookie("username", "John Carter",
time()+30*24*60*60);
?>
```

Note: All the arguments except the name are optional. You may also replace an argument with an empty string ("") in order to skip that argument, however to skip the expire argument use a zero (0) instead, since it is an integer.

Warning: Don't store sensitive data in cookies since it could potentially be manipulated by the malicious user. To store the sensitive data securely use sessions instead.

Accessing Cookies Values

The PHP `$_COOKIE` superglobal variable is used to retrieve a cookie value. It typically an associative array that contains a list of all the cookies values sent by the browser in the current request, keyed by cookie name. The individual cookie value can be accessed using standard array notation, for example to display the username cookie set in the previous example, you could use the following code.

Example
Download

```php
<?php
// Accessing an individual cookie value
echo $_COOKIE["username"];
?>
```

The PHP code in the above example produce the following output.

John Carter

It's a good practice to check whether a cookie is set or not before accessing its value. To do this you can use the PHP `isset()` function, like this:

Example
Download

```php
<?php
// Verifying whether a cookie is set or not
if(isset($_COOKIE["username"])){
    echo "Hi " . $_COOKIE["username"];
} else{
    echo "Welcome Guest!";
}
?>
```

You can use the `print_r()` function like `print_r($_COOKIE);` to see the structure of this `$_COOKIE` associative array, like you with other arrays.

Removing Cookies

You can delete a cookie by calling the same `setcookie()` function with the cookie name and any value (such as an empty string) however this time you need the set the expiration date in the past, as shown in the example below:

Example
Download

```php
<?php
// Deleting a cookie
setcookie("username", "", time()-3600);
?>
```

Tip: You should pass exactly the same path, domain, and other arguments that you have used when you first created the cookie in order to ensure that the correct cookie is deleted.

PHP Sessions

In this tutorial you will learn how to store certain data on the server on a temporary basis using PHP session.

What is a Session

Although you can store data using cookies but it has some security issues. Since cookies are stored on user's computer it is possible for an attacker to easily modify a cookie content to insert potentially harmful data in your application that might break your application.

Also every time the browser requests a URL to the server, all the cookie data for a website is automatically sent to the server within the request. It means if you have stored 5 cookies on user's system, each having 4KB in size, the browser needs to upload 20KB of data each time the user views a page, which can affect your site's performance.

You can solve both of these issues by using the PHP session. A PHP session stores data on the server rather than user's computer. In a session based environment, every user is identified through a unique number called session identifier or SID. This unique session ID is used to link each user with their own information on the server like emails, posts, etc.

Tip: The session IDs are randomly generated by the PHP engine which is almost impossible to guess. Furthermore, because the session data is stored on the server, it doesn't have to be sent with every browser request.

Starting a PHP Session

Before you can store any information in session variables, you must first start up the session. To begin a new session, simply call the PHP `session_start()` function. It will create a new session and generate a unique session ID for the user.

The PHP code in the example below simply starts a new session.

Example
Download

```
<?php
// Starting session
session_start();
?>
```

The `session_start()` function first checks to see if a session already exists by looking for the presence of a session ID. If it finds one, i.e. if the session is already started, it sets up the session variables and if doesn't, it starts a new session by creating a new session ID.

Note: You must call the `session_start()` function at the beginning of the page i.e. before any output generated by your script in the browser, much like you do while setting the cookies with `setcookie()` function.

Storing and Accessing Session Data

You can store all your session data as key-value pairs in the `$_SESSION[]` superglobal array. The stored data can be accessed during lifetime of a session. Consider the following script, which creates a new session and registers two session variables.

Example
Download

```
<?php
// Starting session
session_start();

// Storing session data
$_SESSION["firstname"] = "Peter";
$_SESSION["lastname"] = "Parker";
?>
```

To access the session data we set on our previous example from any other page on the same web domain — simply recreate the session by calling `session_start()` and then pass the corresponding key to the `$_SESSION` associative array.

Example
Download

```
<?php
// Starting session
session_start();

// Accessing session data
echo 'Hi, ' . $_SESSION["firstname"] . ' ' .
$_SESSION["lastname"];
?>
```

The PHP code in the example above produce the following output.

Hi, Peter Parker

Note: To access the session data in the same page there is no need to recreate the session since it has been already started on the top of the page.

Destroying a Session

If you want to remove certain session data, simply unset the corresponding key of the `$_SESSION` associative array, as shown in the following example:

Example
Download

```php
<?php
// Starting session
session_start();

// Removing session data
if(isset($_SESSION["lastname"])){
    unset($_SESSION["lastname"]);
}
?>
```

However, to destroy a session completely, simply call the `session_destroy()` function. This function does not need any argument and a single call destroys all the session data.

Example
Download

```php
<?php
// Starting session
session_start();

// Destroying session
session_destroy();
?>
```

Note: Before destroying a session with the `session_destroy()` function, you need to first recreate the session environment if it is not already there using the `session_start()` function, so that there is something to destroy.

Every PHP session has a timeout value — a duration, measured in seconds — which determines how long a session should remain alive in the absence of any user activity. You can adjust this timeout duration by changing the value of `session.gc_maxlifetime` variable in the PHP configuration file (`php.ini`).

PHP Send Emails

In this tutorial you will learn how to send simple text or HTML emails directly from the script using the PHP `mail()` function.

The PHP `mail()` Function

Sending email messages are very common for a web application, for example, sending welcome email when a user create an account on your website, sending newsletters to your registered users, or getting user feedback or comment through website's contact form, and so on.

You can use the PHP built-in `mail()` function for creating and sending email messages to one or more recipients dynamically from your PHP application either in a plain-text form or formatted HTML. The basic syntax of this function can be given with:

```
mail(to, subject, message, headers, parameters)
```

The following table summarizes the parameters of this function.

Parameter	Description

Parameter	Description
Required — The following parameters are required	
`to`	The recipient's email address.
`subject`	Subject of the email to be sent. This parameter i.e. the subject line cannot conta character (`\n`).
`message`	Defines the message to be sent. Each line should be separated with a line feed- not exceed 70 characters.
Optional — The following parameters are optional	
`headers`	This is typically used to add extra headers such as "From", "Cc", "Bcc". The addi be separated with a carriage return plus a line feed-CRLF (`\r\n`).
`parameters`	Used to pass additional parameters.

Sending Plain Text Emails

The simplest way to send an email with PHP is to send a text email. In the example below we first declare the variables — recipient's email address, subject line and message body — then we pass these variables to the `mail()` function to send the email.

Example
Download

```
<?php
$to = 'maryjane@email.com';
$subject = 'Marriage Proposal';
$message = 'Hi Jane, will you marry me?';
```

```
$from = 'peterparker@email.com';

// Sending email
if(mail($to, $subject, $message)){
    echo 'Your mail has been sent successfully.';
} else{
    echo 'Unable to send email. Please try again.';
}
?>
```

Sending HTML Formatted Emails

When you send a text message using PHP, all the content will be treated as simple text. We're going to improve that output, and make the email into a HTML-formatted email.

To send an HTML email, the process will be the same. However, this time we need to provide additional headers as well as an HTML formatted message.

Example
Download

```
<?php
$to = 'maryjane@email.com';
$subject = 'Marriage Proposal';
$from = 'peterparker@email.com';

// To send HTML mail, the Content-type header must be
set
$headers  = 'MIME-Version: 1.0' . "\r\n";
$headers .= 'Content-type: text/html; charset=iso-
8859-1' . "\r\n";

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

```
// Compose a simple HTML email message
$message = '<html><body>';
$message .= '<h1 style="color:#f40;">Hi Jane!</h1>';
$message .= '<p style="color:#080;font-
size:18px;">Will you marry me?</p>';
$message .= '</body></html>';

// Sending email
if(mail($to, $subject, $message, $headers)){
    echo 'Your mail has been sent successfully.';
} else{
    echo 'Unable to send email. Please try again.';
}
?>
```

Note: However, the PHP `mail()` function is a part of the PHP core but you need to set up a mail server on your machine to make it really work.

In the next two chapters (PHP Form Handling and PHP Form Validation) you will learn how to implement an interactive contact form on your website to receive the user's comment and feedback through emails using this PHP send mail feature.

PHP Form Handling

In this tutorial you'll learn how to collect user inputs submitted through a form using the PHP superglobal variables `$_GET`, `$_POST` and `$_REQUEST`.

Creating a Simple Contact Form

In this tutorial we are going to create a simple HMTL contact form that allows users to enter their comment and feedback then displays it to the browser using PHP.

Open up your favorite code editor and create a new PHP file. Now type the following code and save this file as "contact-form.php" in the root directory of your project.

Example
Download

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Form</title>
</head>
<body>
    <h2>Contact Us</h2>
    <p>Please fill in this form and send us.</p>
    <form action="process-form.php" method="post">
        <p>
            <label
for="inputName">Name:<sup>*</sup></label>
            <input type="text" name="name"
id="inputName">
        </p>
        <p>
            <label
for="inputEmail">Email:<sup>*</sup></label>
            <input type="text" name="email"
id="inputEmail">
        </p>
        <p>
            <label
for="inputSubject">Subject:</label>
            <input type="text" name="subject"
id="inputSubject">
        </p>
        <p>
            <label
for="inputComment">Message:<sup>*</sup></label>
            <textarea name="message"
id="inputComment" rows="5" cols="30"></textarea>
        </p>
        <input type="submit" value="Submit">
        <input type="reset" value="Reset">
    </form>
</body>
</html>
```

Explanation of code

Notice that there are two attributes within the opening `<form>` tag:

- The `action` attribute references a PHP file "process-form.php" that receives the data entered into the form when user submit it by pressing the submit button.
- The `method` attribute tells the browser to send the form data through POST method.

Rest of the elements inside the form are basic form controls to receive user inputs. To learn more about HTML form elements please check out the HTML Forms tutorial.

Capturing Form Data with PHP

To access the value of a particular form field, you can use the following superglobal variables. These variables are available in all scopes throughout a script.

Superglobal	**Description**
$_GET	Contains a list of all the field names and values sent by a form using the get meth parameters).
$_POST	Contains a list of all the field names and values sent by a form using the post met visible in the URL).

Superglobal	Description
`$_REQUEST`	Contains the values of both the `$_GET` and `$_POST` variables as well as the valu the `$_COOKIE` superglobal variable.

When a user submit the above contact form through clicking the submit button, the form data is sent to the "process-form.php" file on the server for processing. It simply captures the information submitted by the user and displays it to browser.

The PHP code of "process-form.php" file will look something like this:

Example
Download

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Form</title>
</head>
<body>
    <h1>Thank You</h1>
    <p>Here is the information you have 
submitted:</p>
    <ol>
        <li><em>Name:</em> <?php echo 
$_POST["name"]?></li>
        <li><em>Email:</em> <?php echo 
$_POST["email"]?></li>
        <li><em>Subject:</em> <?php echo 
$_POST["subject"]?></li>
        <li><em>Message:</em> <?php echo 
$_POST["message"]?></li>
    </ol>
</body>
</html>
```

The PHP code above is quite simple. Since the form data is sent through the post method, you can retrieve the value of a particular form field by passing its name to the `$_POST` superglobal array, and displays each field value using `echo()` statement.

In real world you cannot trust the user inputs; you must implement some sort of validation to filter the user inputs before using them. In the next chapter you will learn how sanitize and validate this contact form data and send it through the email using PHP.

PHP Form Validation

> In this tutorial you'll learn how to sanitize and validate form data using PHP filters.

Sanitizing and Validating Form Data

As you have seen in the previous tutorial, the process of capturing and displaying the submitted form data is quite simple. In this tutorial you will learn how to implement a simple contact form on your website that allows the user to send their comment and feedback through email. We will use the same PHP `mail()` function to send the emails.

We are also going to implement some basic security feature like sanitization and validation of the user's input so that user can not insert potentially harmful data that compromise the website security or might break the application.

The following is our all-in-one PHP script which does the following things:

- It will ask the users to enter his comments about the website.
- The same script displays the contact form and process the submitted form data.

- The script sanitizes and validates the user inputs. If any required field (marked with *) is missing or validation failed due to incorrect inputs the script redisplays the form with an error message for corresponding form field.
- The script remembers which fields the user has already filled in, and prefills those fields when the form redisplayed due to validation error.
- If the data submitted by the user are acceptable and everything goes well it will send an email to the website administrator and display a success message to the user.

Type the following code in "contact.php" file and save in your project root directory:

Example
Download

```
<?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="contact.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>
```

Explanation of code

You might think what that code was all about. OK, let's get straight into it.

- The `filterName()` function (*line no-03*) validate input value as person's name. A valid name can only contain alphabetical characters (a-z, A-Z).
- The `filterEmail()` function (*line no-14*) validate input value as email address.
- The `filterString()` function (*line no-25*) only sanitize the input value by stripping HTML tags and special characters. It doesn't validate the input value against anything.
- The attribute `action="contact.php"` (*line no-111*) inside the `<form>` tag specifies that the same `contact.php` file display the form as well as process the form data.
- The PHP code inside the value attribute of `<input>` and `<textarea>` e.g. `<?php echo $name; ?>` display prefilled value when form is redisplayed upon validation error.
- The PHP code inside the `.error` class e.g. `<span class="error"><?php echo $nameErr; ?></span>` display error for corresponding field.

Rest the thing we have already covered in previous chapters. To learn more about sanitize and validate filters, please check out the PHP Filter reference.

Note: You need to setup a mail server on your machine for the PHP `mail()` function to work. If you just want to implement the form validation you can replace the mail part (line no. 81 to 94) with your own custom code.

PHP Filters

In this tutorial you will learn how to sanitize and validate user inputs in PHP.

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.

Sanitize a String

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

Example
Run this code »

```
<?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.

Example

Run this code »

```
<?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:

Example
Run this code »

```php
<?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.

Example
Run this code »

```php
<?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:

Example
Run this code »

```
<?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.

Example
Run this code »

```
<?php
// Sample email address
$email = "someone@@example.com";

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

```php
// 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";
}
?>
```

Note: The `FILTER_SANITIZE_EMAIL` filter removes all invalid characters from the provided email address string except letters, digits and `!#$%&'*+-=?^_`{|}~@.[]`.

Sanitize and Validate URLs

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

Example
Run this code »

```php
<?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";
}
?>
```

Note: The `FILTER_SANITIZE_URL` filter removes all invalid characters from the provided URL string except letters, digits and `$-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=`.

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:

Example
Run this code »

```
<?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";
}
?>
```

See the tutorial on HTML URL to learn about the different components of a URL.

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.

Example
Run this code »

```
<?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 Error Handling

In this tutorial you will learn how to use the PHP's error handling functions to deal with the error conditions gracefully.

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 executio stopped immediately.
`E_WARNING`	2	A run-time warning. It is non-fatal and most errors tend to fall into execution of the script is not stopped.
`E_NOTICE`	8	A run-time notice. Indicate that the script encountered something error, although the situation could also occur when running a scri
`E_USER_ERROR`	256	A fatal user-generated error message. This is like an `E_ERROR`, ex the PHP script using the function `trigger_error()` rather thar
`E_USER_WARNING`	512	A non-fatal user-generated warning message. This is like an `E_WA` generated by the PHP script using the function `trigger_error` PHP. engine
`E_USER_NOTICE`	1024	A user-generated notice message. This is like an `E_NOTICE`, excep the PHP script using the function `trigger_error()` rather thar

`E_STRICT`	2048	Not strictly an error, but triggered whenever PHP encounters code problems or forward incompatibilities
`E_ALL`	8191	All errors and warnings, except of `E_STRICT` prior to PHP 5.4.0.

For more error levels, please check out the reference on PHP Error Levels.

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 sofisticated. 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
Download

```
<?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
Download

```
<?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 (`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 t 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
Download

```
<?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
Download

```
<?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:

Example
Download

```
<?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.";
?>
```

Send Error Messages by E-Mail

You can also send e-mail with the error details using the same `error_log()` function.

Example
Download

```
<?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, 1, "webmaster@example.com");
    die("There was a problem, please try again. Error
report submitted to webmaster.");
}
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.

Example
Download

```
<?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.

Example
Download

```
<?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 Classes and Objects

In this tutorial you will learn how to write code in object-oriented style in PHP.

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 behavior with less code and shorter development time and high degree of reusability.

The following sections will describe how classes and objects work in PHP.

Tip: 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.

Example
Download

```
<?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. We will learn more about this a little later in this chapter.

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.

Create another PHP file name test.php and put the following code inside it.

Example
Run this code »

```
<?php
// Include class definition
require "Rectangle.php";

// Create a new object from Rectangle class
$obj = new Rectangle;

// Get the object properties values
echo $obj->length; // Output: 0
echo $obj->width; // Output: 0

// Set object properties values
$obj->length = 30;
$obj->width = 20;

// Read the object properties values again to show
the change
echo $obj->length; // Output: 30
echo $obj->width; // Output: 20


// Call the object methods
echo $obj->getPerimeter(); // Output: 100
echo $obj->getArea(); // Output: 600
?>
```

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:

Example
Run this code »

```php
<?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 $obj1->getArea(); // Output: 0
echo $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 $obj1->getArea(); // Output: 600
echo $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.

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.

Example
Run this code »

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

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

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

// Output a message at the end of the file
echo "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:

Example
Run this code »

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

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

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

// Destroy the object
unset($obj);

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

Now, the PHP code in the above example will produce the following output:

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

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

Note: 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.

Extending Classes through Inheritance

Classes can inherit the properties and methods of another class using the `extends` keyword. This process of extensibility is called inheritance. It is probably the most powerful reason behind using the object-oriented programming model.

Example
Run this code »

```
<?php
// Include class definition
require "Rectangle.php";

// Define a new class based on an existing class
class Square extends Rectangle
{
    // Method to test if the rectangle is also a
square
    public function isSquare(){
        if($this->length == $this->width){
            return true; // Square
        } else{
            return false; // Not a square
        }
    }
}

// Create a new object from Square class
```

```
$obj = new Square;

// Set object properties values
$obj->length = 20;
$obj->width = 20;

// Call the object methods
if($obj->isSquare()){
    echo "The area of the square is ";
} else{
    echo "The area of the rectangle is ";
};
echo $obj->getArea();
?>
```

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

The area of the square is 400

As you can see in the above example, even though the class definition of Square doesn't explicitly contain `getArea()` method nor the `$length` and `$width` property, instances of the Square class can use them, as they inherited from the parent Rectangle class.

Tip: Since a child class is derived from a parent class, it is also referred to as a derived class, and its parent is called the base class.

Controlling the Visibility of Properties and Methods

When working with classes, you can even restrict access to its properties and methods using the *visibility keywords* for greater control. There are three visibility keywords (from most visible to least visible): `public`, `protected`, `private`, which determines how and from where properties and methods can be accessed and modified.

- **public** — A public property or method can be accessed anywhere, from within the class and outside. This is the default visibility for all class members in PHP.
- **protected** — A protected property or method can only be accessed from within the class itself or in child or inherited classes i.e. classes that extends that class.
- **private** — A private property or method is accessible only from within the class that defines it. Even child or inherited classes cannot access private properties or methods.

The following example will show you how this visibility actually works:

Example
Download

```
<?php
// Class definition
class Automobile
{
    // Declare  properties
    public $fuel;
    protected $engine;
    private $transmission;
}
class Car extends Automobile
{
    // Constructor
    public function __construct(){
        echo 'The class "' . __CLASS__ . '" was
initiated!<br>';
    }
}

// Create an object from Automobile class
$automobile = new Automobile;

// Attempt to set $automobile object properties
$automobile->fuel = 'Petrol'; // ok
```

```
$automobile->engine = '1500 cc'; // fatal error
$automobile->transmission = 'Manual'; // fatal error

// Create an object from Car class
$car = new Car;

// Attempt to set $car object properties
$car->fuel = 'Diesel'; // ok
$car->engine = '2200 cc'; // fatal error
$car->transmission = 'Automatic'; // undefined
?>
```

Static Properties and Methods

In addition to the visibility, properties and methods can also be declared as `static`, which makes them accessible without needing an instantiation of the class. Static properties and methods can be accessed using the scope resolution operator (`::`), like this: `ClassName::$property` and `ClassName::method()`.

A property declared as static cannot be accessed via the object of that class though a static method can be, as demonstrated in the following example:

Example
Download

```
<?php
// Class definition
class HelloClass
{
    // Declare a static property
    public static $greeting = "Hello World!";

    // Declare a static method
    public static function sayHello(){
        echo self::$greeting;
    }
}
// Attempt to access static property and method
directly
echo HelloClass::$greeting; // Output: Hello World!
HelloClass::sayHello(); // Output: Hello World!

// Attempt to access static property and method via
object
$hello = new HelloClass;
echo $hello->greeting; // Strict Warning
$hello->sayHello(); // Output: Hello World!
?>
```

The keyword `self` in the above example means "the current class". It is never preceded by a dollar sign (`$`) and always followed by the `::` operator (e.g. `self::$name`).

The `self` keyword is different from the `this` keyword which means "the current object" or "the current instance of a class". The `this` keyword is always preceded by a dollar sign (`$`) and followed by the `->` operator (e.g. `$this->name`).

Note: Since static methods can be called without an instance of a class (i.e. object), the pseudo-variable `$this` is not available inside the method declared as static.

We hope you've understood the basic concepts of object-oriented programming by now. You'll find more examples on OOP in PHP and MySQL database section.

PHP Magic Constants

In this tutorial you will learn how to work with PHP magic constants.

What is Magic Constants

In the PHP constants chapter we've learned how to define and use constants in PHP script.

PHP moreover also provide a set of special predefined constants that change depending on where they are used. These constants are called magic constants. For example, the value of `__LINE__` depends on the line that it's used on in your script.

Magic constants begin with two underscores and end with two underscores. The following section describes some of the most useful magical PHP constants.

__LINE__

The `__LINE__` constant returns the current line number of the file, like this:

Example
Run this code »

```
<?php
echo "Line number " . __LINE__ . "<br>"; // Displays:
Line number 2
echo "Line number " . __LINE__ . "<br>"; // Displays:
Line number 3
echo "Line number " . __LINE__ . "<br>"; // Displays:
Line number 4
?>
```

__FILE__

The `__FILE__` constant returns full path and name of the PHP file that's being executed. If used inside an include, the name of the included file is returned.

Example
Download

```
<?php
// Displays the absolute path of this file
echo "The full path of this file is: " . __FILE__;
?>
```

__DIR__

The `__DIR__` constant returns the directory of the file. If used inside an include, the directory of the included file is returned. Here's an example:

Example
Download

```
<?php
// Displays the directory of this file
echo "The directory of this file is: " . __DIR__;
?>
```

__FUNCTION__

The `__FUNCTION__` constant returns the name of the current function.

Example
Run this code »

```
<?php
function myFunction(){
    echo  "The function name is - " . __FUNCTION__;
}
myFunction(); // Displays: The function name is -
myFunction
?>
```

__CLASS__

The `__CLASS__` constant returns the name of the current class.
Here's an example:

Example
Run this code »

```
<?php
class MyClass
{
    public function getClassName(){
        return __CLASS__;
    }
}
$obj = new MyClass();
echo $obj->getClassName(); // Displays: MyClass
?>
```

__METHOD__

The `__METHOD__` constant returns the name of the current class method.

Example
Run this code »

```
<?php
class Sample
{
    public function myMethod(){
        echo __METHOD__;
    }
}
$obj = new Sample();
$obj->myMethod(); // Displays: Sample::myMethod
?>
```

__NAMESPACE__

The `__NAMESPACE__` constant returns the name of the current namespace.

Example
Download

```php
<?php
namespace MyNamespace;
class MyClass
{
    public function getNamespace(){
        return __NAMESPACE__;
    }
}
$obj = new MyClass();
echo $obj->getNamespace(); // Displays: MyNamespace
?>
```

PHP JSON Parsing

In this tutorial you will learn how to encode and decode JSON data in PHP.

What is JSON

JSON stands for **J**ava**S**cript **O**bject **N**otation. JSON is a standard lightweight data-interchange format which is quick and easy to parse and generate.

JSON, like XML, is a text-based format that's easy to write and easy to understand for both humans and computers, but unlike XML, JSON data structures occupy less bandwidth than their XML versions. JSON is based on two basic structures:

- **Object:** This is defined as a collection of key/value pairs (i.e. `key:value`). Each object begins with a left curly bracket `{` and ends with a right curly bracket `}`. Multiple key/value pairs are separated by a comma `,`.

- **Array:** This is defined as an ordered list of values. An array begins with a left bracket `[` and ends with a right bracket `]`. Values are separated by a comma `,`.

In JSON, keys are always strings, while the value can be a **`string`**, **`number`**, **`true`** or **`false`**, **`null`** or even an **`object`** or an **`array`**. Strings must be enclosed in double quotes `"` and can contain escape characters such as `\n`, `\t` and `\`. A JSON object may look like this:

Example

Run this code »

```
{
    "book": {
        "name": "Harry Potter and the Goblet of
Fire",
        "author": "J. K. Rowling",
        "year": 2000,
        "genre": "Fantasy Fiction",
        "bestseller": true
    }
}
```

Whereas an example of JSON array would look something like this:

Example

Run this code »

```
{
    "fruits": [
        "Apple",
        "Banana",
        "Strawberry",
        "Mango"
    ]
}
```

Tip: A data-interchange format is a text format which is used to interchange or exchange data between different platforms and operating systems. JSON is the most popular and lightweight data-interchange format for web applications.

Parsing JSON with PHP

JSON data structures are very similar to PHP arrays. PHP has built-in functions to encode and decode JSON data. These functions are `json_encode()` and `json_decode()`, respectively. Both functions only works with UTF-8 encoded string data.

Encoding JSON Data in PHP

In PHP the `json_encode()` function is used to encode a value to JSON format. The value being encoded can be any PHP data type except a resource, like a database or file handle. The below example demonstrates how to encode a PHP associative array into a JSON object:

Example
Run this code »

```
<?php
// An associative array
$marks = array("Peter"=>65, "Harry"=>80, "John"=>78,
"Clark"=>90);

echo json_encode($marks);
?>
```

The output of the above example will look like this:

```
{"Peter":65,"Harry":80,"John":78,"Clark":90}
```

Similarly, you can encode the PHP indexed array into a JSON array, like this:

Example
Run this code »

```
<?php
// An indexed array
$colors = array("Red", "Green", "Blue", "Orange",
"Yellow");

echo json_encode($colors);
?>
```

The output of the above example will look like this:

```
["Red","Green","Blue","Orange","Yellow"]
```

You can also force `json_encode()` function to return an PHP indexed array as JSON object by using the `JSON_FORCE_OBJECT` option, as shown in the example below:

Example
Run this code »

```
<?php
// An indexed array
$colors = array("Red", "Green", "Blue", "Orange");

echo json_encode($colors, JSON_FORCE_OBJECT);
?>
```

The output of the above example will look like this:

```
{"0":"Red","1":"Green","2":"Blue","3":"Orange"}
```

As you can see in the above examples a non-associative array can be encoded as array or object. However, an associative array always encoded as object.

Decoding JSON Data in PHP

Decoding JSON data is as simple as encoding it. You can use the PHP `json_decode()` function to convert the JSON encoded string into appropriate PHP data type. The following example demonstrates how to decode or convert a JSON object to PHP object.

Example
Run this code »

```
<?php
// Store JSON data in a PHP variable
$json =
'{"Peter":65,"Harry":80,"John":78,"Clark":90}';

var_dump(json_decode($json));
?>
```

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

```
object(stdClass)#1 (4) { ["Peter"]=> int(65)
["Harry"]=> int(80) ["John"]=> int(78) ["Clark"]=>
int(90) }
```

By default the `json_decode()` function returns an object. However, you can optionally specify a second parameter **`$assoc`** which accepts a boolean value that when set as `true` JSON objects are decoded into associative arrays. It is `false` by default. Here's an example:

Example
Run this code »

```
<?php
// Store JSON data in a PHP variable
$json =
'{"Peter":65,"Harry":80,"John":78,"Clark":90}';

var_dump(json_decode($json, true));
```

```
?>
```

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

```
array(4) { ["Peter"]=> int(65) ["Harry"]=> int(80)
["John"]=> int(78) ["Clark"]=> int(90) }
```

Now let's check out an example that will show you how to decode the JSON data and access individual elements of the JSON object or array in PHP.

Example
Run this code »

```
<?php
// Assign JSON encoded string to a PHP variable
$json =
'{"Peter":65,"Harry":80,"John":78,"Clark":90}';

// Decode JSON data to PHP associative array
$arr = json_decode($json, true);
// Access values from the associative array
echo $arr["Peter"];  // Output: 65
echo $arr["Harry"];  // Output: 80
echo $arr["John"];   // Output: 78
echo $arr["Clark"];  // Output: 90

// Decode JSON data to PHP object
$obj = json_decode($json);
// Access values from the returned object
echo $obj->Peter;   // Output: 65
echo $obj->Harry;   // Output: 80
echo $obj->John;    // Output: 78
echo $obj->Clark;   // Output: 90
?>
```

You can also loop through the decoded data using `foreach()` loop, like this:

Example
Run this code »

```
<?php
```

```
// Assign JSON encoded string to a PHP variable
$json =
'{"Peter":65,"Harry":80,"John":78,"Clark":90}';

// Decode JSON data to PHP associative array
$arr = json_decode($json, true);

// Loop through the associative array
foreach($arr as $key=>$value){
    echo $key . "=>" . $value . "<br>";
}
echo "<hr>";
// Decode JSON data to PHP object
$obj = json_decode($json);

// Loop through the object
foreach($obj as $key=>$value){
    echo $key . "=>" . $value . "<br>";
}
?>
```

Extracting Values from Nested JSON Data in PHP

JSON objects and arrays can also be nested. A JSON object can arbitrarily contains other JSON objects, arrays, nested arrays, arrays of JSON objects, and so on. The following example will show you how to decode a nested JSON object and print all its values in PHP.

Example
Run this code »

```
<?php
// Define recursive function to extract nested values
function printValues($arr) {
    global $count;
    global $values;
    // Check input is an array
    if(!is_array($arr)){
        die("ERROR: Input is not an array");
    }

    /*
    Loop through array, if value is itself an array
recursively call th
unction else add the value found to the output items
array,
    and increment counter by 1 for each value found
    */
    foreach($arr as $key=>$value){
        if(is_array($value)){
            printValues($value);
        } else{
            $values[] = $value;
            $count++;
        }
    }

    // Return total count and values found in array
    return array('total' => $count, 'values' =>
$values);
}

// Assign JSON encoded string to a PHP variable
$json = '{
    "book": {
        "name": "Harry Potter and the Goblet of
Fire",
        "author": "J. K. Rowling",
        "year": 2000,
        "characters": ["Harry Potter", "Hermione
Granger", "Ron Weasley"],
        "genre": "Fantasy Fiction",
        "price": {
            "paperback": "$10.40", "hardcover":
"$20.32", "kindle": "4.11"
        }
    }
}';
// Decode JSON data into PHP associative array format
```

```
$arr = json_decode($json, true);

// Call the function and print all the values
$result = printValues($arr);
echo "<h3>" . $result["total"] . " value(s) found:
</h3>";
echo implode("<br>", $result["values"]);

echo "<hr>";

// Print a single value
echo $arr["book"]["author"] . "<br>";  // Output: J.
K. Rowling
echo $arr["book"]["characters"][0] . "<br>";  //
Output: Harry Potter
echo $arr["book"]["price"]["hardcover"];  // Output:
$20.32
?>
```

PHP Regular Expressions

In this tutorial you will learn how regular expressions work, as well as how to use them to perform pattern matching in an efficient way in PHP.

What is Regular Expression

Regular Expressions, commonly known as "**regex**" or "**RegExp**", are a specially formatted text strings used to find patterns in text. Regular expressions are one of the most powerful tools available today for effective and efficient text processing and manipulations. For example, it can be used to verify whether the format of data i.e. name, email, phone number, etc. entered by the user was correct or not, find or replace matching string within text content, and so on.

PHP (version 5.3 and above) supports Perl style regular expressions via its `preg_` family of functions. Why Perl style regular expressions?

Because Perl (*Practical Extraction and Report Language*) was the first mainstream programming language that provided integrated support for regular expressions and it is well known for its strong support of regular expressions and its extraordinary text processing and manipulation capabilities.

Let's begin with a brief overview of the commonly used PHP's built-in pattern-matching functions before delving deep into the world of regular expressions.

Function	**What it Does**
`preg_match()`	Perform a regular expression match.
`preg_match_all()`	Perform a global regular expression match.
`preg_replace()`	Perform a regular expression search and replace.
`preg_grep()`	Returns the elements of the input array that matched the patte
`preg_split()`	Splits up a string into substrings using a regular expression.
`preg_quote()`	Quote regular expression characters found within a string.

Note: The PHP `preg_match()` function stops searching after it finds the first match, whereas the `preg_match_all()` function continues searching until the end of the string and find all possible matches instead of stopping at the first match.

Regular Expression Syntax

Regular expression syntax includes the use of special characters (do not confuse with the HTML special characters). The characters that are given special meaning within a regular expression, are: `. * ? + [ ] ( ) { } ^ $ | \`. You will need to backslash these characters whenever you want to use them literally. For example, if you want to match ".", you'd have to write `\.`. All other characters automatically assume their literal meanings.

The following sections describe the various options available for formulating patterns:

Character Classes

Square brackets surrounding a pattern of characters are called a character class e.g. `[abc]`. A character class always matches a single character out of a list of specified characters that means the expression `[abc]` matches only a, b or c character.

Negated character classes can also be defined that match any character except those contained within the brackets. A negated character class is defined by placing a caret (`^`) symbol immediately after the opening bracket, like this `[^abc]`.

You can also define a range of characters by using the hyphen (`-`) character inside a character class, like `[0-9]`. Let's look at some examples of character classes:

RegExp	**What it Does**
`[abc]`	Matches any one of the characters a, b, or c.
`[^abc]`	Matches any one character other than a, b, or c.
`[a-z]`	Matches any one character from lowercase a to lowercase z.
`[A-Z]`	Matches any one character from uppercase a to uppercase z.
`[a-Z]`	Matches any one character from lowercase a to uppercase Z.

RegExp	What it Does
`[0-9]`	Matches a single digit between 0 and 9.
`[a-z0-9]`	Matches a single character between a and z or between 0 and 9.

The following example will show you how to find whether a pattern exists in a string or not using the regular expression and PHP `preg_match()` function:

Example
Run this code »

```php
<?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.";
}
?>
```

Similarly, you can use the `preg_match_all()` function to find all matches within a string:

Example
Run this code »

```php
<?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.";
?>
```

Tip: Regular expressions aren't exclusive to PHP. Languages such as Java, Perl, Python, etc. use the same notation for finding patterns in text.

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).
`\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). S `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
Run this code »

```
<?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

In the previous section we've learnt how to match a single character in a variety of fashions. But what if you want to match on more than one character? For example, let's say you want to find out words containing one or more instances of the letter p, or words containing at least two p's, and so on. This is where quantifiers come into play. With quantifiers you can specify how many times a character in a regular expression should match.

The following table lists the various ways to quantify a particular pattern:

RegExp	What it Does
p+	Matches one or more occurrences of the letter p.
p*	Matches zero or more occurrences of the letter p.
p?	Matches zero or one occurrences of the letter p.
p{2}	Matches exactly two occurrences of the letter p.
p{2,3}	Matches at least two occurrences of the letter p, but not more than three occurren
p{2,}	Matches two or more occurrences of the letter p.

RegExp	What it Does
`p{,3}`	Matches at most three occurrences of the letter p

The regular expression in the following example will splits the string at comma, sequence of commas, whitespace, or combination thereof using the PHP `preg_split()` function:

Example
Run this code »

```php
<?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>";
}
?>
```

Position Anchors

There are certain situations where you want to match at the beginning or end of a line, word, or string. To do this you can use anchors. Two common anchors are caret (`^`) which represent the start of the string, and the dollar (`$`) sign which represent the end of the string.

RegExp	What it Does
^p	Matches the letter p at the beginning of a line.

RegExp	What it Does
p$	Matches the letter p at the end of a line.

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:

Example
Run this code »

```
<?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>";
}
?>
```

Pattern Modifiers

A pattern modifier allows you to control the way a pattern match is handled. Pattern modifiers are placed directly after the regular expression, for example, if you want to search for a pattern in a case-insensitive manner, you can use the `i` modifier, like this: `/pattern/i`. The following table lists some of the most commonly used pattern modifiers.

Modifier	What it Does

Modifier	What it Does
`i`	Makes the match case-insensitive manner.
`m`	Changes the behavior of `^` and `$` to match against a newline boundary (i.e. start or e multiline string), instead of a string boundary.
`g`	Perform a global match i.e. finds all occurrences.
`o`	Evaluates the expression only once.
`s`	Changes the behavior of `.` (dot) to match all characters, including newlines.
`x`	Allows you to use whitespace and comments within a regular expression for clarity.

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.

Example

Run this code »

```
<?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.";
?>
```

Similarly, the following example shows how to match at the beginning of every line in a multi-line string using `^` anchor and `m` modifier with PHP `preg_match_all()` function.

Example

Run this code »

```
<?php
```

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

Word Boundaries

A word boundary character (`\b`) helps you search for the words that begins and/or ends with a pattern. For example, the regexp `/\bcar/` matches the words beginning with the pattern car, and would match cart, carrot, or cartoon, but would not match oscar.

Similarly, the regexp `/car\b/` matches the words ending with the pattern car, and would match scar, oscar, or supercar, but would not match cart. Likewise, the `/\bcar\b/` matches the words beginning and ending with the pattern car, and would match only the word car.

The following example will highlight the words beginning with car in bold:

Example
Run this code »

```
<?php
$pattern = '/\bcar\w*/';
$replacement = '<b>$0</b>';
$text = 'Words begining with car: cart, carrot,
cartoon. Words ending with car: scar, oscar,
supercar.';
echo preg_replace($pattern, $replacement, $text);
?>
```

We hope you have understood the basics of regular expression. To learn how to validate form data using regular expression, please check out the tutorial on PHP Form Validation.

PHP Exception Handling

In this tutorial you will learn how to throw and catch exceptions in PHP.

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.

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.

The following example demonstrates how exception handling works:

Example
Download

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

Example
Download

```
<?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.

Defining Custom Exceptions

You can even define your own custom exception handlers to treat different types of exceptions in a different way. It allows you to use a separate `catch` block for each exception type.

You can define a custom exception by extending the Exception class, because Exception is the base class for all exceptions. The custom exception class inherits all the properties and methods from PHP's Exception class. You can also add your custom methods to the custom exception class. Let's check out the following example:

Example
Download

```
<?php
// Extending the Exception class
class EmptyEmailException extends Exception {}
class InvalidEmailException extends Exception {}

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

try{
    // Throw exception if email is empty
    if($email == ""){
        throw new EmptyEmailException("<p>Please
enter your E-mail address!</p>");
    }

    // Throw exception if email is not valid
    if(filter_var($email, FILTER_VALIDATE_EMAIL) ===
FALSE) {
        throw new
InvalidEmailException("<p><b>$email</b> is not a
valid E-mail address!</p>");
    }

    // Display success message if email is valid
    echo "<p>SUCCESS: Email validation
successful.</p>";
} catch(EmptyEmailException $e){
    echo $e->getMessage();
} catch(InvalidEmailException $e){
    echo $e->getMessage();
}
?>
```

In the above example we've derived two new exception classes: **EmptyEmailException**, and **InvalidEmailException** from the Exception base class. Multiple `catch` blocks are used to display different error messages, depending on the type of exception generated.

Since these custom exception classes inherits the properties and methods from the Exception class, so we can use the Exception's class methods like `getMessage()`, `getLine()`, `getFile()`, etc. to retrieve error information from the exception object.

Setting a Global Exception Handler

As we've discussed earlier in this chapter if an exception is not caught, PHP generates a Fatal Error with an "Uncaught Exception ..." message. This error message may contain sensitive information like file name and line number where the problem occurs. If you don't want to expose such information to the user, you can create a custom function and register it with the `set_exception_handler()` function to handle all uncaught exceptions.

Example
Download

```
<?php
function handleUncaughtException($e){
    // Display generic error message to the user
    echo "Opps! Something went wrong. Please try
again, or contact us if the problem persists.";
```

```
    // Construct the error string
    $error = "Uncaught Exception: " . $message =
date("Y-m-d H:i:s - ");
    $error .= $e->getMessage() . " in file " . $e-
>getFile() . " on line " . $e->getLine() . "\n";

    // Log details of error in a file
    error_log($error, 3, "var/log/exceptionLog.log");
}

// Register custom exception handler
set_exception_handler("handleUncaughtException");

// Throw an exception
throw new Exception("Testing Exception!");
?>
```

Note: An uncaught exception will always result in script termination. So if you want the script to continue executing beyond the point where the exception occurred, you must have have at least one corresponding `catch` block for each `try` block.