Hyper Text Pre-Processor

Mohd Kaif
Mudit Kushwaha
Girish Kumar Shekhawat
Aman Deewan

PHP (Hyper Text Pre Processor)

- PHP, its features and advantages
- Basic PHP Syntax, tags, Data types, Constants and Variables, Operators and expressions.
- Paginators, popovers, progress, spinner
- Table, toasts, tooltips
- Bootstrap styling essentials and its use. Explain typography, floats, flex, Alignment, borders, position of elements shadow and visibility in bootstrap.
- PHP Conditional Events, Flow control and looping in PHP
- Functions in PHP
- PHP MySql connection-Get connected with mysqli_connect
- Password encryption with SHA in online forms
- Cookies in PHP
- Captcha text generation

Introduction to PHP

What is PHP (Hyper Text Pre Processor)?

PHP, which stands for Hyper Text Preprocessor, is a widely-used open-source server-side scripting language. Originally designed for web development, PHP is now also used as a general-purpose programming language. It was created by Rasmus Lerdorf in 1994 and has since evolved into one of the most widely used programming languages on the web.

It is embedded within HTML code and executed on the server, generating dynamic content that is sent to the user's web browser.

Origins and Evolution:

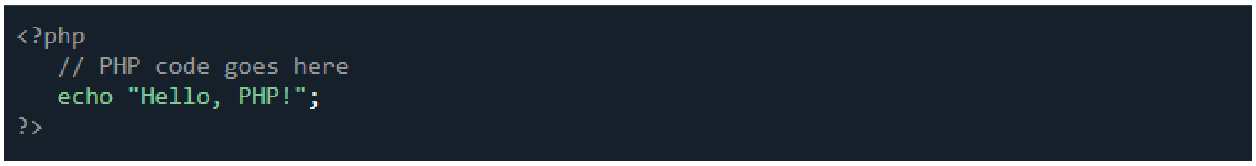
PHP's inception was as a set of Common Gateway Interface (CGI) binaries written in the C programming language to track online visits to Rasmus Lerdorf's resume. Over time, as more functionality was added, it transformed into a scripting language capable of building dynamic web pages.

Server-Side Scripting:

One of PHP's defining features is its server-side scripting capability. Unlike client-side scripting languages such as JavaScript, PHP code is executed on the server, generating HTML or other output sent to the user's browser. This allows for the creation of dynamic and interactive web pages.

Example :

Consider a simple "Hello, World!" program in PHP:



```php
<?php
   // PHP code goes here
   echo "Hello, PHP!";
?>
```

In this example, the PHP code is enclosed within `<?php ... ?>` tags. The `echo` statement is used to output the text "Hello, World!" to the web page.

PHP - Environment Setup

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

What Do I Need?

To start using PHP, you can:

- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL

To install PHP, it is best to install AMP (Apache, MySQL, PHP). There are many options for different operating systems.

As -

- WAMP (Window Apache, MySQL, PHP) for Windows Operating Systems
- LAMP(Linux Apache, MySQL, PHP) for Linux Operating Systems
- MAMP(Mac Apache, MySQL, PHP) for MAC Operating System
- XAMPP (Cross Apache, MySQL, PHP, PERL) It is cross platform, apart from this it also provides FileZilla, Mercury Mail,

To install PHP, it is best to install AMP (Apache, MySQL, PHP). There are many options for different operating systems.

Download WAMP Server

https://www.wampserver.com/en/

Download LAMP Server

https://csg.sph.umich.edu/abecasis/LAMP/download/

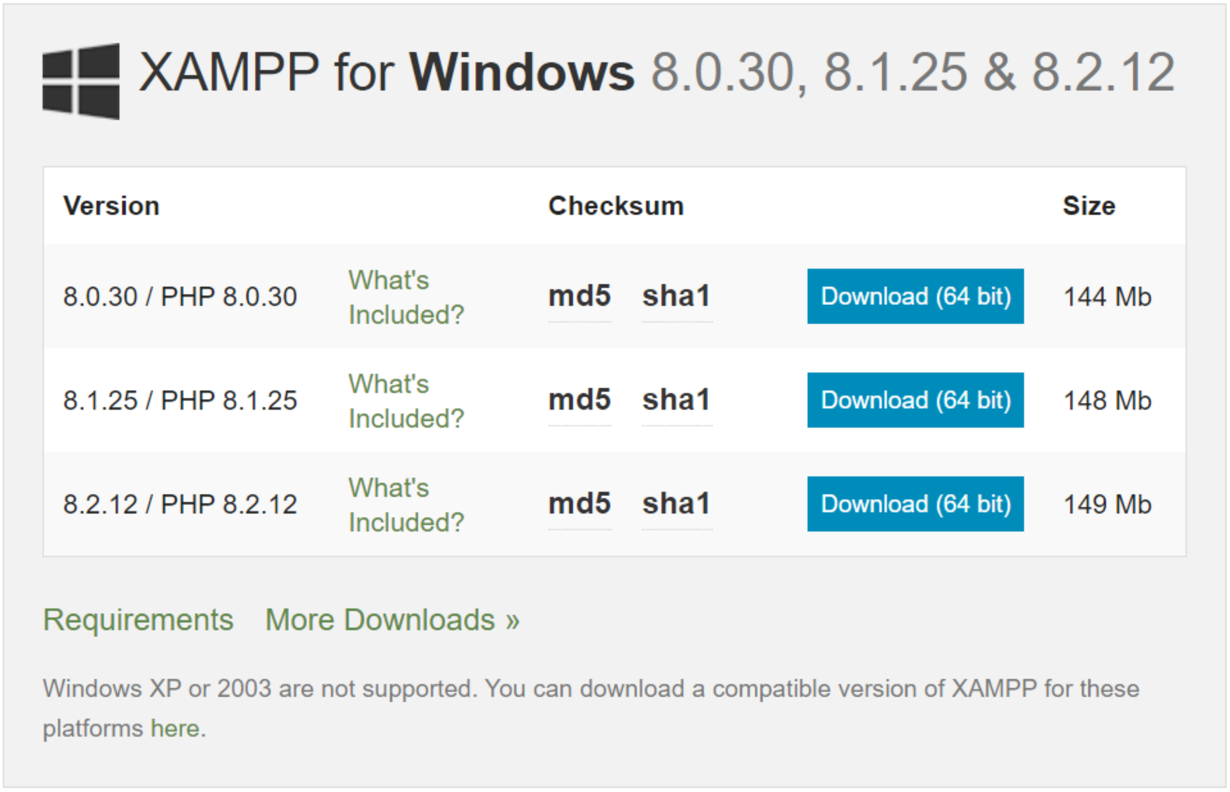
Download MAMP Server

https://www.mamp.info/en/downloads/

Download and Install XAMPP Server
https://www.apachefriends.org/download.html

After downloading the application by clicking on the links given above, you can go to the download directory and install it by double clicking.

Steps to Install XAMPP
To download XAMPP, visit the official website .



XAMPP for **Windows** 8.0.30, 8.1.25 & 8.2.12

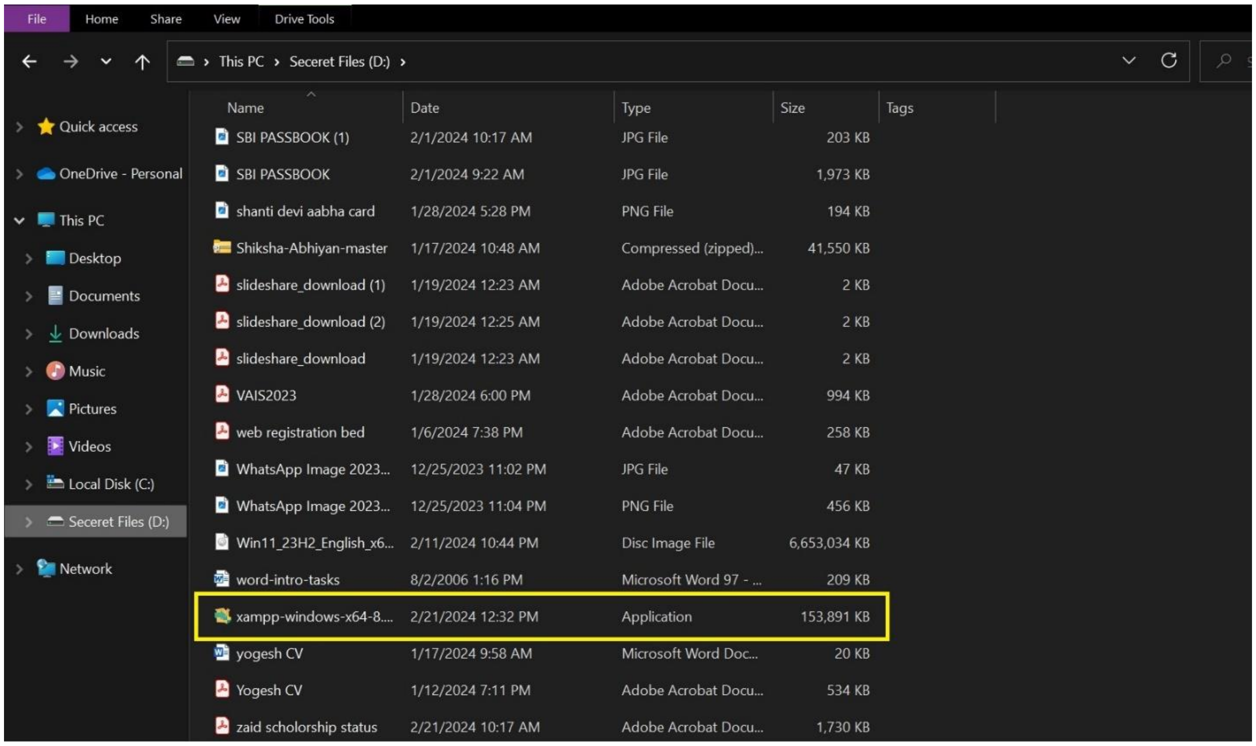
Version		Checksum			Size
8.0.30 / PHP 8.0.30	What's Included?	md5	sha1	Download (64 bit)	144 Mb
8.1.25 / PHP 8.1.25	What's Included?	md5	sha1	Download (64 bit)	148 Mb
8.2.12 / PHP 8.2.12	What's Included?	md5	sha1	Download (64 bit)	149 Mb

Requirements More Downloads »

Windows XP or 2003 are not supported. You can download a compatible version of XAMPP for these platforms here.

Now download XAMPP according to your operating system (32 Bit or 64 Bit)

Now install XAMPP by double clicking on the downloaded file.



Name	Date	Type	Size	Tags
SBI PASSBOOK (1)	2/1/2024 10:17 AM	JPG File	203 KB	
SBI PASSBOOK	2/1/2024 9:22 AM	JPG File	1,973 KB	
shanti devi aabha card	1/28/2024 5:28 PM	PNG File	194 KB	
Shiksha-Abhiyan-master	1/17/2024 10:48 AM	Compressed (zipped)...	41,550 KB	
slideshare_download (1)	1/19/2024 12:23 AM	Adobe Acrobat Docu...	2 KB	
slideshare_download (2)	1/19/2024 12:25 AM	Adobe Acrobat Docu...	2 KB	
slideshare_download	1/19/2024 12:23 AM	Adobe Acrobat Docu...	2 KB	
VAIS2023	1/28/2024 6:00 PM	Adobe Acrobat Docu...	994 KB	
web registration bed	1/6/2024 7:38 PM	Adobe Acrobat Docu...	258 KB	
WhatsApp Image 2023...	12/25/2023 11:02 PM	JPG File	47 KB	
WhatsApp Image 2023...	12/25/2023 11:04 PM	PNG File	456 KB	
Win11_23H2_English_x6...	2/11/2024 10:44 PM	Disc Image File	6,653,034 KB	
word-intro-tasks	8/2/2006 1:16 PM	Microsoft Word 97 - ...	209 KB	
xampp-windows-x64-8....	2/21/2024 12:32 PM	Application	153,891 KB	
yogesh CV	1/17/2024 9:58 AM	Microsoft Word Doc...	20 KB	
Yogesh CV	1/12/2024 7:11 PM	Adobe Acrobat Docu...	534 KB	
zaid scholorship status	2/21/2024 10:17 AM	Adobe Acrobat Docu...	1,730 KB	

PHP Version

To check your php version you can use the phpversion() function:

PHP Run First Program

In this program you will learn how to run PHP programs on XAMPP.

Write and save program

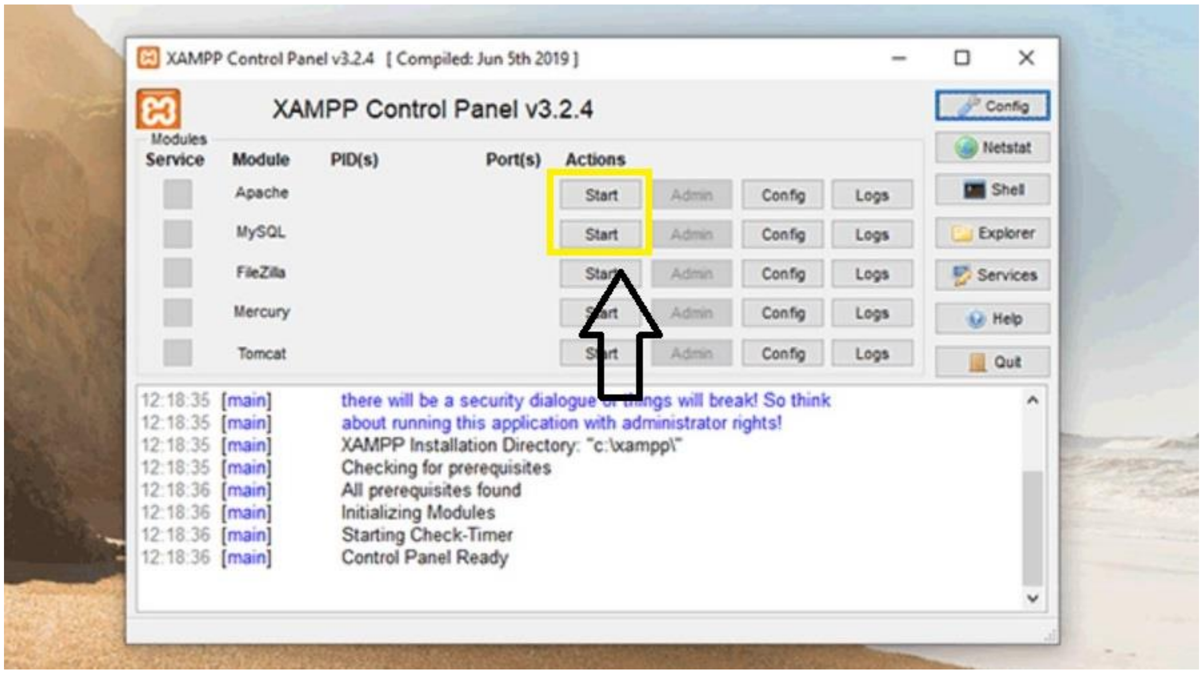
To run a PHP program, first write the PHP program in a text editor. And keep it anywhere in C:/xampp/htdocs. If possible, you can also create a new directory. I have created a directory named test inside the htdocs folder. And the name test.php is saved inside this directory.

Ex. C:/xampp/htdocs/test/test.php

```
<?php     $var = "Hello World !";
     echo $var;
  ?>
```

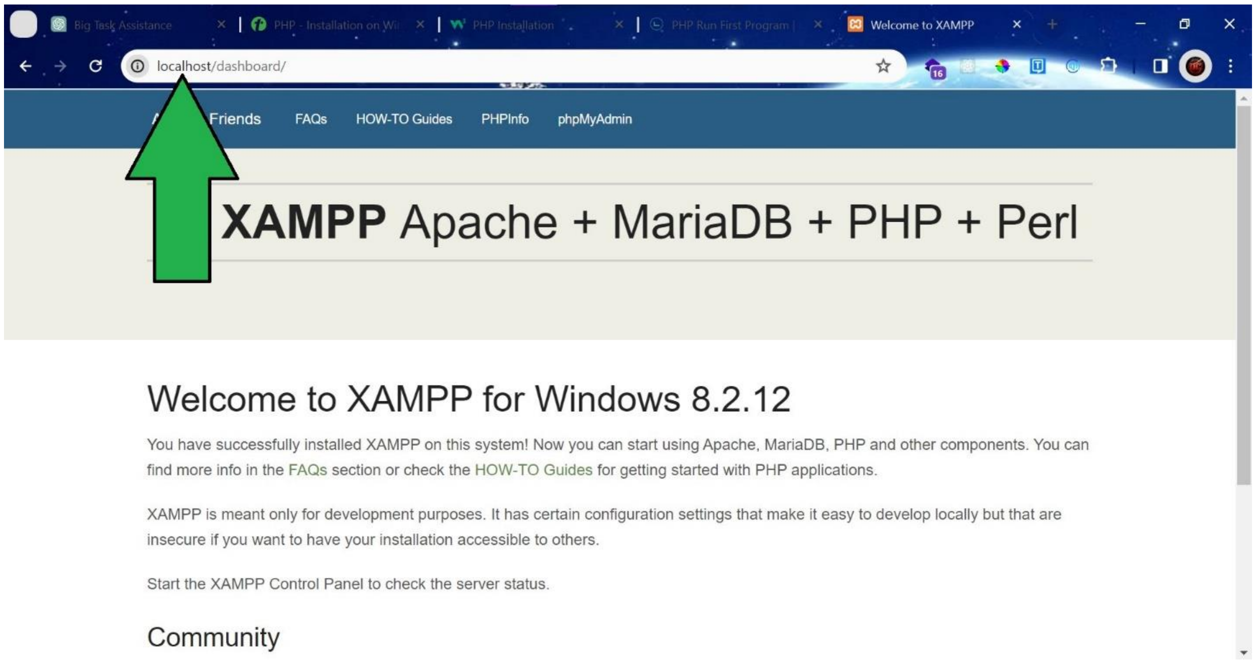
Open XAMPP Control Panel

Now open the XAMPP Control Panel, and then start the PHP Server. If you need MYSQL then you can also start it.



```
12:18:35 [main]   there will be a security dialogue or things will break! So think
12:18:35 [main]   about running this application with administrator rights!
12:18:35 [main]   XAMPP Installation Directory: "c:\xampp\"
12:18:35 [main]   Checking for prerequisites
12:18:36 [main]   All prerequisites found
12:18:36 [main]   Initializing Modules
12:18:36 [main]   Starting Check-Timer
12:18:36 [main]   Control Panel Ready
```

Open Browser



XAMPP Apache + MariaDB + PHP + Perl

Welcome to XAMPP for Windows 8.2.12

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the FAQs section or check the HOW-TO Guides for getting started with PHP applications.

XAMPP is meant only for development purposes. It has certain configuration settings that make it easy to develop locally but that are insecure if you want to have your installation accessible to others.

Start the XAMPP Control Panel to check the server status.

Community

Now Locate the file

Now remove dashboard after localhost and locate wherever you have saved the file and enter. I saved my file in C:/xampp/htdocs/test/test.php.

So write like this in the browser localhost/test/test.php then write like this in the browser and you will see the result something like this.

Output

Hello World !

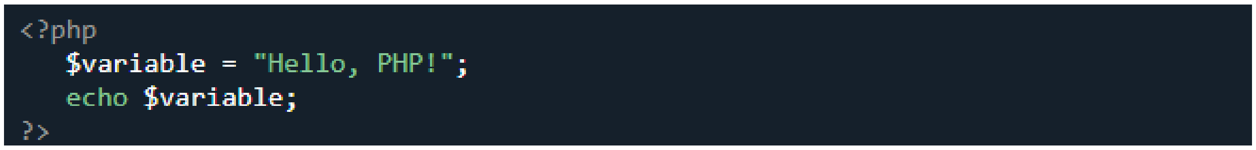
Now open the browser or click on Admin of PHP and MySQL in XAMPP Control Panel or write localhost in the browser and enter. After entering you will see the output something like this.

Features and Advantages of PHP

PHP comes with several features and advantages that contribute to its popularity in web development.

1. Easy to Learn and Use:

- PHP has a syntax that is similar to C and other programming languages, making it easy for developers to learn and use.



```php
<?php
   $variable = "Hello, PHP!";
   echo $variable;
?>
```

2. Open Source:

- PHP is open-source, meaning it is freely available for use, distribution, and modification.

3. Platform Independence:

- PHP is platform-independent, which means it can run on various operating systems like[1] Windows, Linux, and macOS.

4. Support for Multiple Databases:

- PHP supports a wide range of databases, including MySQL, PostgreSQL, and SQLite, making it flexible for database-driven web applications.

5. Server-Side Scripting:

- Being a server-side scripting language, PHP performs tasks on the server before sending the results to the client's browser. This allows for dynamic content generation.

6. Integration with Other Technologies:

- PHP can be easily integrated with other technologies like HTML, CSS, JavaScript, and various web servers.

7. Extensive Community Support:

- PHP has a large and active community of developers who contribute to its growth, providing support, resources, and a rich ecosystem of libraries and frameworks.

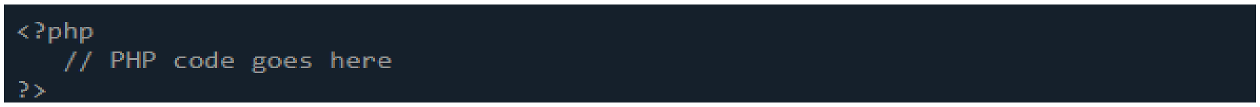
These features make PHP a powerful and versatile language for building dynamic and interactive web applications.

Getting Started With PHP

Basic PHP Syntax & Tags

PHP is a server-side scripting language, meaning that PHP scripts are executed on the server before the results are sent back to the browser as plain HTML. Understanding the basic syntax is crucial for effective PHP coding.

Placing PHP Script: A PHP script can be placed anywhere in the HTML document. It starts with **<?php** and ends with **?>**. The default file extension for PHP files is ".php".

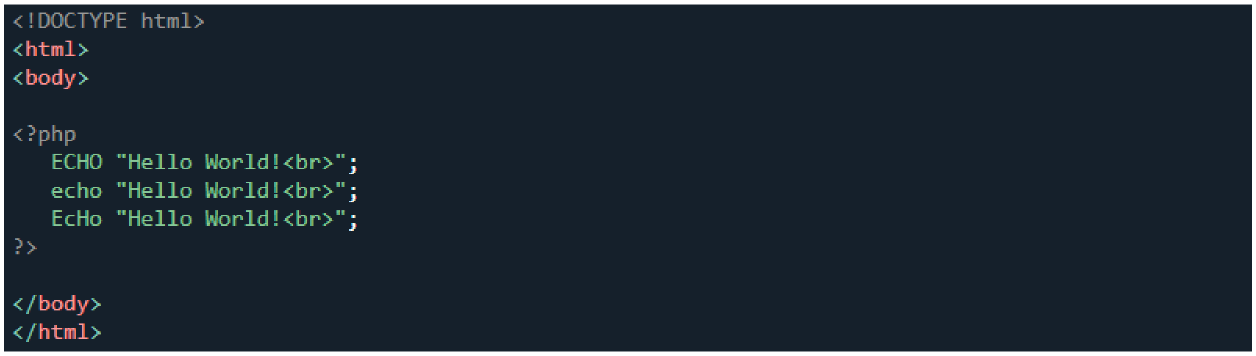


```
<?php
   // PHP code goes here
?>
```

Example PHP File: A typical PHP file contains a mix of HTML and PHP scripting code. Below is an example of a simple PHP file that outputs "Hello World!" using the built-in PHP function **echo**

Note that PHP statements end with a semicolon (**;**).

PHP Case Sensitivity: PHP keywords, classes, functions, and user-defined functions are not case-sensitive. However, variable names are case-sensitive. In the example below, various case variations of the **echo** keyword are considered equal and legal.



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

<?php
   ECHO "Hello World!<br>";
   echo "Hello World!<br>";
   EcHo "Hello World!<br>";
?>

</body>
</html>
```

Note: Variable names, on the other hand, must be used consistently with their defined case.

Data Types in PHP

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean

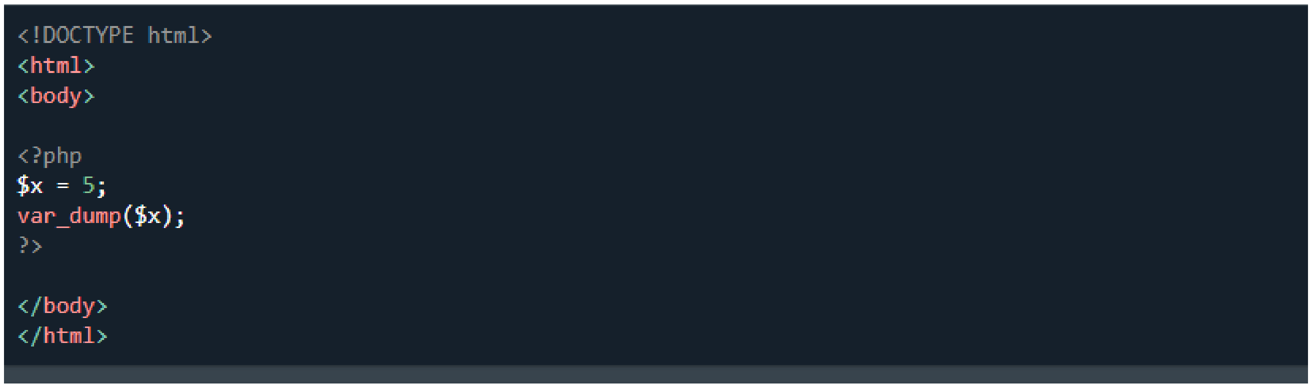
- Array
- Object
- NULL
- Resource

Getting the Data Type

You can get the data type of any object by using the var_dump() function.

Example

The var_dump() function returns the data type and the value:



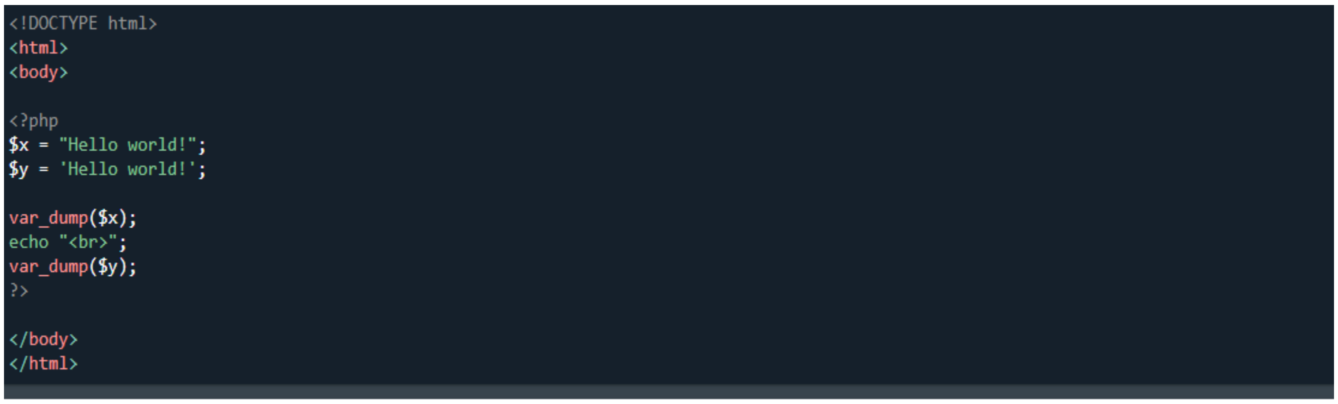
```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5;
var_dump($x);
?>

</body>
</html>
```

```
int(5)
```

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:



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

<?php
$x = "Hello world!";
$y = 'Hello world!';

var_dump($x);
echo "<br>";
var_dump($y);
?>

</body>
</html>
```

```
string(12) "Hello world!"
string(12) "Hello world!"
```

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

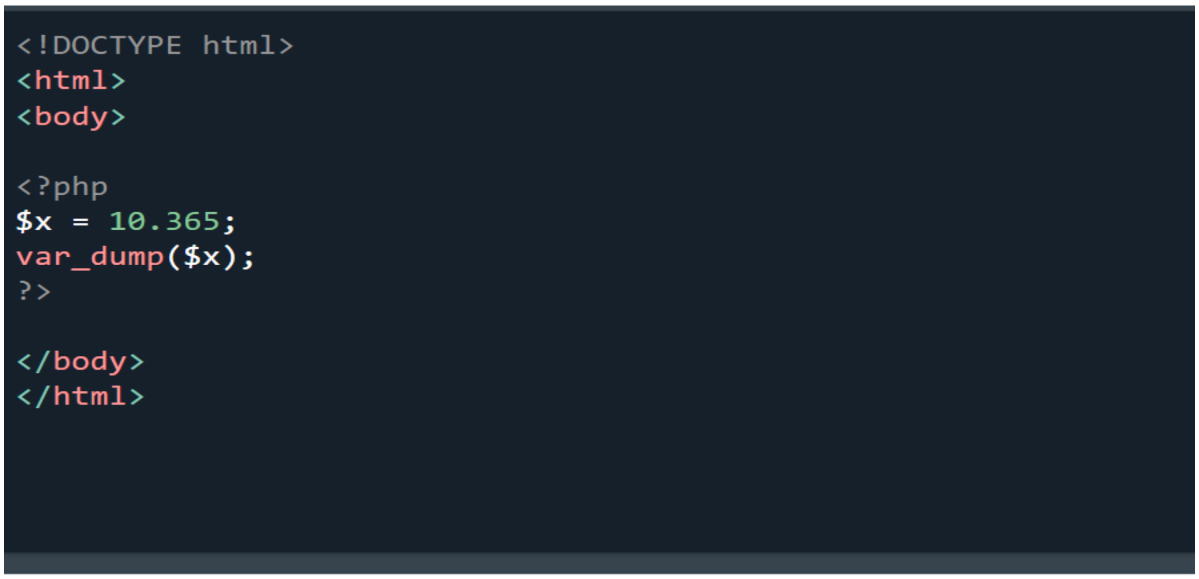
- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var_dump() function returns the data type and value:



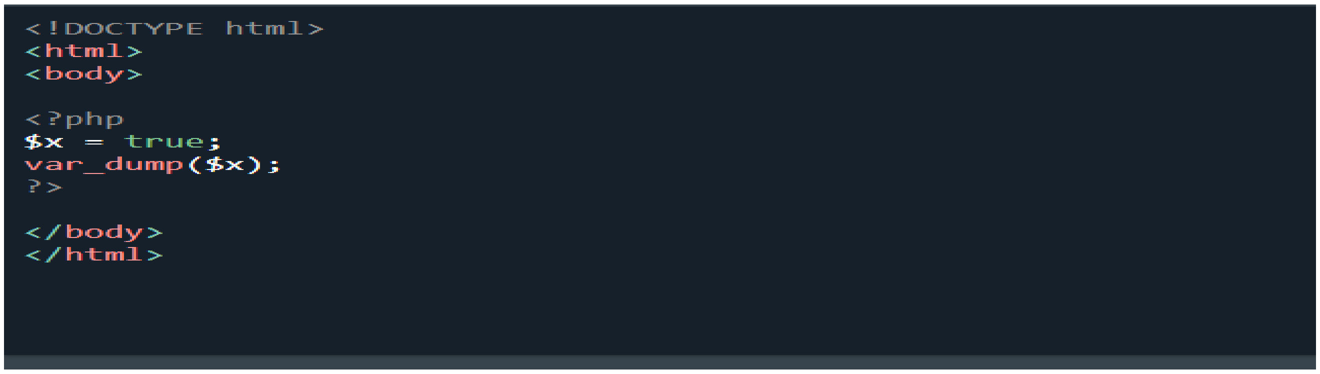
```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 10.365;
var_dump($x);
?>

</body>
</html>
```

```
float(10.365)
```

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

Booleans are often used in conditional testing.



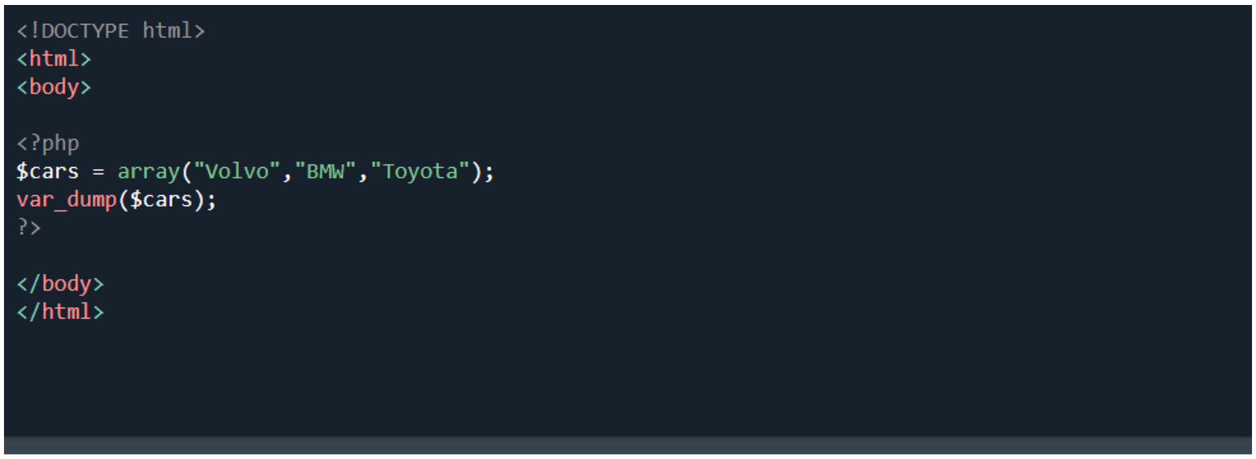
```
<!DOCTYPE html>
<html>
<body>

<?php
$x = true;
var_dump($x);
?>

</body>
</html>
```

```
bool(true)
```

PHP Array

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var_dump() function returns the data type and value:



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

<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>

</body>
</html>
```

```
array(3) {
  [0]=>
  string(5) "Volvo"
  [1]=>
  string(3) "BMW"
  [2]=>
  string(6) "Toyota"
}
```

PHP Object

Classes and objects are the two main aspects of object-oriented programming.

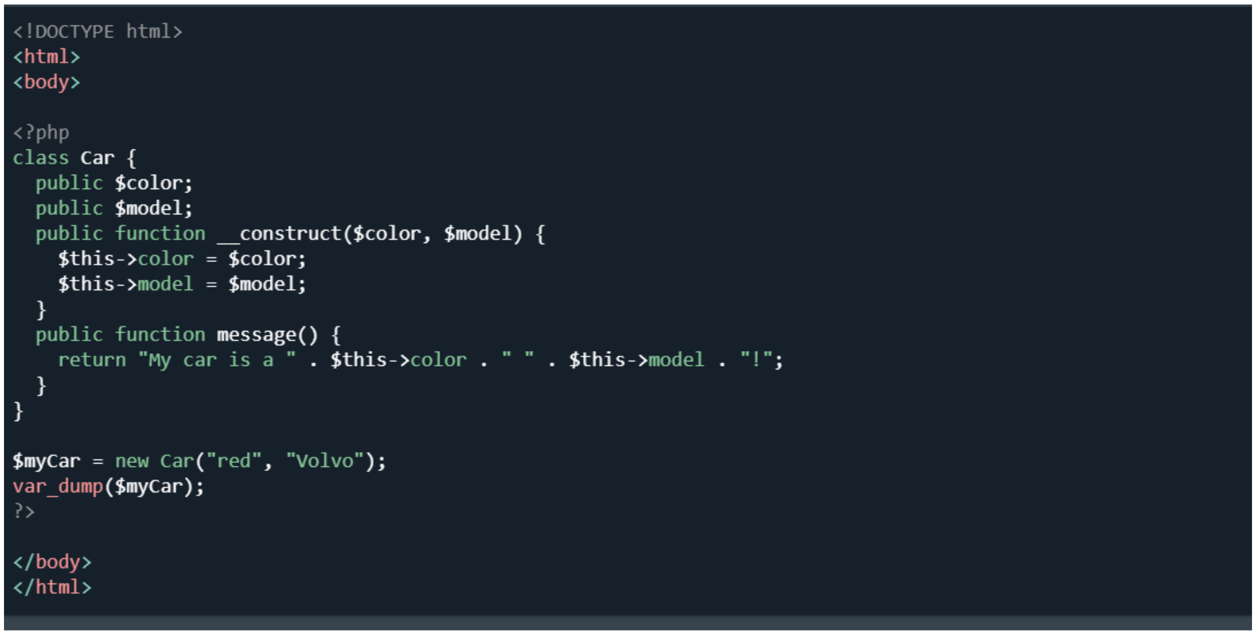
A class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Let's assume we have a class named Car that can have properties like model, color, etc. We can define variables like $model, $color, and so on, to hold the values of these properties.

When the individual objects (Volvo, BMW, Toyota, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

If you create a __construct() function, PHP will automatically call this function when you create an object from a class.



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

<?php
class Car {
  public $color;
  public $model;
  public function __construct($color, $model) {
    $this->color = $color;
    $this->model = $model;
  }
  public function message() {
    return "My car is a " . $this->color . " " . $this->model . "!";
  }
}

$myCar = new Car("red", "Volvo");
var_dump($myCar);
?>

</body>
</html>
```

```
object(Car)#1 (2) { ["color"]=> string(3) "red" ["model"]=> string(5) "Volvo" }
```

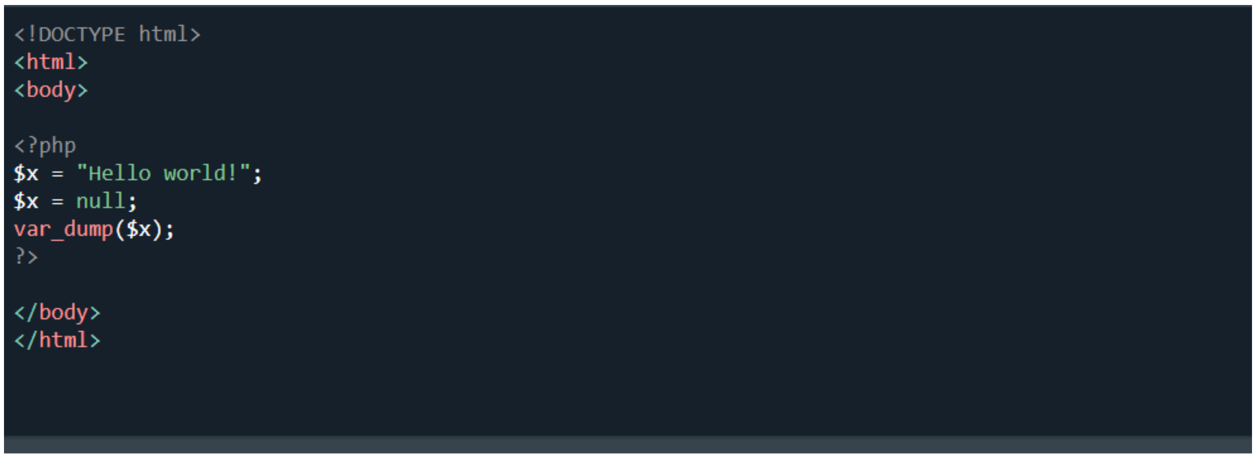
PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

Tip: If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:



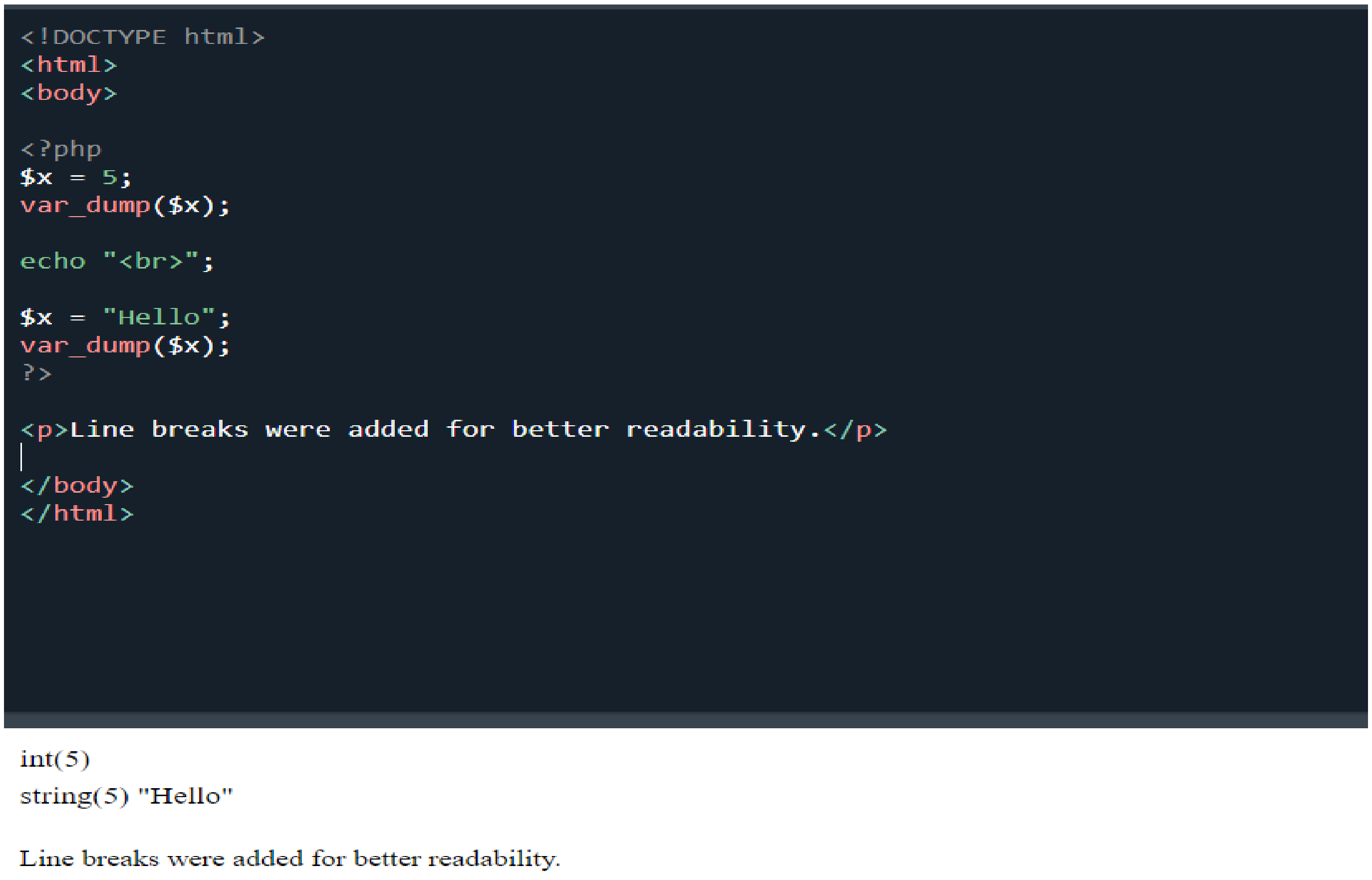
```
<!DOCTYPE html>
<html>
<body>

<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>

</body>
</html>
```

```
NULL
```

Change Data Type

If you assign an integer value to a variable, the type will automatically be an integer.

If you assign a string to the same variable, the type will change to a string



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

<?php
$x = 5;
var_dump($x);

echo "<br>";

$x = "Hello";
var_dump($x);
?>

<p>Line breaks were added for better readability.</p>

</body>
</html>
```

```
int(5)
string(5) "Hello"

Line breaks were added for better readability.
```

PHP Variables

In PHP, a variable is declared using a $ sign followed by the variable name. Here, some important points to know about variables:

- As PHP is a loosely typed language, so we do not need to declare the data types of the variables. It automatically analyzes the values and makes conversions to its correct datatype.
- After declaring a variable, it can be reused throughout the code.
- Assignment Operator (=) is used to assign the value to a variable.

Syntax of declaring a variable in PHP is given below:

```
$variablename=value;
```

Rules for declaring PHP variable:

- A variable must start with a dollar ($) sign, followed by the variable name.
- It can only contain alpha-numeric character and underscore (A-z, 0-9, _).
- A variable name must start with a letter or underscore (_) character.
- A PHP variable name cannot contain spaces.
- One thing to be kept in mind that the variable name cannot start with a number or special symbols.
- PHP variables are case-sensitive, so $name and $NAME both are treated as different variable.

PHP Variable: Declaring string, integer, and float

Let's see the example to store string, integer, and float values in PHP variables.

File: variable1.php

```
<?php
$str="hello string";
$x=200;
$y=44.6;
echo "string is: $str <br/>";
echo "integer is: $x <br/>";
echo "float is: $y <br/>";
?>
```

```
string is: hello string
integer is: 200
float is: 44.6
```

File: variable2.php

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

PHP Variable: case sensitive

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc.

File: variable3.php

```
<?php
$color="red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
```

Output:

```
My car is red
Notice: Undefined variable: COLOR in C:\wamp\www\variable.php on line 4
My house is
Notice: Undefined variable: coLOR in C:\wamp\www\variable.php on line 5
My boat is
```

PHP Variable: Rules

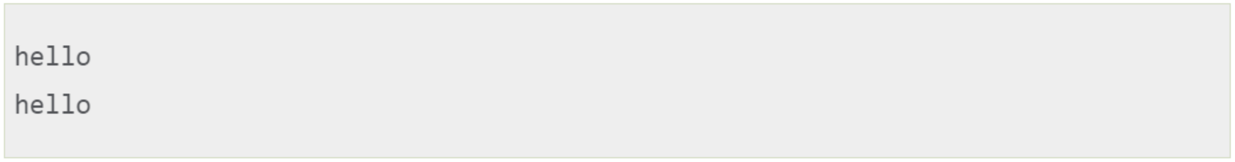
PHP variables must start with letter or underscore only.

PHP variable can't be start with numbers and special symbols.

File: variablevalid.php

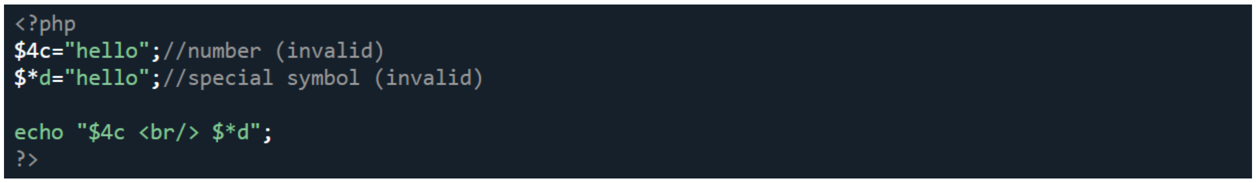
```
<?php
$a="hello";//letter (valid)
$_b="hello";//underscore (valid)

echo "$a <br/> $_b";
?>
```

Output:



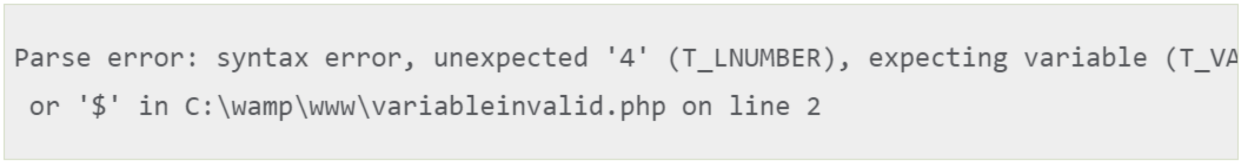
```
hello
hello
```

File: variableinvalid.php



```
<?php
$4c="hello";//number (invalid)
$*d="hello";//special symbol (invalid)

echo "$4c <br/> $*d";
?>
```

Output:



```
Parse error: syntax error, unexpected '4' (T_LNUMBER), expecting variable (T_VA
 or '$' in C:\wamp\www\variableinvalid.php on line 2
```

PHP: Loosely typed language

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

PHP Constants

PHP constants are name or identifier that can't be changed during the execution of the script except for magic constants, which are not really constants. PHP constants can be defined by 2 ways:

1. Using define() function
2. Using const keyword

Constants are similar to the variable except once they defined, they can never be undefined or changed. They remain constant across the entire program. PHP constants follow the same PHP variable rules. For example, it can be started with a letter or underscore only.

Conventionally, PHP constants should be defined in uppercase letters.

Note: Unlike variables, constants are automatically global throughout the script.

PHP constant: define()

Use the define() function to create a constant. It defines constant at run time. Let's see the syntax of define() function in PHP.

```
define(name, value, case-insensitive)
```

name: It specifies the constant name.

value: It specifies the constant value.

case-insensitive: Specifies whether a constant is case-insensitive. Default value is false. It means it is case sensitive by default.

Let's see the example to define PHP constant using define().

File: constant1.php

```
<?php
define("MESSAGE","Hello PHP ");
echo MESSAGE;
?>
```

```
Hello PHP
```

Create a constant with case-insensitive name:

File: constant2.php

```
<?php
define("MESSAGE","Hello PHP",true);//not case sensitive
echo MESSAGE, "</br>";
echo message;
?>
```

```
Hello PHP
Hello PHP
```

File: constant3.php

```
<?php
define("MESSAGE","Hello PHP",false);//case sensitive
echo MESSAGE;
echo message;
?>
```

Hello PHPmessage

```
Notice: Use of undefined constant message - assumed 'message'
in C:\wamp\www\vconstant3.php on line 4
message
```

PHP constant: const keyword

PHP introduced a keyword const to create a constant. The const keyword defines constants at compile time. It is a language construct, not a function. The constant defined using const keyword are case-sensitive.

File: constant4.php

```
<?php
const MESSAGE="Hello PHP const";
echo MESSAGE;
?>
```

Hello PHP const

```
Notice: Use of undefined constant message - assumed 'message'
in C:\wamp\www\vconstant3.php on line 4
message
```

Constant() function

There is another way to print the value of constants using constant() function instead of using the echo statement.

Syntax:

The syntax for the following constant function:

```
constant (name)
```

File: constant5.php

```php
<?php
    define("MSG", "Hello World");
    echo MSG, "</br>";
    echo constant("MSG");
    //both are similar
?>
```

```
Hello World
Hello World
```

Constant vs Variables

Constant	**Variables**
Once the constant is defined, it can never be redefined.	A variable can be undefined as well as redefined easily.
A constant can only be defined using define() function. It cannot be defined by any simple assignment.	A variable can be defined by simple assignment (=) operator.
There is no need to use the dollar ($) sign before constant during the assignment.	To declare a variable, always use the dollar ($) sign before the variable.
Constants do not follow any variable scoping rules, and they can be defined and accessed anywhere.	Variables can be declared anywhere in the program, but they follow variable scoping rules.
Constants are the variables whose values can't be changed throughout the program.	The value of the variable can be changed.
By default, constants are global.	Variables can be local, global, or static.

PHP Operators

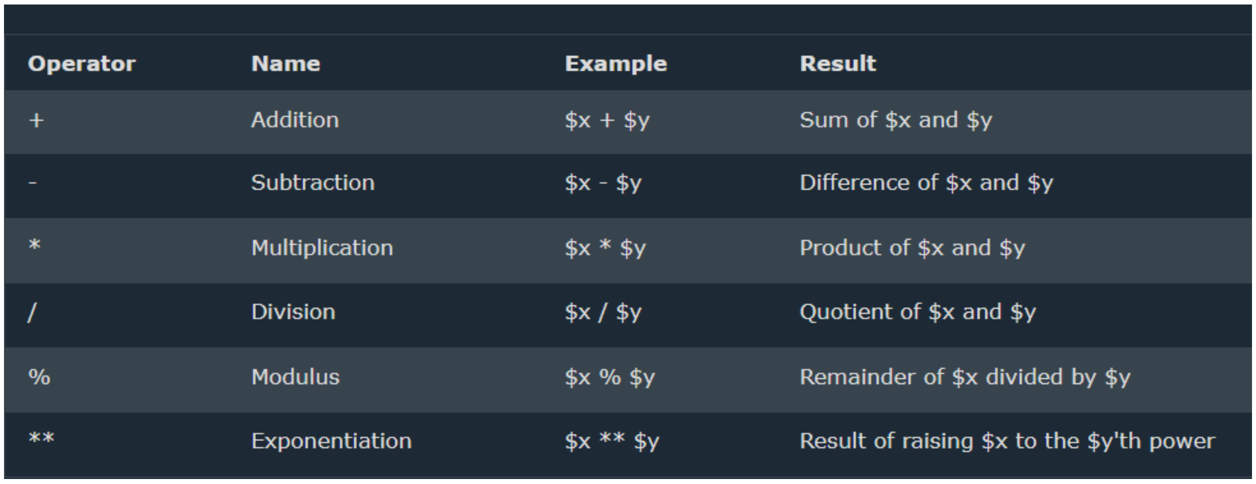
Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

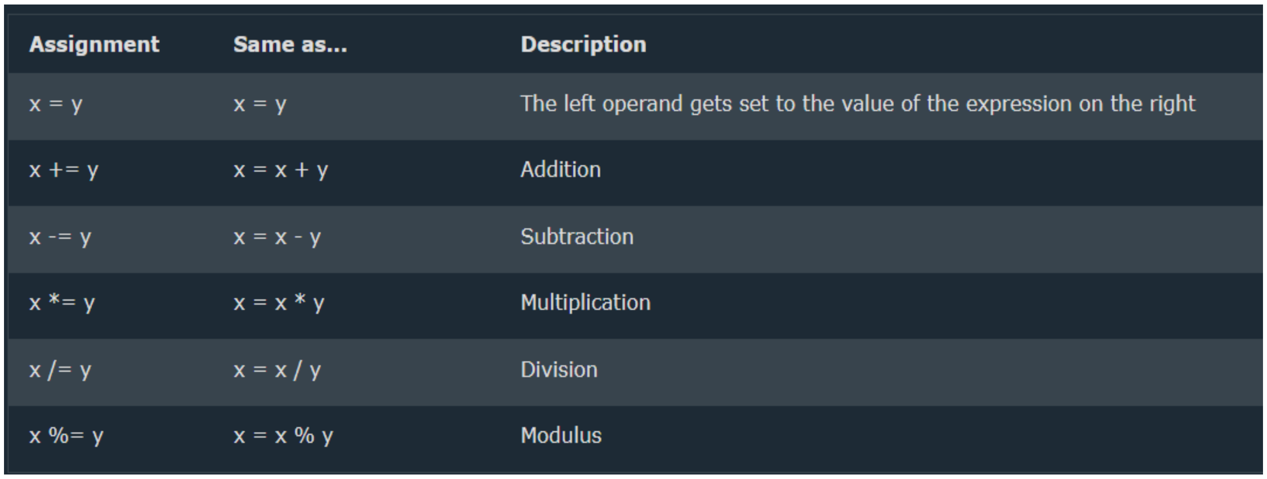


Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

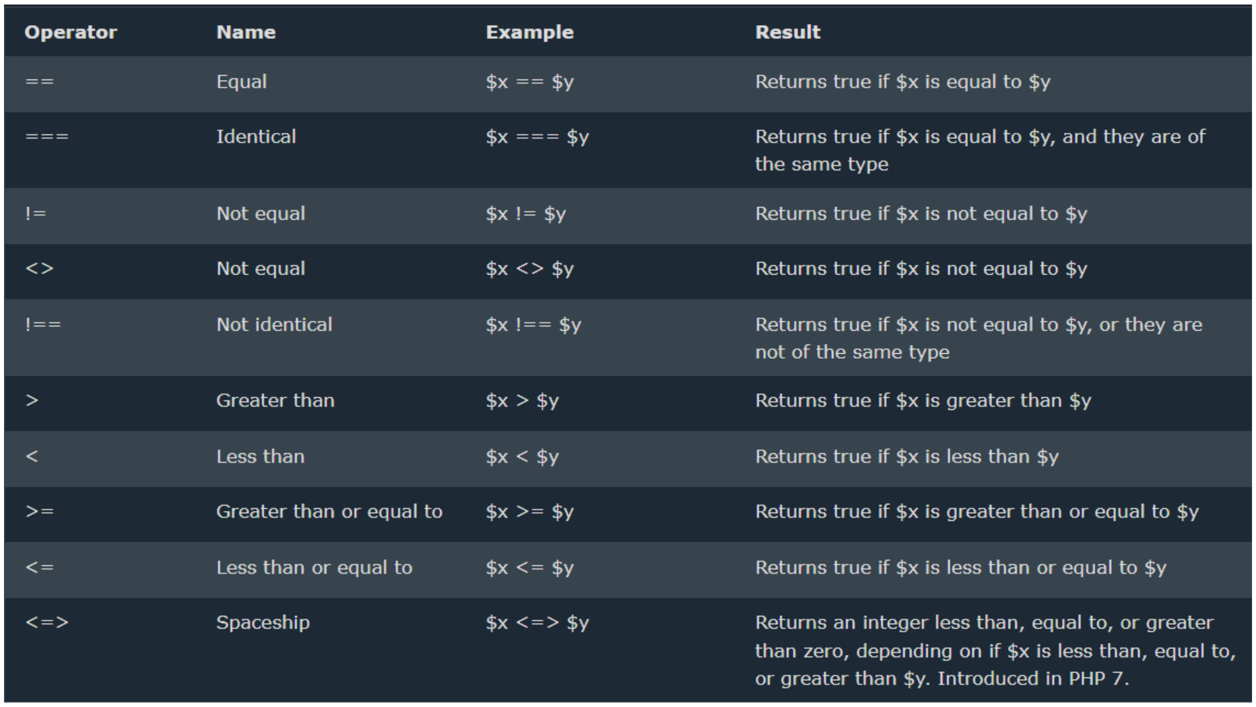


Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

The PHP comparison operators are used to compare two values (number or string):

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

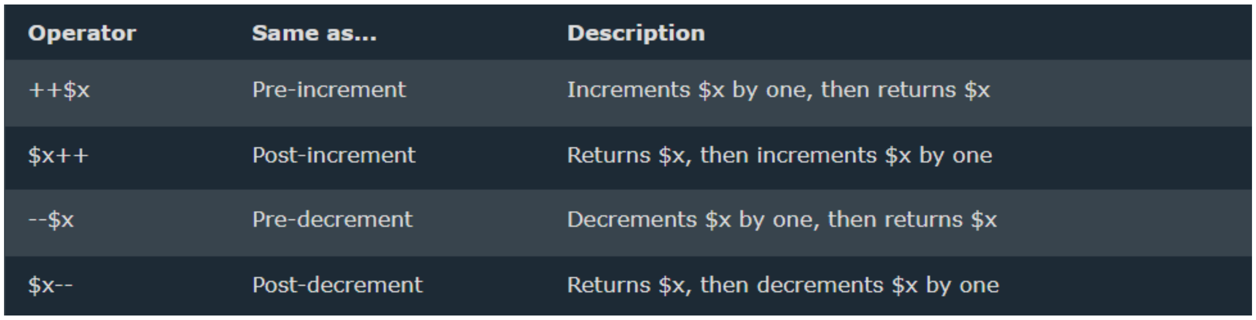


Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.

PHP Increment / Decrement Operators

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

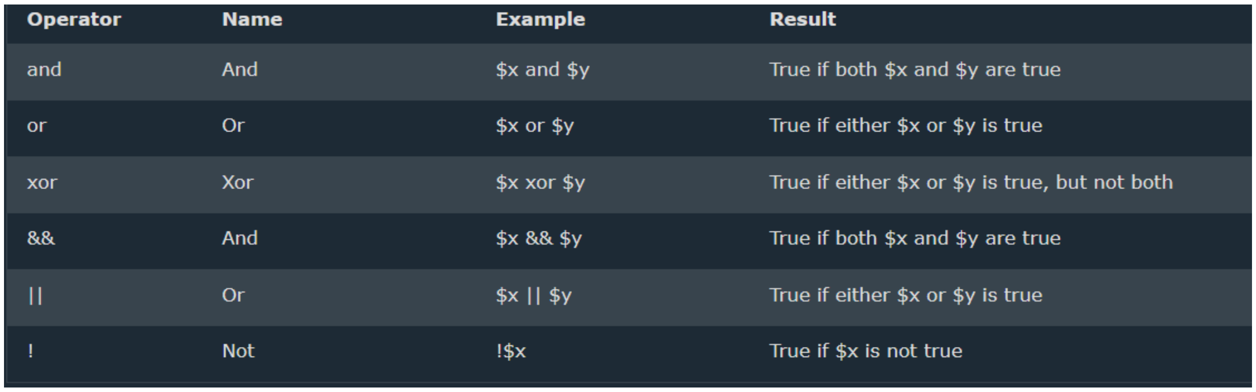
The PHP decrement operators are used to decrement a variable's value.



Operator	Same as...	Description
++$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

PHP Logical Operators

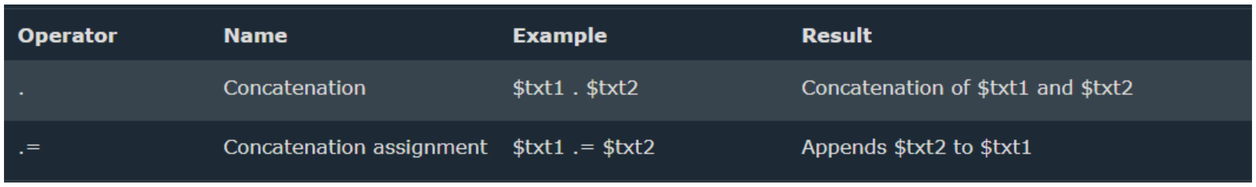
The PHP logical operators are used to combine conditional statements.



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

PHP String Operators

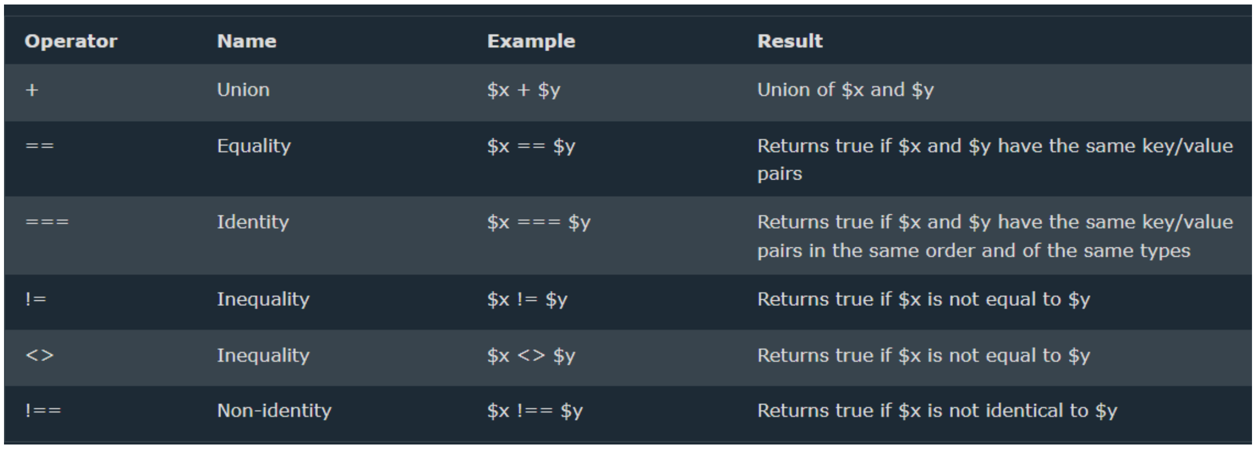
PHP has two operators that are specially designed for strings



Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

PHP Array Operators

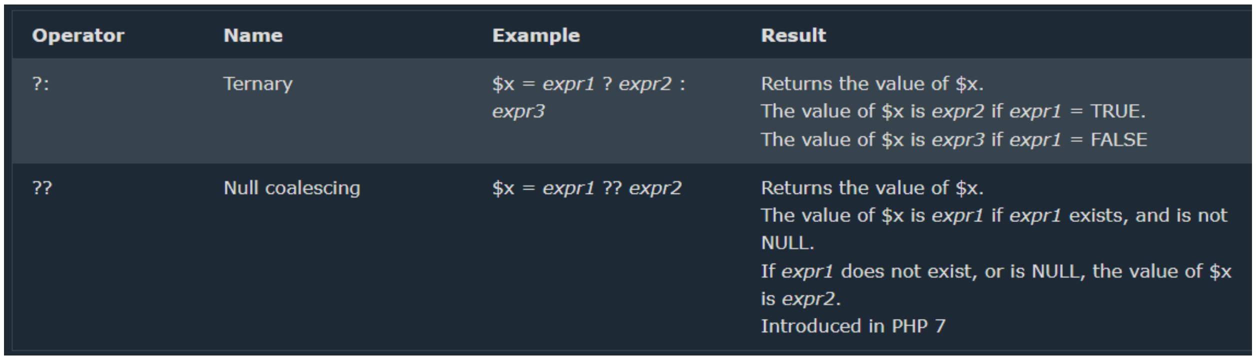
The PHP array operators are used to compare arrays.



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

PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions:



Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

User Interface Components

In PHP, you typically handle the backend logic and data processing, while user interface components like paginators, popovers, progress bars, spinners, tables, toasts, and tooltips are usually implemented using HTML, CSS, and JavaScript. However, you can integrate PHP with these components to dynamically generate or manipulate them based on server-side data.

Here's a brief overview of how you can integrate PHP with these UI components:

- Paginators
- Popovers
- Progress and Spinner
- Tables, Toasts, Tooltips in php

Paginators

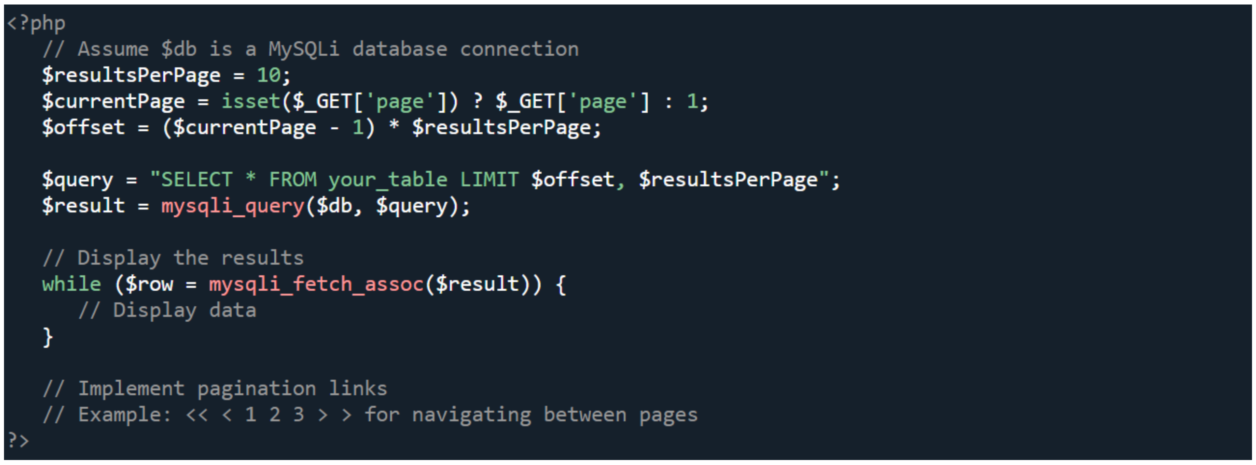
Pagination in PHP refers to the process of dividing a large set of data into smaller, more manageable sections called "pages." It is commonly used in web applications to display a limited number of records or results per page, allowing users to navigate through the data easily.

Pagination is essential when dealing with large data sets because displaying all the records on a single page can lead to performance issues and an overwhelming user interface. By implementing pagination, you can improve the user experience and optimize the performance of your application.

PHP program to pagination

- Create a new PHP file in your working location and save it with your desired name.
- Here we are using student data table with 121 records in localhost database.

Example:
Consider a scenario where you fetch data from a MySQL database and implement pagination:



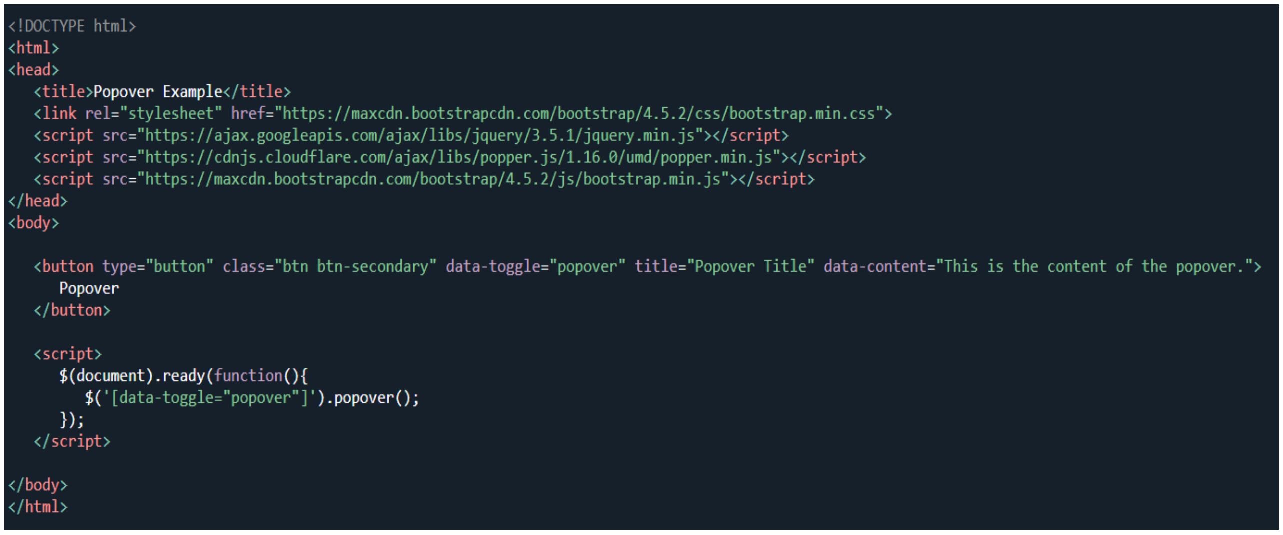
```php
<?php
   // Assume $db is a MySQLi database connection
   $resultsPerPage = 10;
   $currentPage = isset($_GET['page']) ? $_GET['page'] : 1;
   $offset = ($currentPage - 1) * $resultsPerPage;

   $query = "SELECT * FROM your_table LIMIT $offset, $resultsPerPage";
   $result = mysqli_query($db, $query);

   // Display the results
   while ($row = mysqli_fetch_assoc($result)) {
      // Display data
   }

   // Implement pagination links
   // Example: << < 1 2 3 > > for navigating between pages
?>
```

Popovers

Popovers are interactive UI components that display additional information or options when triggered. They are commonly used for tooltips or contextual information.

Example:

Using Bootstrap, you can create a simple popover:



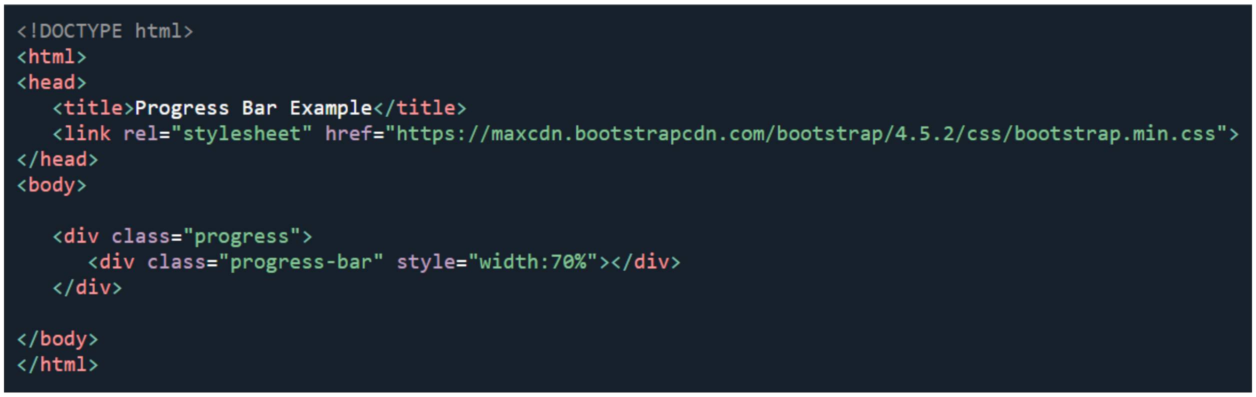
```html
<!DOCTYPE html>
<html>
<head>
   <title>Popover Example</title>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
   <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>
<body>

   <button type="button" class="btn btn-secondary" data-toggle="popover" title="Popover Title" data-content="This is the content of the popover.">
      Popover
   </button>

   <script>
      $(document).ready(function(){
         $('[data-toggle="popover"]').popover();
      });
   </script>

</body>
</html>
```

Progress and Spinner

Progress bars and spinners provide visual feedback on the status of ongoing processes. They are particularly useful when loading data or performing time-consuming operations.

Example:

Using Bootstrap for a progress bar:



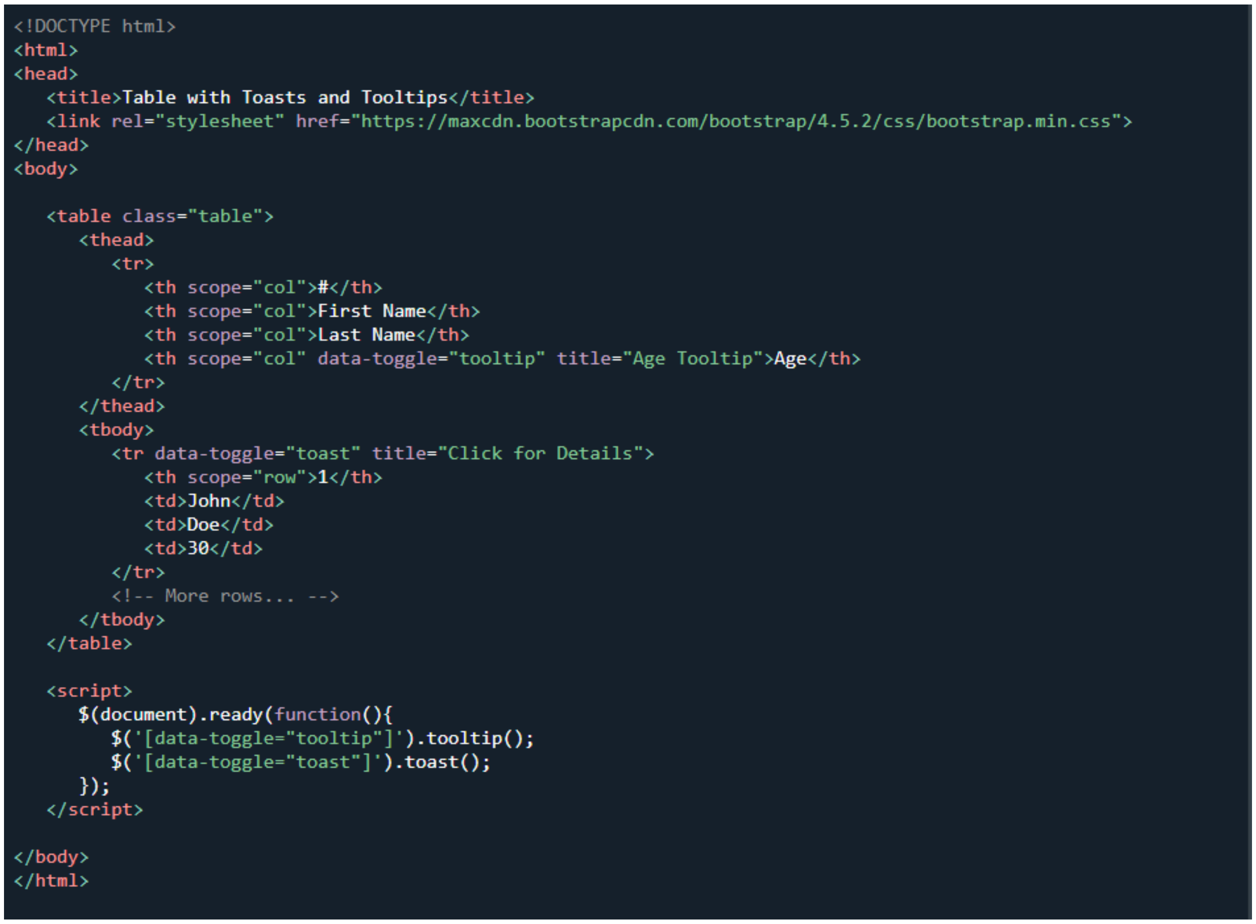
```html
<!DOCTYPE html>
<html>
<head>
   <title>Progress Bar Example</title>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>

   <div class="progress">
      <div class="progress-bar" style="width:70%"></div>
   </div>

</body>
</html>
```

Tables, Toasts, Tooltips

Tables are foundational for displaying structured data, while toasts and tooltips enhance user notifications and interactions.

Example:

Creating a table with toasts and tooltips:



```html
<!DOCTYPE html>
<html>
<head>
   <title>Table with Toasts and Tooltips</title>
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>

   <table class="table">
      <thead>
         <tr>
            <th scope="col">#</th>
            <th scope="col">First Name</th>
            <th scope="col">Last Name</th>
            <th scope="col" data-toggle="tooltip" title="Age Tooltip">Age</th>
         </tr>
      </thead>
      <tbody>
         <tr data-toggle="toast" title="Click for Details">
            <th scope="row">1</th>
            <td>John</td>
            <td>Doe</td>
            <td>30</td>
         </tr>
         <!-- More rows... -->
      </tbody>
   </table>

   <script>
      $(document).ready(function(){
         $('[data-toggle="tooltip"]').tooltip();
         $('[data-toggle="toast"]').toast();
      });
   </script>

</body>
</html>
```

These examples showcase professional and interactive UI components that enhance user experience. As you incorporate these elements into your PHP projects, remember to tailor them to suit your specific requirements.

PHP Conditional Events and Flow Control

Control structures in php are fundamental programming concepts that enable developers to write powerful and efficient code. They allow the program to execute specific instructions based on certain conditions. There are three primary types of control structures: selection, iteration, and sequence. The selection structure executes a set of instructions if a specific condition is met. The iteration structure is used when a set of instructions needs to be repeated multiple times based on a given condition. The sequence structure is used to execute instructions in sequential order. These structures are essential to writing efficient and reliable code.

Introduction

Control structures in php are an essential part of any programming language, including PHP. They are used to control the flow of a program by allowing the programmer to specify which statements should be executed under certain conditions. Understanding control structures in PHP is crucial for any aspiring PHP developer, as they form the backbone of many programming tasks.

Control structures in PHP are used to make decisions based on conditions, loop through a set of instructions multiple times, and break out of a loop or switch statement. The most common control structures used in PHP are if-else statements, loops such as for and while loops, and switch statements.

By using if-else statements, a programmer can check a certain condition and execute a block of code if the condition is true or execute a different block of code if the condition is false. Loops allow a programmer to repeat a set of instructions multiple times, either for a specified number of times or until a certain condition is met. Switch statements provide an alternative to multiple if-else statements, allowing a programmer to check multiple conditions with a single statement. Control structures are a vital part of any programming language, and understanding them is essential for any PHP developer. By mastering the use of control structures, a programmer can create more efficient and effective PHP code, leading to better software applications.

Conditional statements

Conditional statements are used to perform different actions based on different conditions.

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

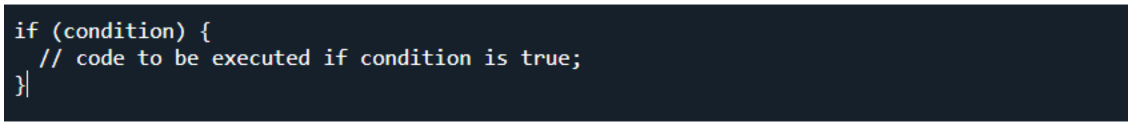
In PHP we have the following conditional statements:

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif...else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

PHP - The if Statement

The if statement executes some code if one condition is true.

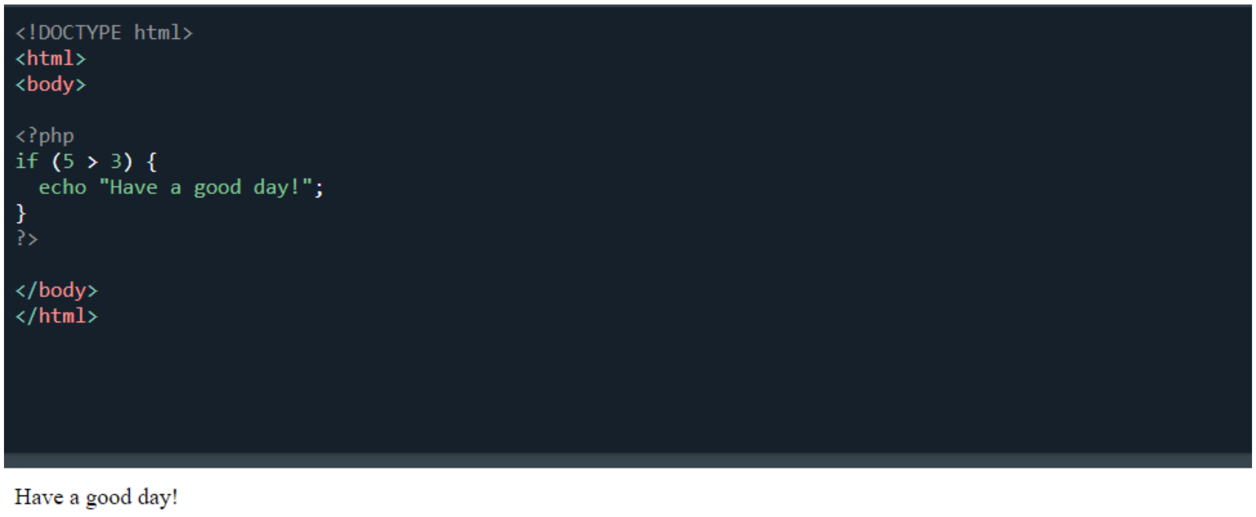
Syntax



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

Example

Output "Have a good day!" if 5 is larger than 3:



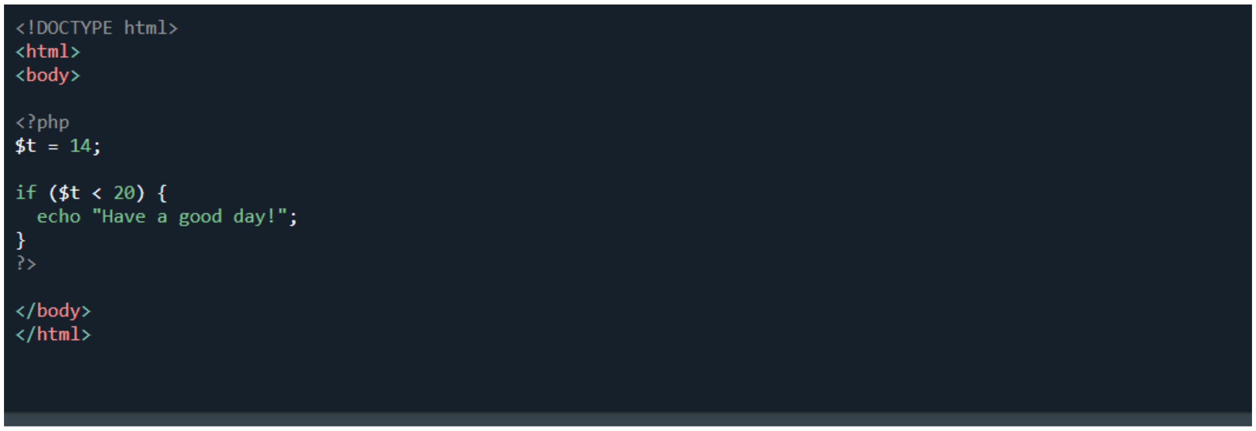
```
<!DOCTYPE html>
<html>
<body>

<?php
if (5 > 3) {
  echo "Have a good day!";
}
?>

</body>
</html>
```

```
Have a good day!
```

We can also use variables in the if statement:

Example

Output "Have a good day!" if $t is less than 20:



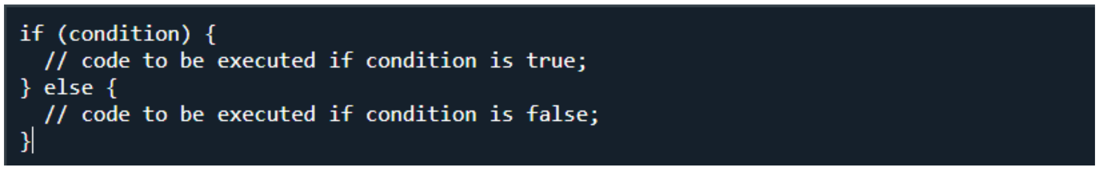
```
<!DOCTYPE html>
<html>
<body>

<?php
$t = 14;

if ($t < 20) {
  echo "Have a good day!";
}
?>

</body>
</html>
```

```
Have a good day!
```

PHP - The if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

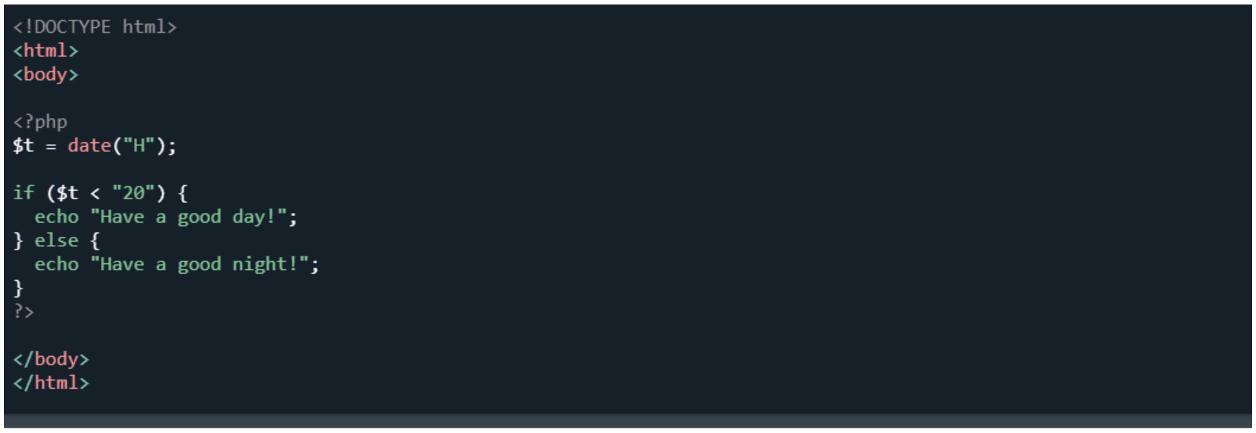
Syntax



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

Example

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:



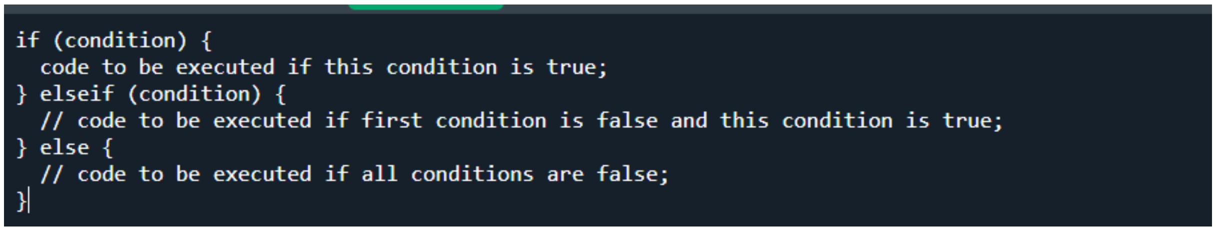
```
<!DOCTYPE html>
<html>
<body>

<?php
$t = date("H");

if ($t < "20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>

</body>
</html>
```

```
Have a good day!
```

PHP - The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

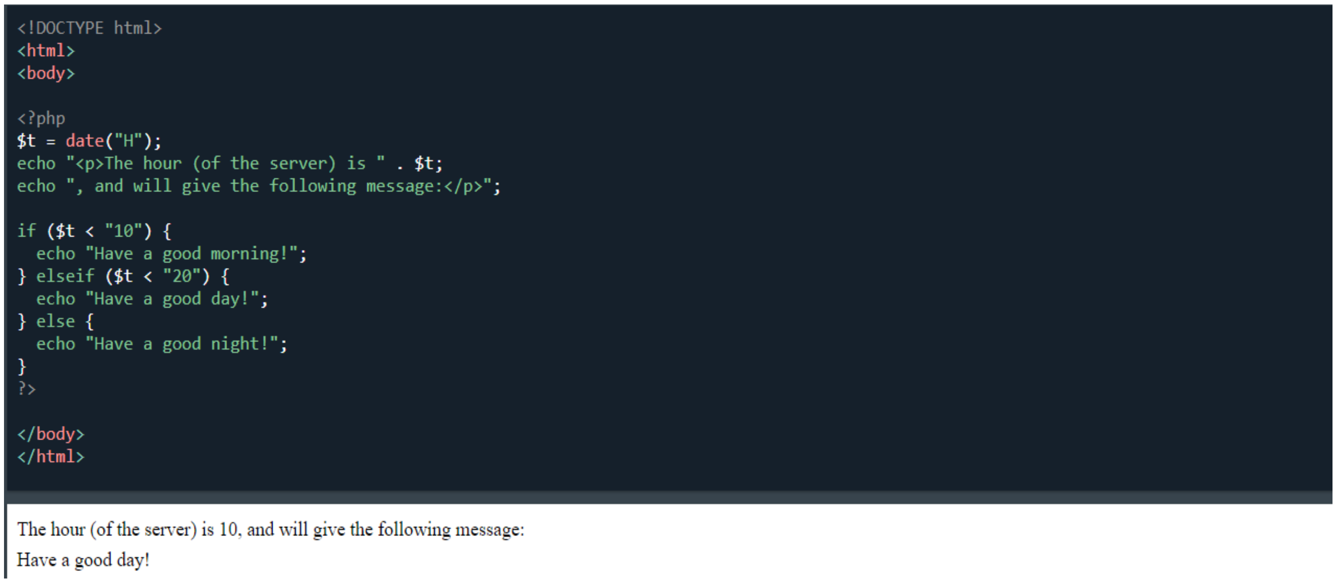
Syntax



```
if (condition) {
  code to be executed if this condition is true;
} elseif (condition) {
  // code to be executed if first condition is false and this condition is true;
} else {
  // code to be executed if all conditions are false;
}
```

Example

Output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":



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

<?php
$t = date("H");
echo "<p>The hour (of the server) is " . $t;
echo ", and will give the following message:</p>";

if ($t < "10") {
  echo "Have a good morning!";
} elseif ($t < "20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>

</body>
</html>
```

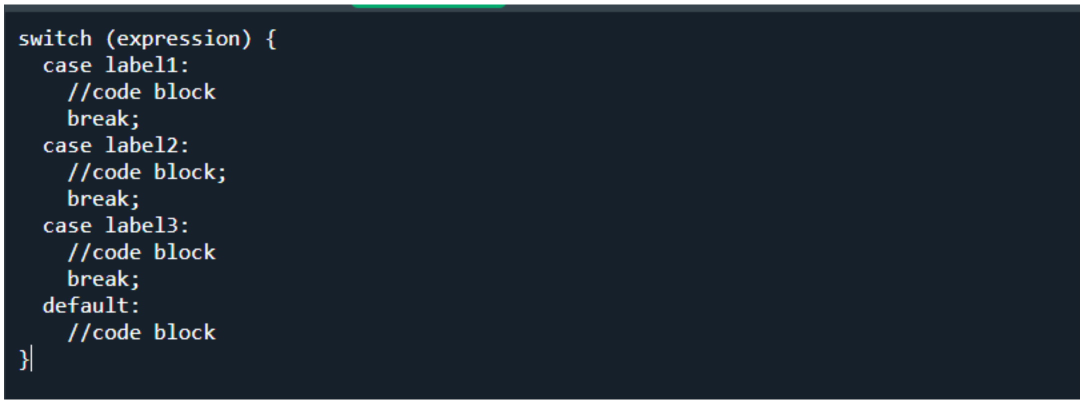
```
The hour (of the server) is 10, and will give the following message:
Have a good day!
```

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

The PHP switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

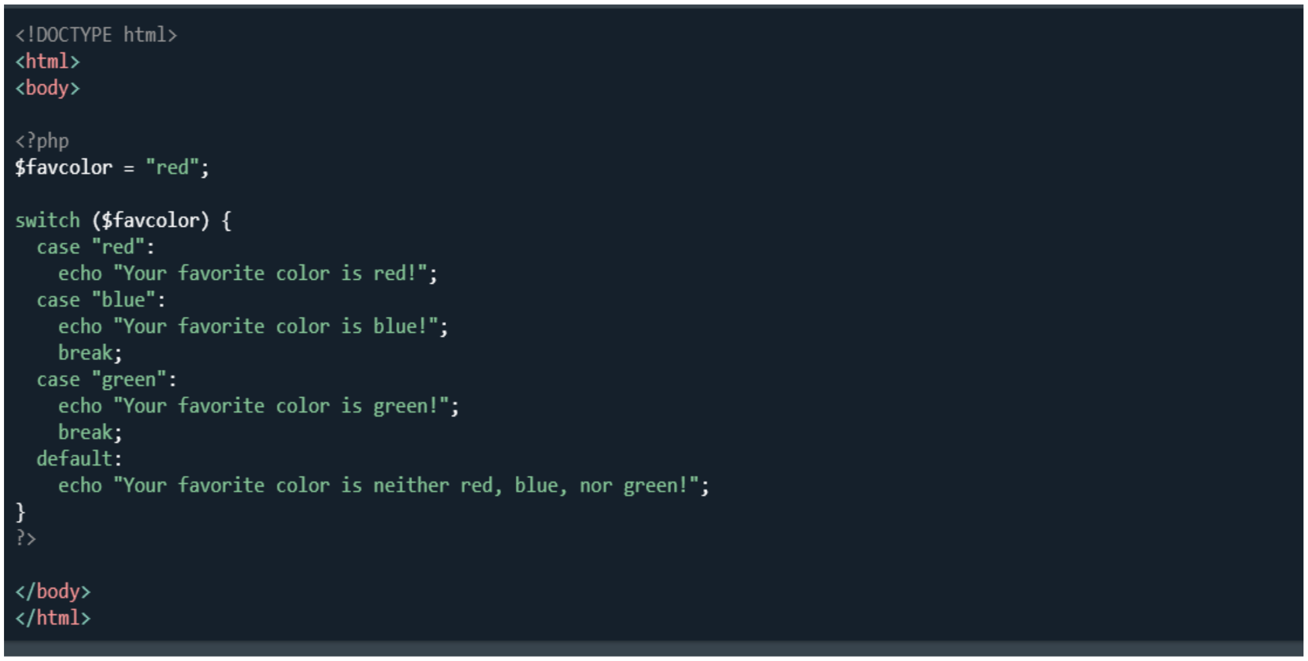


```
switch (expression) {
  case label1:
    //code block
    break;
  case label2:
    //code block;
    break;
  case label3:
    //code block
    break;
  default:
    //code block
}
```

This is how it works:

- The *expression* is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- The break keyword breaks out of the switch block
- The default code block is executed if there is no match

Example



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

<?php
$favcolor = "red";

switch ($favcolor) {
  case "red":
    echo "Your favorite color is red!";
  case "blue":
    echo "Your favorite color is blue!";
    break;
  case "green":
    echo "Your favorite color is green!";
    break;
  default:
    echo "Your favorite color is neither red, blue, nor green!";
}
?>

</body>
</html>
```

```
Your favorite color is red!Your favorite color is blue!
```

The break Keyword

When PHP reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code, and no more cases are tested.

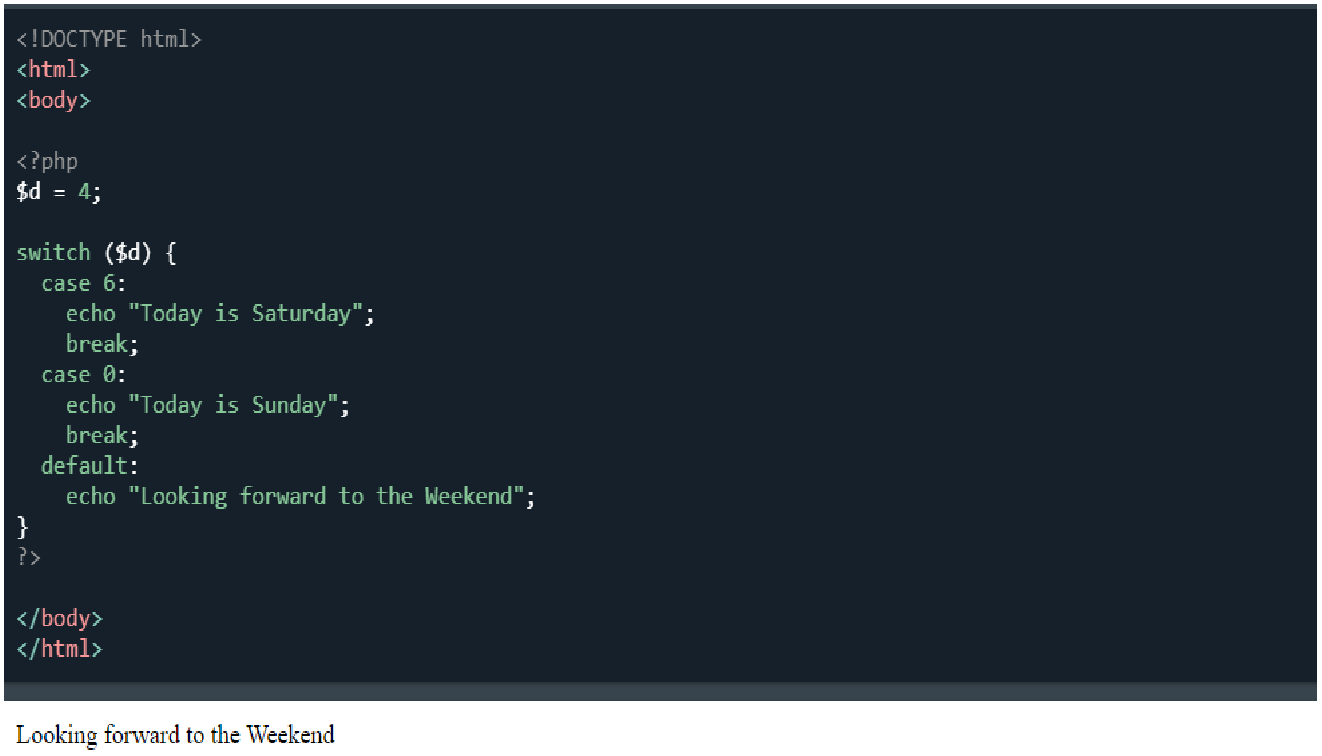
The last block does not need a break, the block breaks (ends) there anyway.

Warning: If you omit the break statement in a case that is not the last, and that case gets a match, the next case will also be executed even if the evaluation does not match the case!

Example

What happens if we remove the break statement from case "red"?

$favcolor is red, so the code block from case "red" is executed, but since it has no break statement, the code block from case "blue" will also be executed:



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

<?php
$d = 4;

switch ($d) {
  case 6:
    echo "Today is Saturday";
    break;
  case 0:
    echo "Today is Sunday";
    break;
  default:
    echo "Looking forward to the Weekend";
}
?>

</body>
</html>
```

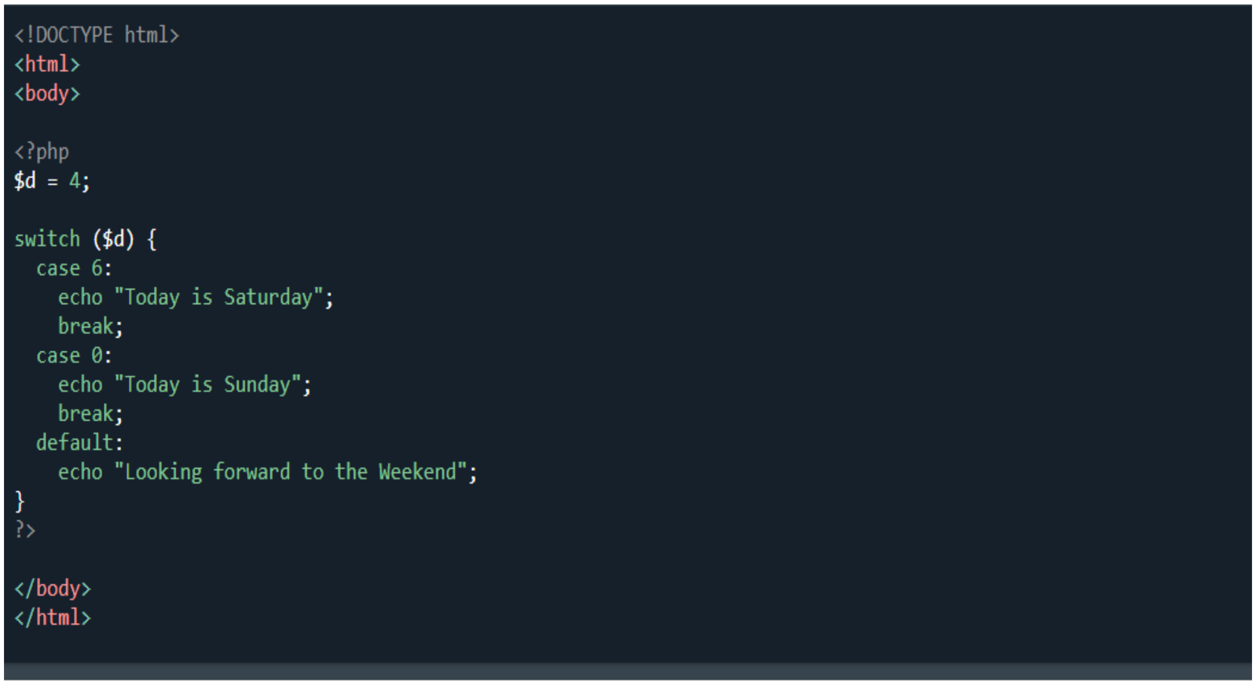
```
Looking forward to the Weekend
```

The default Keyword

The default keyword specifies the code to run if there is no case match:

Example

If no cases get a match, the default block is executed:



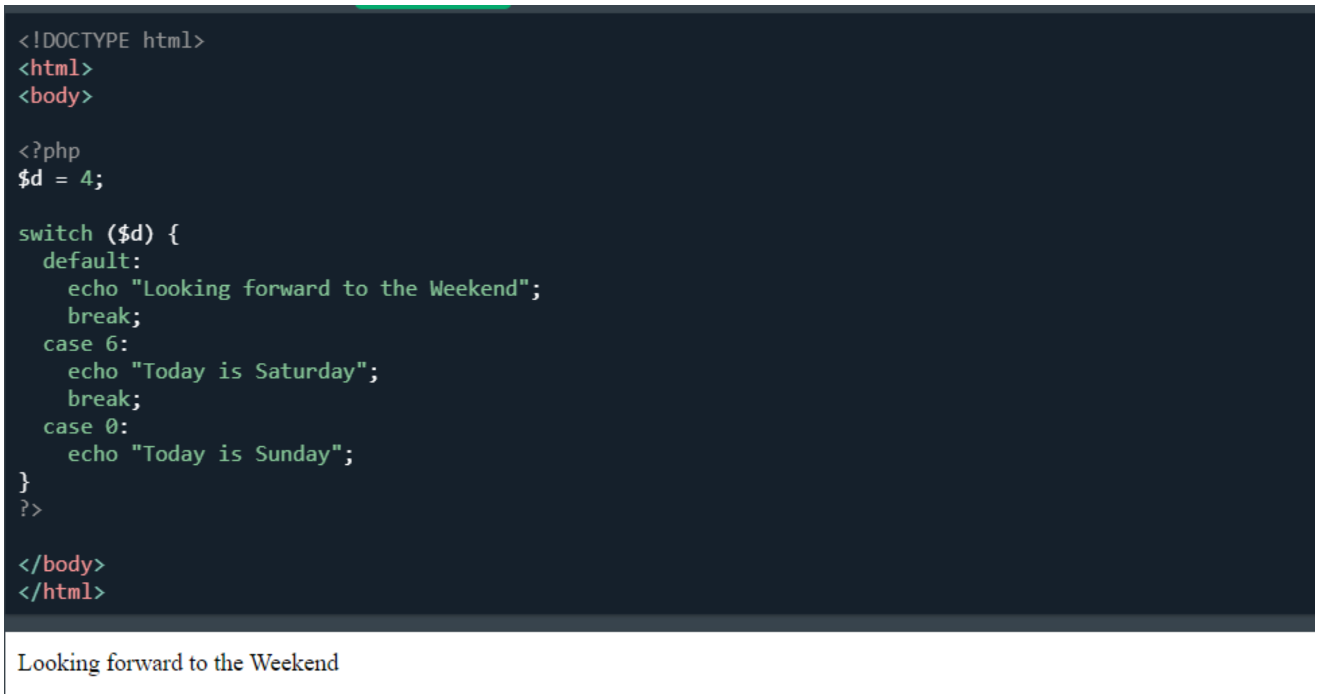
```
<!DOCTYPE html>
<html>
<body>

<?php
$d = 4;

switch ($d) {
  case 6:
    echo "Today is Saturday";
    break;
  case 0:
    echo "Today is Sunday";
    break;
  default:
    echo "Looking forward to the Weekend";
}
?>

</body>
</html>
```

```
Looking forward to the Weekend
```

The default case does not have to be the last case in a switch block:

Example

Putting the default block elsewhere than at the end of the switch block is allowed, but not recommended.



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

<?php
$d = 4;

switch ($d) {
  default:
    echo "Looking forward to the Weekend";
    break;
  case 6:
    echo "Today is Saturday";
    break;
  case 0:
    echo "Today is Sunday";
}
?>

</body>
</html>
```

```
Looking forward to the Weekend
```

Note: If default is not the last block in the switch block, remember to end the default block with a break statement.

Looping In PHP

In the following chapters you will learn how to repeat code by using loops in PHP.

PHP Loops

Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

- while - loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array

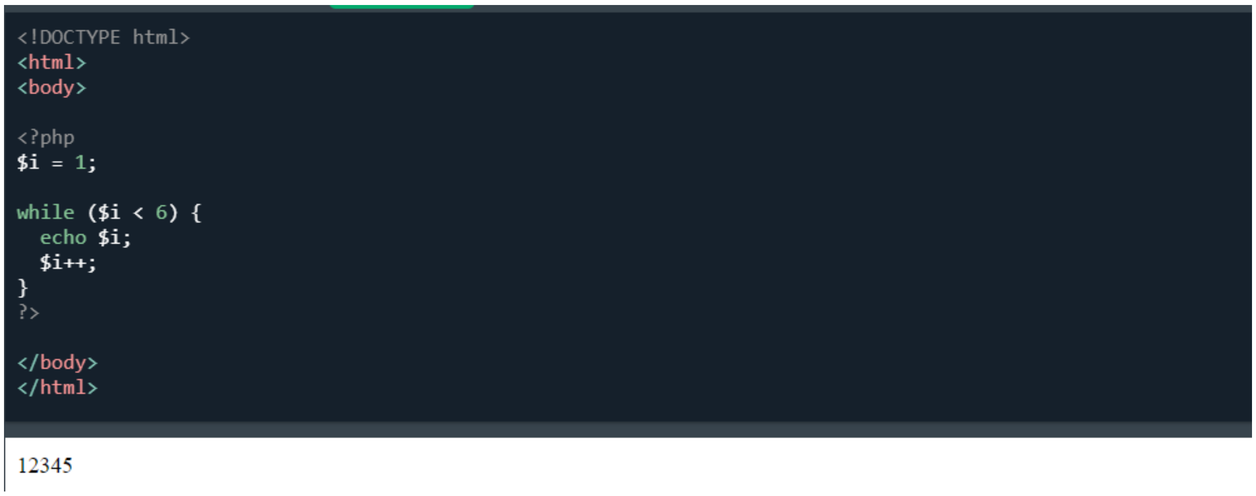
The following chapters will explain and give examples of each loop type.

PHP while Loop

The while loop - Loops through a block of code as long as the specified condition is true.

Example:

Print $i as long as $i is less than 6:



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

<?php
$i = 1;

while ($i < 6) {
  echo $i;
  $i++;
}
?>

</body>
</html>
```

```
12345
```

Note: remember to increment $i, or else the loop will continue forever.

The while loop does not run a specific number of times, but checks after each iteration if the condition is still true.

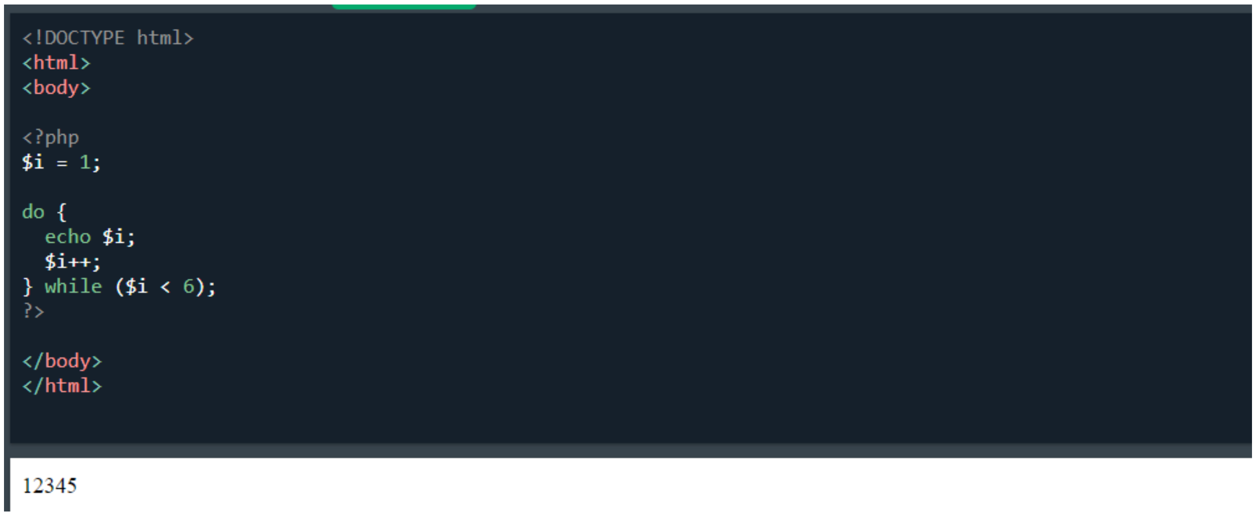
The condition does not have to be a counter, it could be the status of an operation or any condition that evaluates to either true or false.

PHP do while Loop

The do...while loop - Loops through a block of code once, and then repeats the loop as long as the specified condition is true.

Example:

Print $i as long as $i is less than 6:


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

<?php
$i = 1;

do {
  echo $i;
  $i++;
} while ($i < 6);
?>

</body>
</html>
```

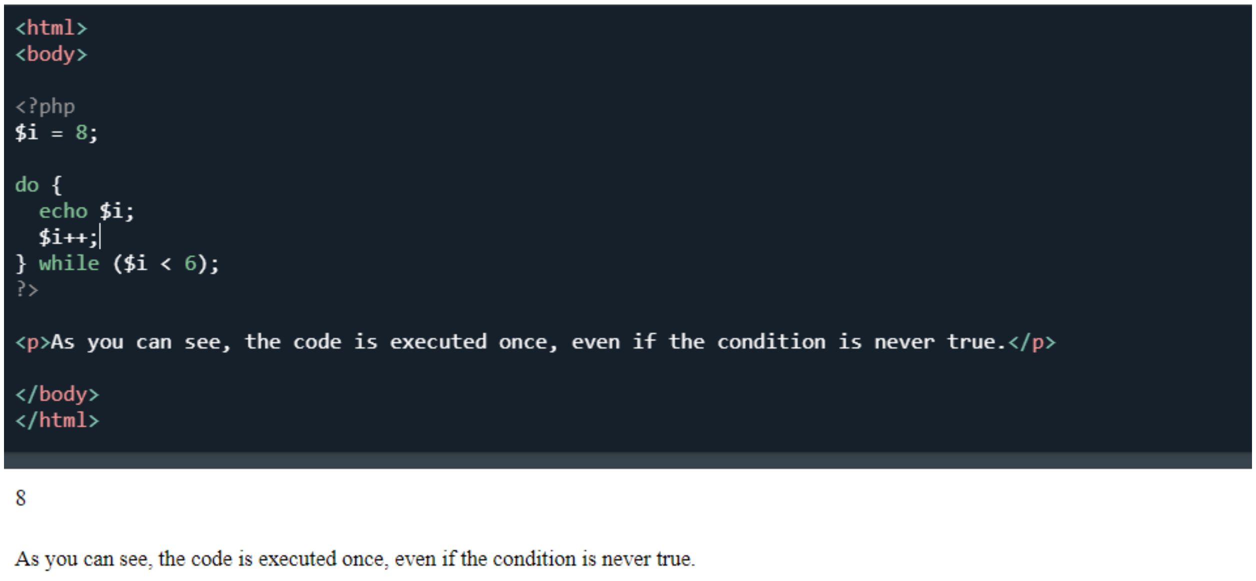
```
12345
```

Note: In a do...while loop the condition is tested AFTER executing the statements within the loop. This means that the do...while loop will execute its statements at least once, even if the condition is false. See example below.

Let us see what happens if we set the $i variable to 8 instead of 1, before execute the same do...while loop again:

Example:

Set $i = 8, then print $i as long as $i is less than 6:

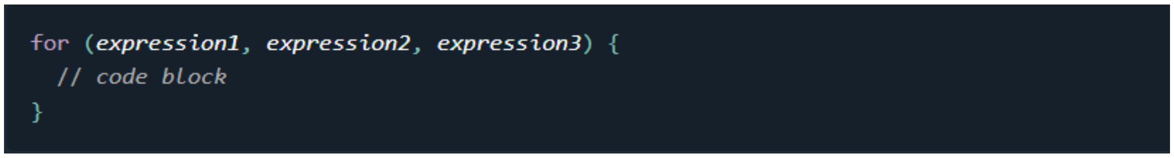

```
<html>
<body>

<?php
$i = 8;

do {
  echo $i;
  $i++;
} while ($i < 6);
?>

<p>As you can see, the code is executed once, even if the condition is never true.</p>

</body>
</html>
```

```
8

As you can see, the code is executed once, even if the condition is never true.
```

PHP for Loop

The for loop - Loops through a block of code a specified number of times.

Syntax



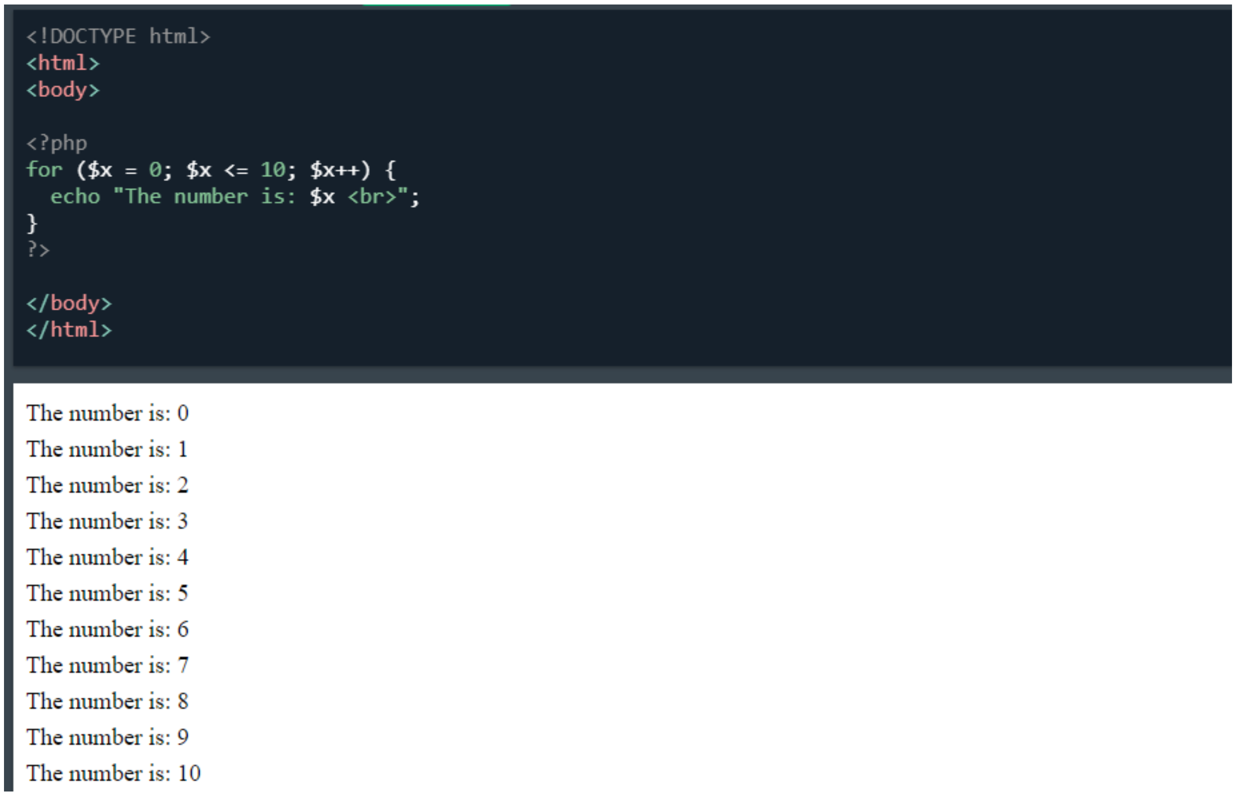
```
for (expression1, expression2, expression3) {
  // code block
}
```

This is how it works:

- expression1 is evaluated once
- expression2 is evaluated before each iterarion
- expression3 is evaluated after each iterarion

Example:

Print the numbers from 0 to 10:



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

<?php
for ($x = 0; $x <= 10; $x++) {
  echo "The number is: $x <br>";
}
?>

</body>
</html>
```

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
The number is: 10
```

Example Explained

1. The first expression, $x = 0;, is evaluated once and sets a counter to 0.

2. The second expression, $x <= 10;, is evaluated before each iteration, and the code block is only executed if this expression evaluates to true. In this example the expression is true as long as $x is less than, or equal to, 10.
3. The third expression, $x++;, is evaluated after each iteration, and in this example, the expression increases the value of $x by one at each iteration.

PHP foreach Loop

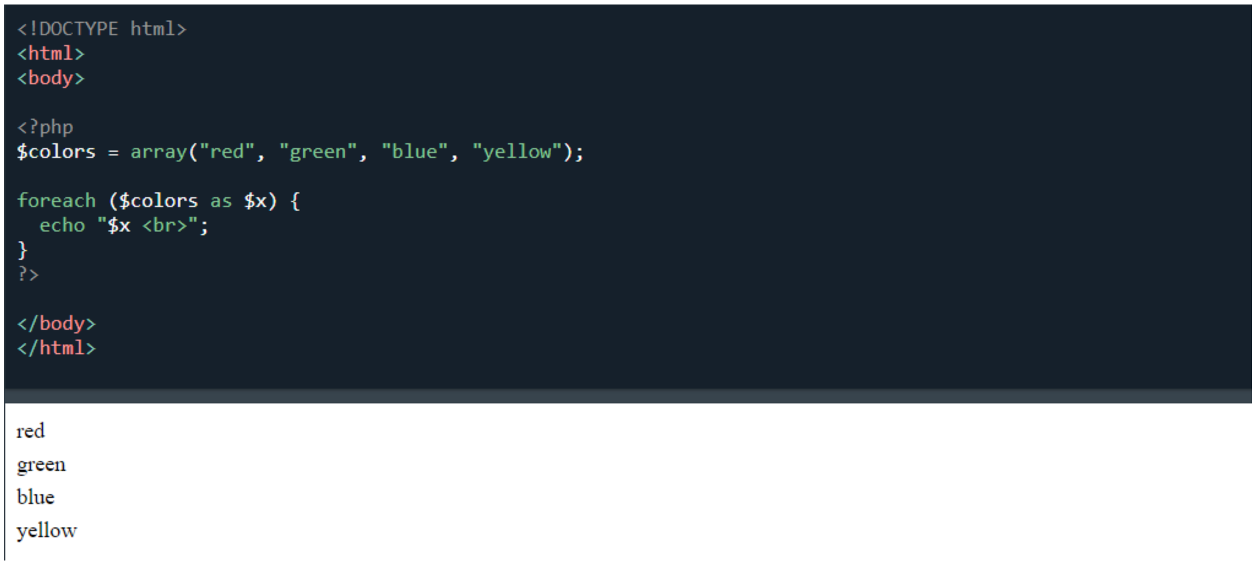
The foreach loop - Loops through a block of code for each element in an array or each property in an object.

The foreach Loop on Arrays

The most common use of the foreach loop, is to loop through the items of an array.

Example:

Loop through the items of an indexed array:



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

<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {
  echo "$x <br>";
}
?>

</body>
</html>
```

```
red
green
blue
yellow
```

For every loop iteration, the value of the current array element is assigned to the variabe $x. The iteration continues until it reaches the last array element.

Keys and Values

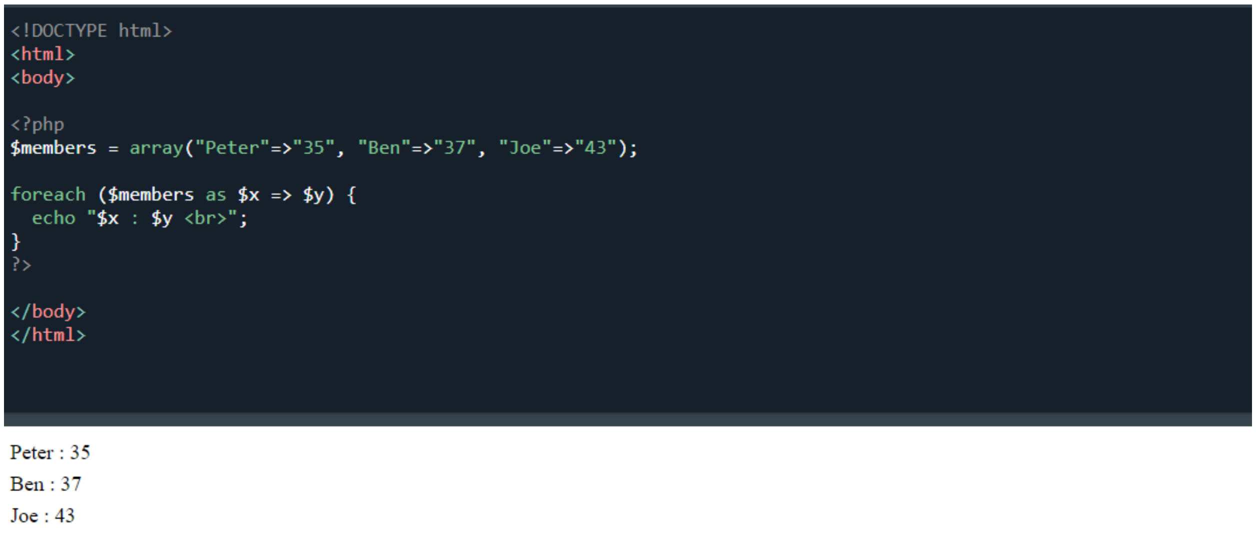
The array above is an indexed array, where the first item has the key 0, the second has the key 1, and so on.

Associative arrays are different, associative arrays use named keys that you assign to them, and when looping through associative arrays, you might want to keep the key as well as the value.

This can be done by specifying both the key and value in the foreach defintition, like this:

Example:

Print both the key and the value from the $members array:



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

<?php
$members = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach ($members as $x => $y) {
  echo "$x : $y <br>";
}
?>

</body>
</html>
```

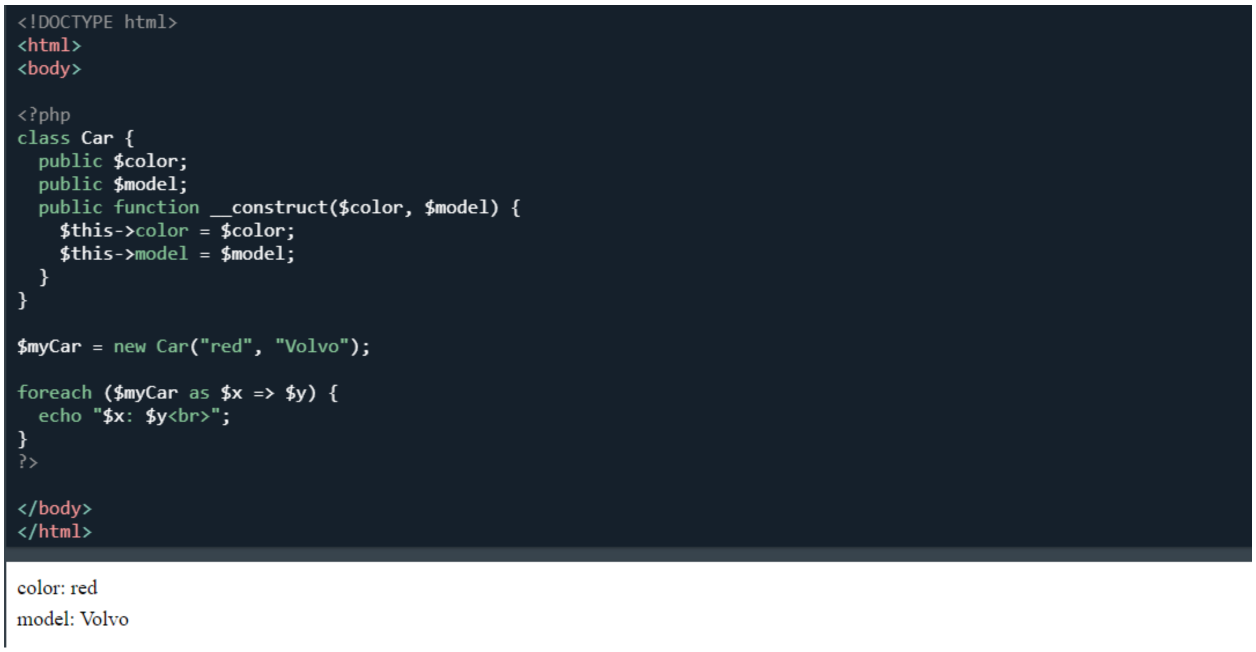
```
Peter : 35
Ben : 37
Joe : 43
```

The foreach Loop on Objects

The foreach loop can also be used to loop through properties of an object:

Example:

Print the property names and values of the $myCar object:



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

<?php
class Car {
  public $color;
  public $model;
  public function __construct($color, $model) {
    $this->color = $color;
    $this->model = $model;
  }
}

$myCar = new Car("red", "Volvo");

foreach ($myCar as $x => $y) {
  echo "$x: $y<br>";
}
?>

</body>
</html>
```

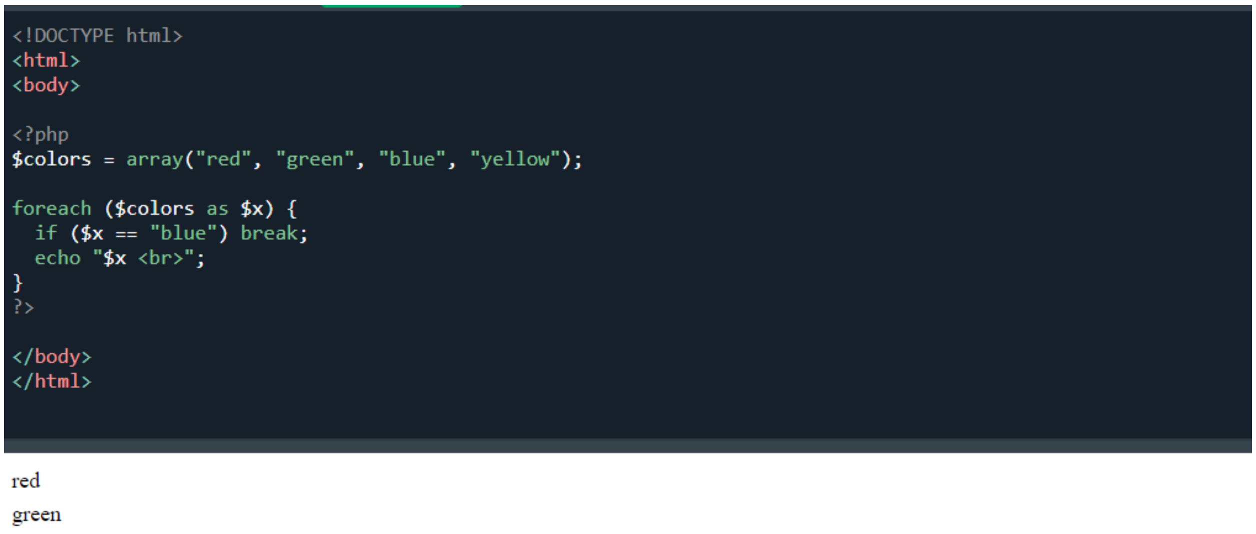
```
color: red
model: Volvo
```

The break Statement

With the break statement we can stop the loop even if it has not reached the end:

Example:

Stop the loop if $x is "blue":



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

<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {
  if ($x == "blue") break;
  echo "$x <br>";
}
?>

</body>
</html>
```

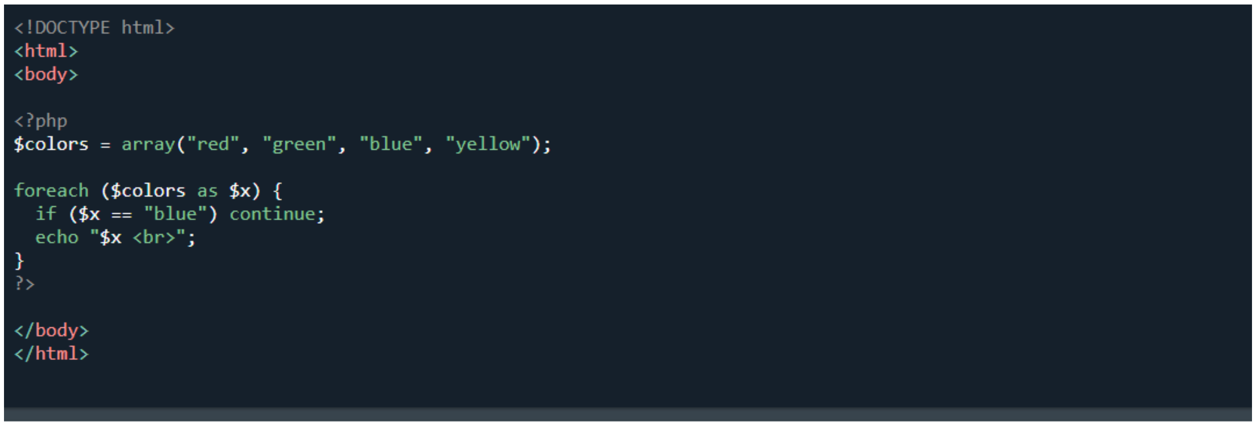
```
red
green
```

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

Example:

top, and jump to the next iteration if $x is "blue":



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

<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {
  if ($x == "blue") continue;
  echo "$x <br>";
}
?>

</body>
</html>
```

```
red
green
yellow
```

Functions in PHP

In this section, we'll explore the fundamental concepts of functions in PHP. Understanding the role of functions in programming and how they contribute to code organization and reusability is crucial. We'll cover:

- What are Functions: Explanation of the concept of functions in programming.
- Why Use Functions: Benefits of using functions, such as code modularity and maintainability.
- Function Syntax: An overview of the basic structure of PHP functions

Introduction to Functions

In modern programming, functions play a vital role in organizing code and promoting reusability. They encapsulate a set of instructions and can be called upon to perform a specific task. Let's explore the core concepts:

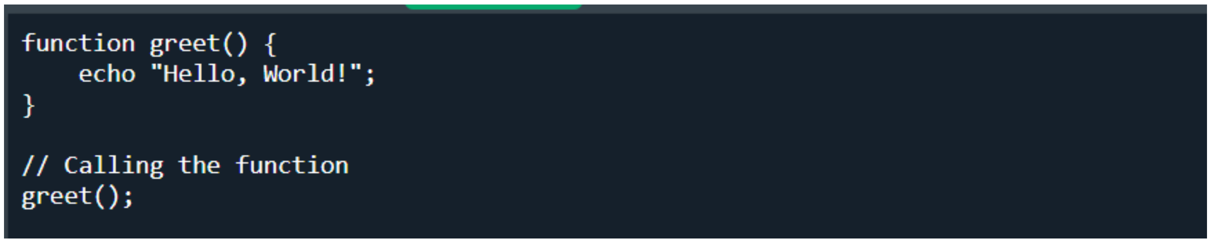
What are Functions:

A function is a block of code that performs a specific task. It enhances code modularity, making it easier to understand and maintain. Functions can be reused in different parts of a program.

The real power of PHP comes from its functions.

PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.

Example:



```
function greet() {
    echo "Hello, World!";
}

// Calling the function
greet();
```

Why Use Functions:

Code Organization: Functions help break down a program into smaller, manageable pieces.

Reusability: Once defined, functions can be reused, reducing redundancy in your code.

Function Syntax:

In PHP, a basic function looks like this:

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

Defining Functions:

Let's dive into the process of creating and using custom functions in PHP.

Function Declaration:

To declare a function, use the `function` keyword, followed by the function name, and a pair of parentheses.

```
function addNumbers($a, $b) {
    $sum = $a + $b;
    echo "Sum: $sum";
}
```

Note: A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

Tip: Give the function a name that reflects what the function does!

Call a Function

To call the function, just write its name followed by parentheses ():

Example:

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

<?php
function myMessage() {
  echo "Hello world!";
}

myMessage();
?>

</body>
</html>
```

```
Hello world!
```

In our example, we create a function named myMessage().

The opening curly brace { indicates the beginning of the function code, and the closing curly brace } indicates the end of the function.

The function outputs "Hello world!".

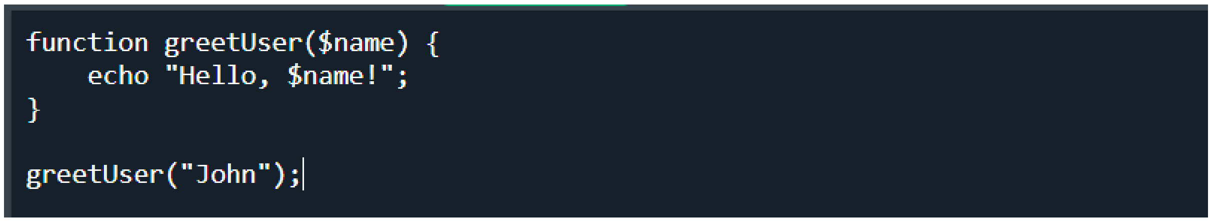
Function Name and Scope:

Function names are case-insensitive. However, it's good practice to follow a consistent naming convention. Functions have their own scope, meaning variables inside a function are not accessible outside of it.

Function Parameters:

Parameters allow us to pass data into functions. They are defined within the parentheses during function declaration.

Example:



```
function greetUser($name) {
    echo "Hello, $name!";
}

greetUser("John");
```

Function Body:

The function body contains the code to be executed when the function is called.

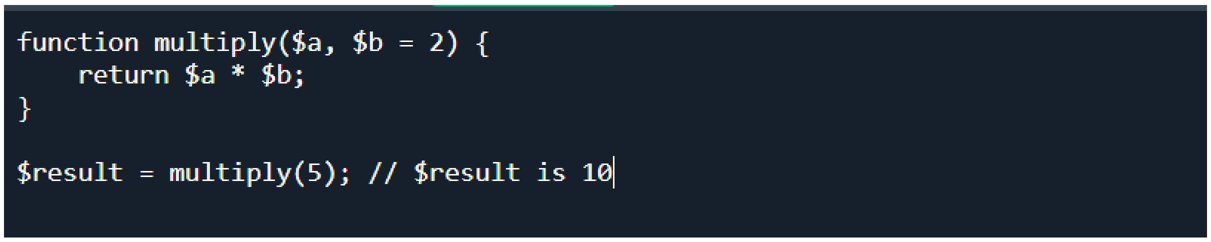
Parameters and Return Values

Functions can accept parameters and return values, making them versatile and adaptable to various scenarios.

There are three types of parameters:

- Required Parameters: Must be passed during the function call.
- Default Parameters: Have default values and can be omitted.
- Variable-length Parameter Lists: Allow an arbitrary number of arguments.

Example:

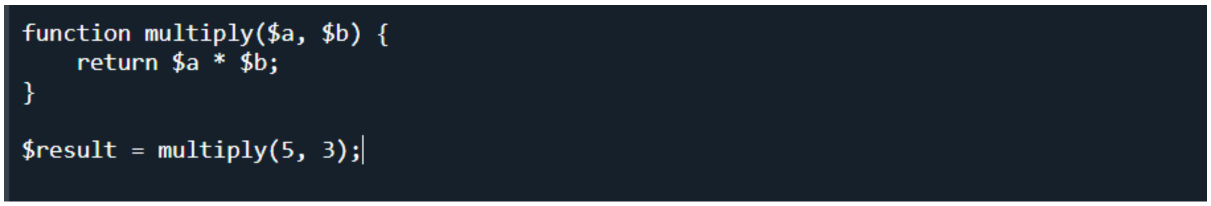


```
function multiply($a, $b = 2) {
    return $a * $b;
}

$result = multiply(5); // $result is 10
```

Required Parameters:

Definition: Required parameters are values that must be provided when calling a function. These parameters are essential for the function to execute properly.

Example:



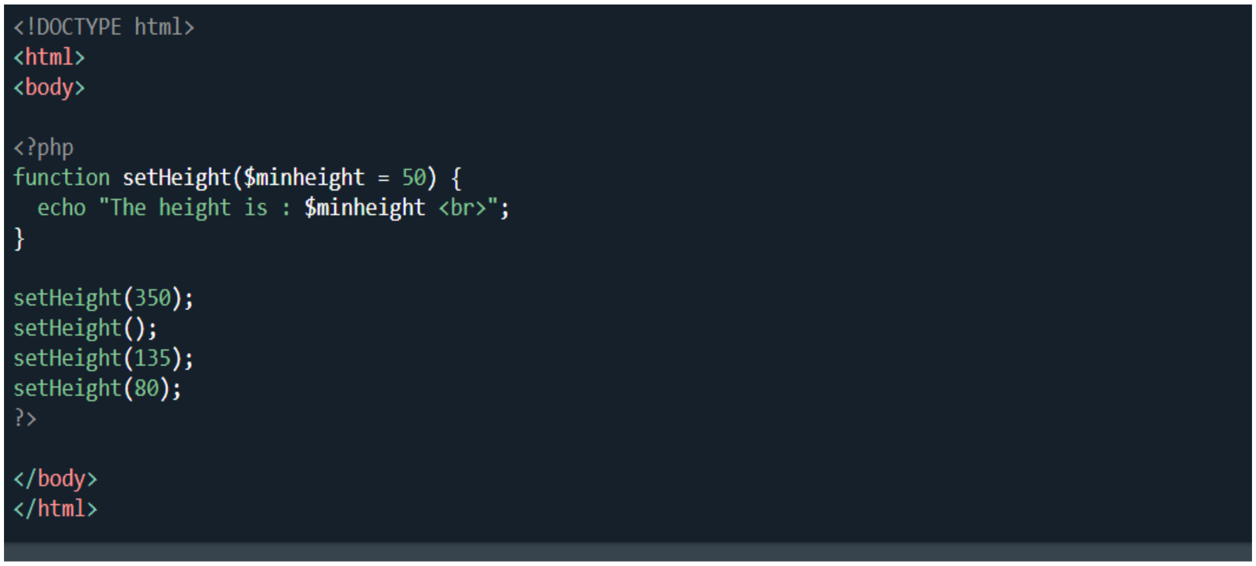
```
function multiply($a, $b) {
    return $a * $b;
}

$result = multiply(5, 3);
```

In this example, the multiply function requires two parameters,` $a `and `$b`. When calling the function, you must provide values for both parameters (5 and 3 in this case).

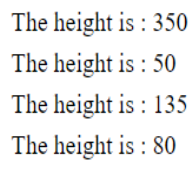
PHP Default Parameter Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example:



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

<?php
function setHeight($minheight = 50) {
  echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
?>

</body>
</html>
```

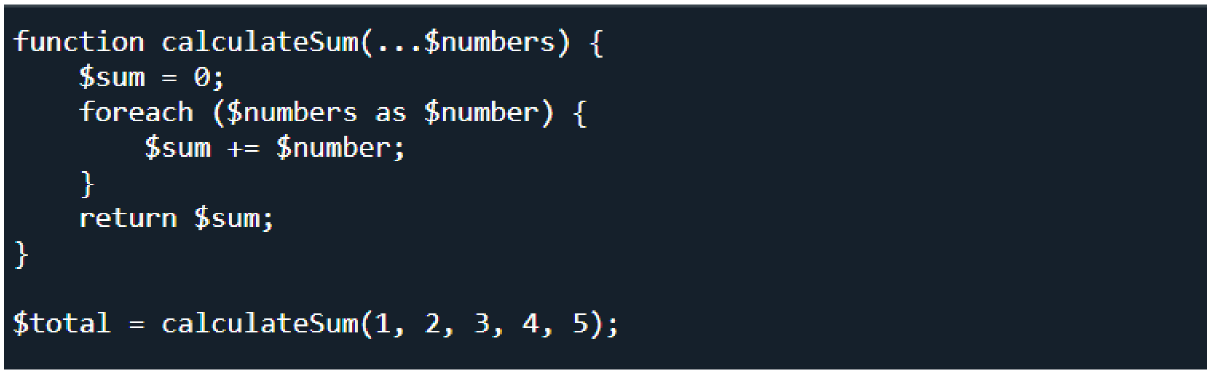


```
The height is : 350
The height is : 50
The height is : 135
The height is : 80
```

Variable-length Parameter Lists:

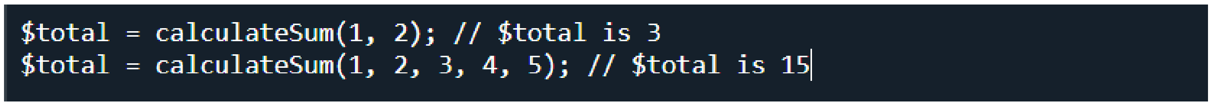
Definition: Variable-length parameter lists, also known as variadic functions, allow a function to accept an arbitrary number of arguments. This is useful when you don't know in advance how many arguments will be passed.

Example:



```
function calculateSum(...$numbers) {
    $sum = 0;
    foreach ($numbers as $number) {
        $sum += $number;
    }
    return $sum;
}

$total = calculateSum(1, 2, 3, 4, 5);
```

In this example, the `calculateSum` function accepts any number of arguments using the `...$numbers` syntax. It then calculates the sum of all provided numbers. You can call this function with different numbers of arguments, and it will work accordingly.



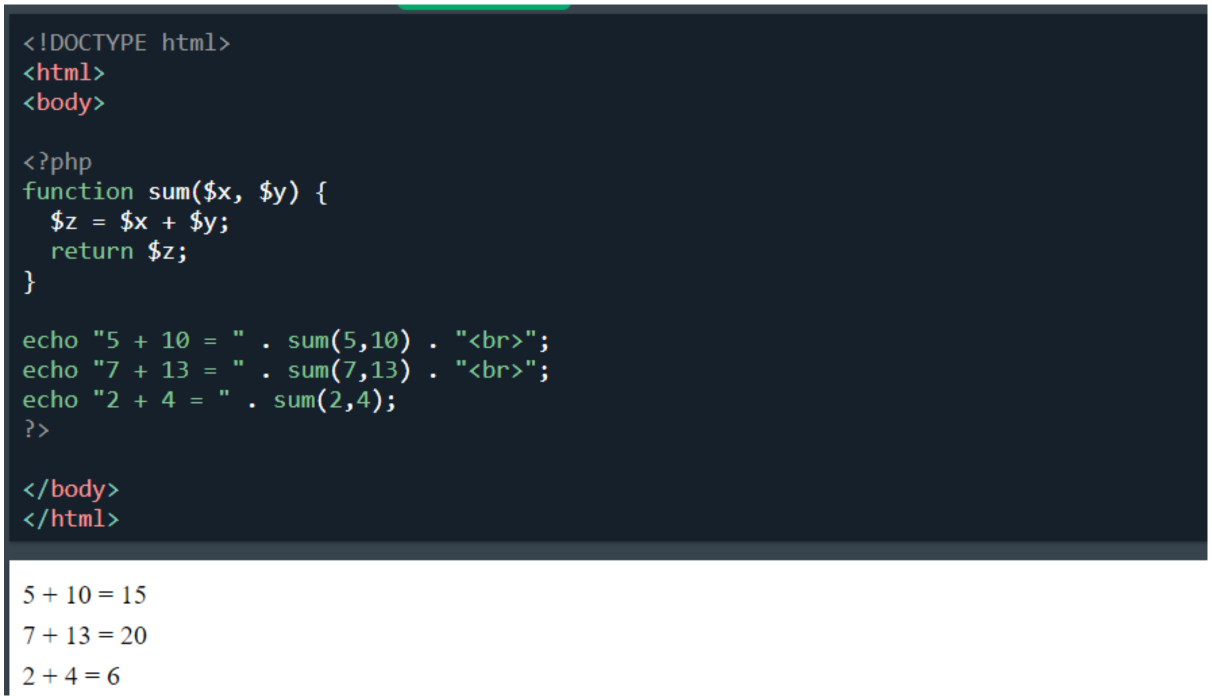
```php
$total = calculateSum(1, 2); // $total is 3
$total = calculateSum(1, 2, 3, 4, 5); // $total is 15
```

This flexibility is beneficial when you need to create functions that can handle various scenarios without explicitly defining the number of parameters.

PHP Functions - Returning values:

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

Example:



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

<?php
function sum($x, $y) {
  $z = $x + $y;
  return $z;
}

echo "5 + 10 = " . sum(5,10) . "<br>";
echo "7 + 13 = " . sum(7,13) . "<br>";
echo "2 + 4 = " . sum(2,4);
?>

</body>
</html>
```

```
5 + 10 = 15
7 + 13 = 20
2 + 4 = 6
```

PHP Built-in Functions

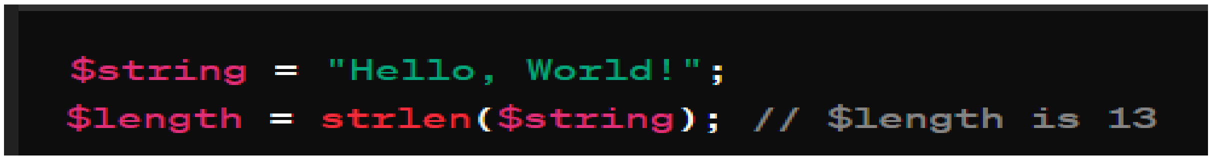
PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

Built-in functions in PHP are pre-defined functions that come with the language. These functions are readily available for use and cover a wide range of functionalities, from string manipulation to mathematical operations. Here's a more in-depth look:

Common PHP Functions:

Explore commonly used functions like `strlen`, `str`_`replace`, `count`, etc.

Example:



```
$string = "Hello, World!";
$length = strlen($string); // $length is 13
```

PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

PHP MySQL Connection

Introduction:

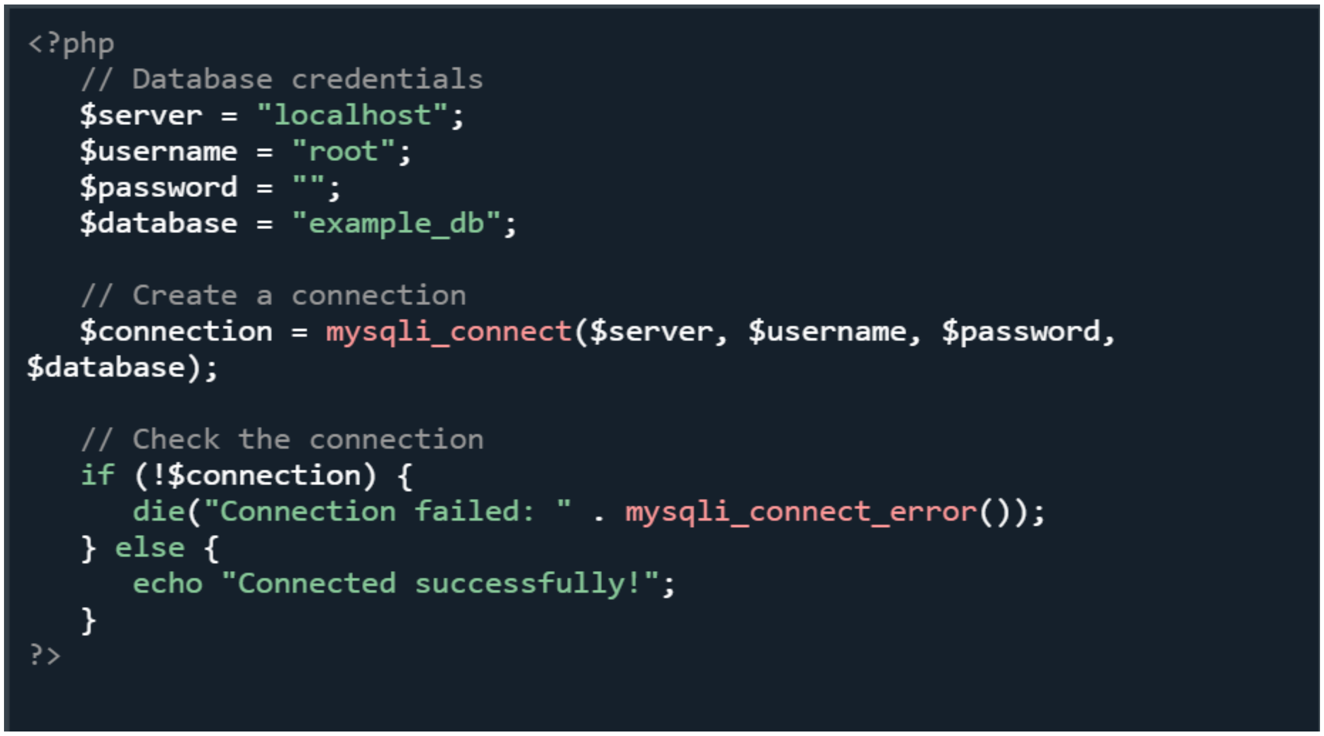
Establishing a connection between PHP and MySQL is fundamental in building dynamic web applications. MySQL is a popular relational database management system, and PHP provides robust functions, particularly through the `mysqli` extension, to interact seamlessly with MySQL databases. This chapter will cover the process of connecting to MySQL using `mysqli_connect` and performing basic MySQL queries and operations.

Connecting to MySQL using mysqli_connect:

Explanation:

`mysqli_connect` is a PHP function that establishes a connection to a MySQL database. It requires specific parameters such as the MySQL server address, username, password, and the target database name.

Example:



```php
<?php
   // Database credentials
   $server = "localhost";
   $username = "root";
   $password = "";
   $database = "example_db";

   // Create a connection
   $connection = mysqli_connect($server, $username, $password,
$database);

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

In this example, we attempt to connect to a MySQL database on the local server with the given credentials. If the connection fails, an error message is displayed; otherwise, a success message is echoed.

Basic MySQL queries and operations

A list of commonly used MySQL queries to create database, use database, create table, insert record, update record, delete record, select record, truncate table and drop table are given below.

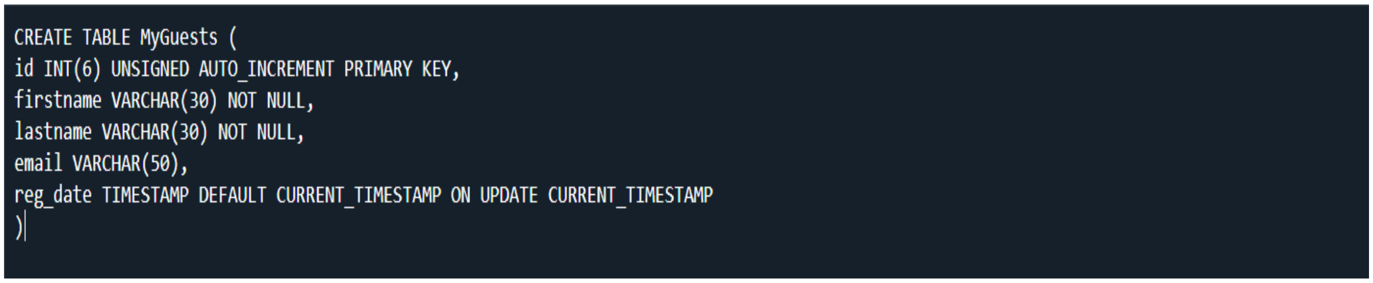
PHP MySQL Create Table

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

Create a MySQL Table Using MySQLi

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

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



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

Notes on the table above:

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

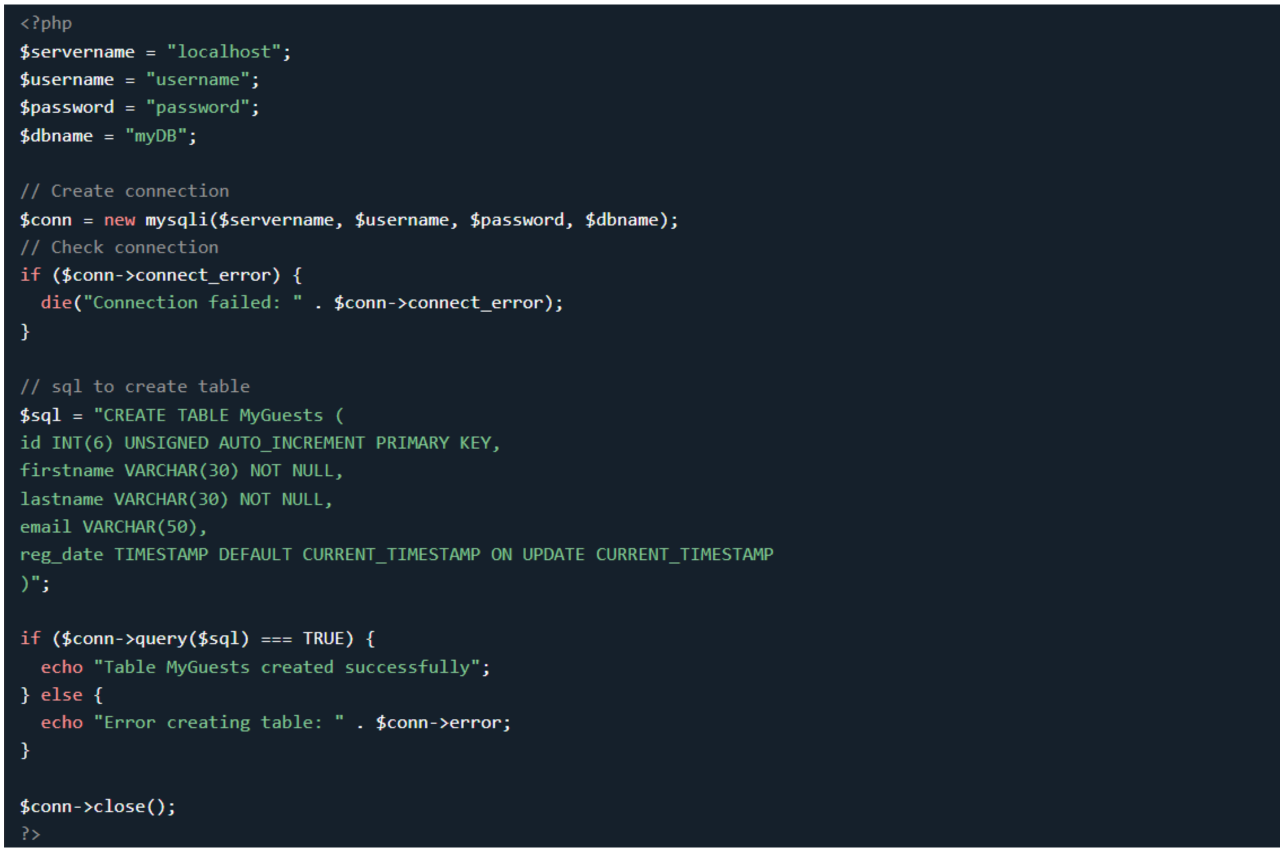
After the data type, you can specify other optional attributes for each column:

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

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

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

Example:



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

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

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

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

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

Insert Data Into MySQL Using MySQLi

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

Here are some syntax rules to follow:

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

The INSERT INTO statement is used to add new records to a MySQL table:

INSERT INTO table_name (column1, column2, column3,...)

VALUES (value1, value2, value3,...)

To learn more about SQL, please visit our SQL tutorial.

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

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

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

Example (MySQLi Object-oriented):

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

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

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

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

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

Select Data From a MySQL Database

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

```
SELECT column_name(s) FROM table_name
```

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

```
SELECT * FROM table_name
```

Select Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

Example (MySQLi Object-oriented):

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

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

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo "0 results";
}
$conn->close();
?>
```

Update Data In a MySQL Table Using MySQLi

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

The following examples update the record with id=2 in the "MyGuests" table:

Example (MySQLi Object-oriented):

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

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

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . $conn->error;
}

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

Delete Data From a MySQL Table Using MySQLi

The DELETE statement is used to delete records from a table:

```sql
DELETE FROM table_name
WHERE some_column = some_value
```

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Example (MySQLi Object-oriented):

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

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

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . $conn->error;
}

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

Security Measures in PHP

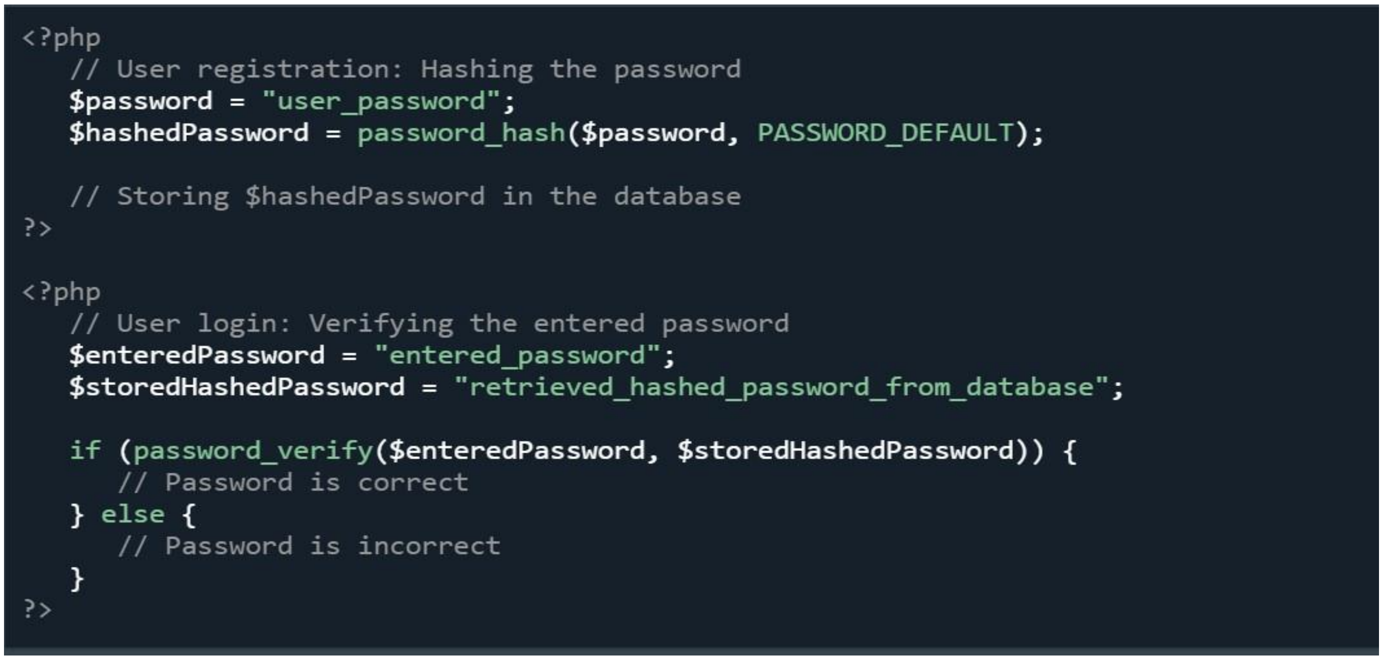
Password Encryption with SHA in Online Forms

Introduction:

Ensuring the confidentiality of user passwords is paramount in web development. Employing strong encryption techniques is vital, and one widely used method is hashing passwords with SHA (Secure Hash Algorithm).

Explanation:

The `password_hash` function in PHP simplifies the process of securely hashing passwords. When a user registers and sets a password, the hashed version is stored in the database. During login attempts, the entered password is hashed and compared to the stored hash**.**



```php
<?php
   // User registration: Hashing the password
   $password = "user_password";
   $hashedPassword = password_hash($password, PASSWORD_DEFAULT);

   // Storing $hashedPassword in the database
?>

<?php
   // User login: Verifying the entered password
   $enteredPassword = "entered_password";
   $storedHashedPassword = "retrieved_hashed_password_from_database";

   if (password_verify($enteredPassword, $storedHashedPassword)) {
      // Password is correct
   } else {
      // Password is incorrect
   }
?>
```

Using Cookies in PHP

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Example:

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example:

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example:

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
  echo "Cookies are enabled.";
} else {
  echo "Cookies are disabled.";
}
?>

</body>
</html>
```

CAPTCHA Text Generation in PHP

Introduction:

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a security mechanism designed to differentiate between human users and automated bots. Text-based CAPTCHAs involve generating and validating distorted text that users must interpret and enter correctly to prove they are human.

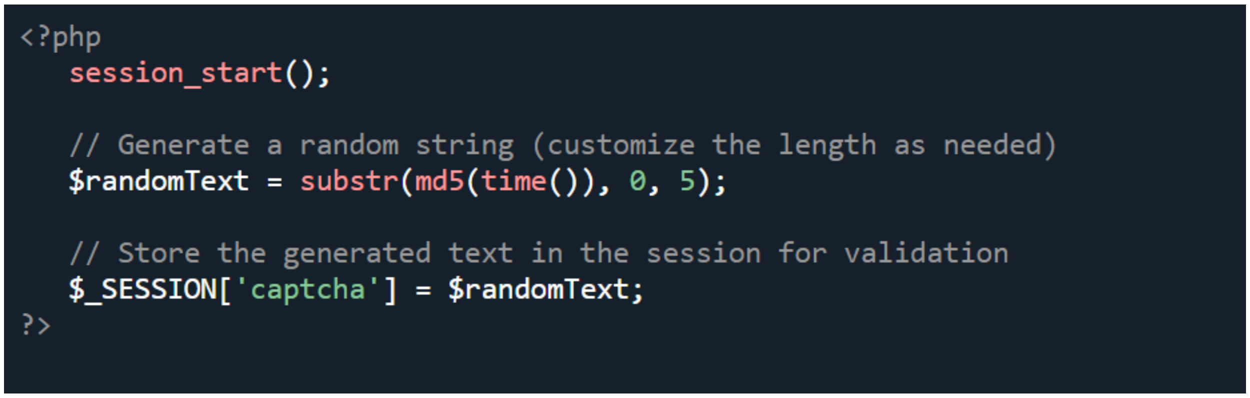
Explanation:

The primary purpose of a text-based CAPTCHA is to prevent automated scripts from submitting forms on websites, as bots often struggle to decipher and interpret distorted text. In PHP, CAPTCHAs can be implemented by generating a random text string, displaying it as an image, and validating the entered text against the stored value.

Implementation:

Let's break down the steps to create a simple text-based CAPTCHA in PHP using the GD library:

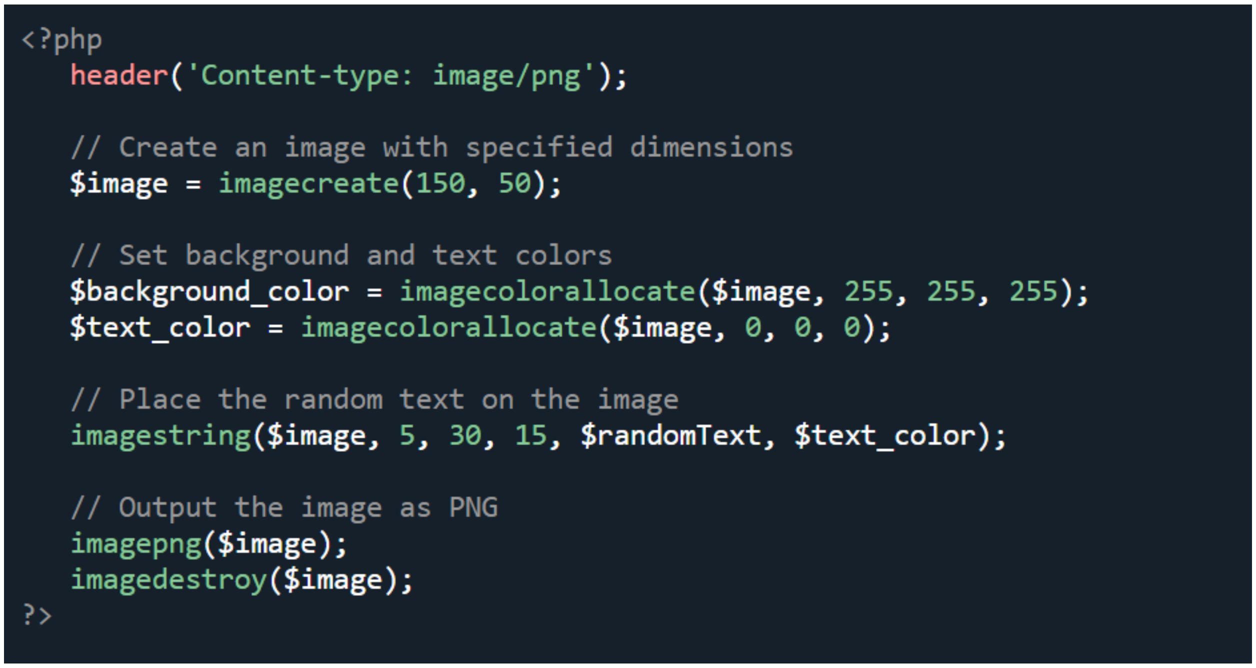
1.Generate a Random Text String:



```php
<?php
   session_start();

   // Generate a random string (customize the length as needed)
   $randomText = substr(md5(time()), 0, 5);

   // Store the generated text in the session for validation
   $_SESSION['captcha'] = $randomText;
?>
```

In this example, we use the `md5` function with the current timestamp to create a random string and store it in the session variable 'captcha.'

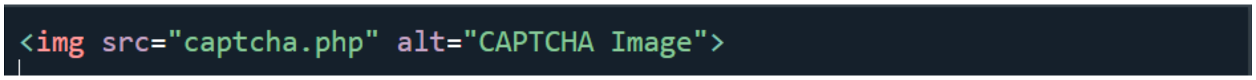
2. Create and Output the CAPTCHA Image:



```php
<?php
   header('Content-type: image/png');

   // Create an image with specified dimensions
   $image = imagecreate(150, 50);

   // Set background and text colors
   $background_color = imagecolorallocate($image, 255, 255, 255);
   $text_color = imagecolorallocate($image, 0, 0, 0);

   // Place the random text on the image
   imagestring($image, 5, 30, 15, $randomText, $text_color);

   // Output the image as PNG
   imagepng($image);
   imagedestroy($image);
?>
```

In this code snippet, we use the GD library to create an image with a specified background color, set text color, and place the random text on the image. The image is then output as PNG.

3.Embed the CAPTCHA in HTML:

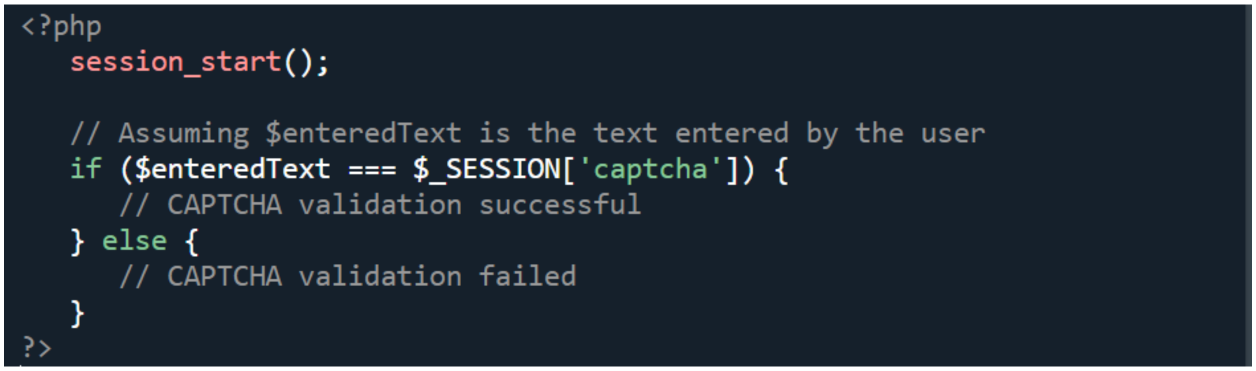


```html
<img src="captcha.php" alt="CAPTCHA Image">
```

Include this HTML code on your form page to display the generated CAPTCHA image.

4. Validate the Entered Text:

Upon form submission, retrieve the entered text and validate it against the stored value in the session.



```php
<?php
   session_start();

   // Assuming $enteredText is the text entered by the user
   if ($enteredText === $_SESSION['captcha']) {
      // CAPTCHA validation successful
   } else {
      // CAPTCHA validation failed
   }
?>
```

Security Considerations:

- Randomness: Ensure the generated text is sufficiently random to challenge automated scripts effectively.
- Length and Complexity: Adjust the length and complexity of the generated text based on the desired security level.
- Session Storage: Store the generated CAPTCHA text securely in the session to validate it during form submission.

Implementing a text-based CAPTCHA adds an additional layer of security to your PHP forms, reducing the risk of automated submissions and enhancing the overall security of your web application.