

Chapter 6 Server Side Scripting using PHP

In the previous chapter, we provided detailed coverage of the basic structure of HTML, CSS, client side scripting using Javascript, AJAX and XML. In this chapter we cover server side scripting language i.e. PHP. We will cover syntax, variables, datatypes, form processing using PHP, connecting to database, creating, selecting, updating, deleting the records from the database.

Basics of PHP

PHP : Hypertext Preprocessor (or simply PHP) is a general purpose programming language originally designed for web development. PHP originally stood for Personal Home Page but now it stands for the recursive acronym PHP i.e. Hypertext Preprocessor

PHP place in the world

PHP is a programming language used mostly for building websites. Instead of a PHP program running on a desktop computer for the use of one person, it typically runs on a web server and is accessed by lots of people using web browsers on their own computer

PHP is a programming language. Something in the web server computer reads your PHP programs, which are instructions written in this programming language and figures out what to do. The PHP engine follows the instruction.

PHP is called sever side scripting language because it runs on a web server. A language such as JavaScript can be used as a client-side language because, embedded in a web browser, it can cause that browser, while running on your PC to do something such as pop up a new window

Advantages of PHP

The following are the advantages of PHP:

PHP is Free

You don't have to pay anyone to use PHP. Whether you run the PHP engine on a beat-up 10-year-old PC in your basement or in a room full of million-dollar "enterprise-class" servers, there are no licensing fees, support fees, maintenance fees, upgrade fees, or any other kind of charge.

PHP is Cross Platform

PHP can be used with a web server computer that runs Windows, Mac OS , Linux and many other version of Unix.

PHP is Widely used

PHP is used on different websites from countless tiny personal homepage to giant's websites

PHP hides its complexity

Powerful ecommerce websites can be build using PHP as well as simple sites. Simpler and complex features can be used simultaneously which doesn't get in the way with each other's

PHP is Built for Web Programming

Unlike other programming languages, PHP was created for generating web pages.

Syntax

Unlike other programming language do have its own syntax PHP also has its own syntax

Start Tag: The less than sign(<) followed by ? symbol and the word 'php' i.e : <?php any thing written inside this tag is considered as PHP code

End Tag: The ? symbol followed by greater than sign(>) i.e ?>

So the syntax for PHP is:

```
<?php
```

```
//statements or blocks
```

```
?>
```

PHP can support multiple start and end tags. Each start tag must be closed by its own end tags.

Comments in PHP

1. Single Line comment

The two backslash (//) or the hash (#) is used to post a single line comment in PHP

2. Multiline comments

For a multiline comment, start the comment with/* and end it with*/. Everything between the /*and */ is treated as a comment by the PHP engine. Multiline comments are useful for temporarily turning off a small block of code.

Basic Rules of PHP Programs

1. Start and End Tags
2. Whitespace and Case-Sensitivity
3. Comments

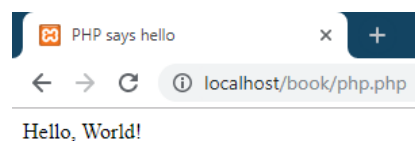
Printing statement in PHP

To print the statement in PHP we use **print** or **echo**

Example 1: Printing Hello world

```
<html>
<head>
<title>PHP says hello</title>
</head>
<body>
<?php print "Hello, World!";
?>
</body>
</html>
```

Output:



Variables

Variables hold the data that your program manipulates while it runs, such as user information that you've loaded from a database or entries that have been typed into an HTML form. In PHP, variables are denoted by a (dollar sign) \$ followed by the variable's name. To assign a value to a variable, use an equals sign (=). This is known as the assignment operator.

Variable names may only include:

- Uppercase or lowercase Basic Latin letters (A-Z and a-z)
- Digits (0-9)
- Underscore (_)
- Any non-Basic Latin character (such as ç), if you're using a character encoding such as UTF-8 for your program file
- Additionally, the first character of a variable name is not allowed to be a digit. Table 2-2 lists some allowable variable names.

Following are the list of some allowed variable names:

\$size
\$drinkSize
\$SUPER_BIG_DRINK
\$_d_r_i_n_k_y
\$drink4you2
\$напиток

The following are the list of some disallowed variable names and reason

Variable name	Reason
\$2hot4u	Begins with a number
\$drink-size	Unacceptable character: -
\$drinkmaster@example.com	Unacceptable characters: @ and .
\$drink!lots	Unacceptable character: !
\$drink+dinner	Unacceptable character: +

Variable names are case-sensitive. This means that variables named \$dinner, \$Dinner, and \$DINNER are separate and distinct, with no more in common than if they were named \$breakfast, \$lunch, and \$supper. In practice, you should avoid using variable names that differ only by letter case. They make programs difficult to read and debug.

Example 2: Defining the variables

```
<?php
$name;
$roll;
?>
```

Text

When they're used in computer programs, pieces of text are called strings. This is because they consist of individual items, strung together. Strings can contain letters, numbers, punctuation, spaces, tabs, or any other characters. A string can even contain the contents of a binary file, such as an image or a sound. The only limit to the length of a string in a PHP program is the amount of memory your computer has.

Defining Text Strings

There are a few ways to indicate a string in a PHP program. The simplest is to surround the string with single quotes or double quotes. The single quotes aren't part of the string. They are delimiters, which tell the PHP engine where the start and end of the string is. If you want to include a single quote inside a string surrounded with single quotes, put a backslash (\) before the single quote inside the string. The backslash tells the PHP engine to treat the following character as a literal single quote instead of the single quote that means "end of string." This is called escaping, and the backslash is called the escape character. An escape character tells the system to do something special with the character that comes after it. Inside a single-quoted string, a single quote usually means "end of string." Preceding the single quote with a backslash changes its meaning to a literal single quote character. To concatenate strings and variables we use "." Known concatenating operator

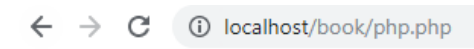
Example 3: Defining Strings

```
<?php
print('Hello, World!')
?>
```

Example 4: Defining strings with backslash

```
<?php
print ('We\'ll each have a bowl of soup.');
```

Output:



← → ↻ ⓘ localhost/book/php.php

We'll each have a bowl of soup.

The biggest difference between single-quoted and double-quoted strings is that when you include variable names inside a double-quoted string, the value of the variable is substituted into the string, which doesn't happen with single-quoted strings. For example, if the variable \$user holds the value Bill, then 'Hi \$user' is just that: Hi \$user. However, "Hi \$user" is Hi Bill. "Variables" gets into this in more detail.

Validating Strings

Validation is the process of checking that input coming from an external source conforms to an expected format or meaning.

The **trim()** function removes whitespace from the beginning and end of a string. **strlen()**, which tells the length of a string, **rtrim()** removes the whitespace from the right side of the string whereas **ltrim()** removes the whitespace from the left side of the string

Syntax:

`trim(variable name or strings)`

`ltrim(variable name or strings)`

`rtrim(variable name or strings)`

`strlen(variable name or strings)`

To compare strings without paying attention to case, use **strcasecmp()**. It compares two strings while ignoring differences in capitalization. If the two strings you provide to **strcasecmp()** are the same independent of any differences between upper- and lowercase letters, it returns 0. The **strcmp()** function is used to compare the strings without ignoring the differences in capitalization.

Syntax:

`strcasecmp(string1, string2)`

`strcmp(string1, string2)`

The **ucwords()** function uppercases the first letter of each word in a string. This is useful when combined with **strtolower()** to produce nicely capitalized names when they are provided to you in all uppercase. The **strtoupper()** converts the string into uppercase. The **substr()** function, you can extract just part of a string. The three arguments to **substr()** are the string to work with, the starting position of the substring to extract, and the number of bytes to extract. The beginning of the string is position 0, not 1. When you give **substr()** a negative number for a start position, it counts back from the end of the string to figure out where to start. A start position of -4 means “start four bytes from the end.” The **str_replace()** function changes parts of a string. It looks for a substring and replaces the substring with a new string

Syntax:

`ucwords(string)`

`strtolower(string)`

`strtoupper(string)`

`substr(string,start,length)`

`str_replace(find,replace,string)`

Data Types

Data Types defines the type of data a variable can store. PHP allows eight different types of data types. All of them are discussed below. The first five are called simple data types and the last three are compound data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- NULL
- Array
- Object
- Resource

Constants

PHP constants are name or identifier that can't be changed during the execution of the script. PHP constants can be defined by 2 ways:

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

PHP constants follow the same PHP variable rules. For example, it can be started with letter or underscore only. Conventionally, PHP constants should be defined in uppercase letters.

PHP constant: define()

It is case sensitive by default.

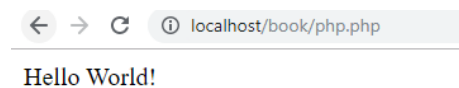
Syntax:

define(name,value)

Example 6: define example

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

Output:



The screenshot shows a web browser window with the address bar displaying 'localhost/book/php.php'. Below the address bar, the text 'Hello World!' is displayed.

PHP constant: const keyword


The const keyword defines constants at compile time. It is a language construct not a function.

1. It is bit faster than define().
2. It is always case sensitive.

Example 7: Const example

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

Output:



A screenshot of a web browser window. The address bar shows 'localhost/book/php.php'. The main content area displays the text 'Hello const by PHP'.

PHP Operator:

PHP Operator is a symbol i.e. used to perform operations on operands. PHP Operators can be categorized in following forms:

- Arithmetic Operators
- Comparison Operators
- Bitwise Operators
- Logical Operators
- String Operators
- Incrementing/Decrementing Operators
- Array Operators
- Type Operators
- Execution Operators
- Error Control Operators
- Assignment Operators

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

- Unary Operators: works on single operands such as ++, -- etc.
- Binary Operators: works on two operands such as binary +, -, *, / etc.
- Ternary Operators: works on three operands such as "?:".

Control Structure

PHP If Else

PHP if else statement is used to test condition. There are various ways to use if statement in PHP.

- if
- if-else
- if-else-if
- nested if

PHP if statement

PHP if statement is executed if condition is true.

Syntax:

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

PHP If-else Statement

PHP if-else statement is executed whether condition is true or false.

Syntax:

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

PHP Switch

PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement

Syntax

```
switch(expression){  
case value1:  
    //code to be executed  
    break;  
case value2:  
    //code to be executed  
    break;  
.....  
default:  
    code to be executed if all cases are not matched;  
}
```

PHP For loop

PHP for loop can be used to traverse set of code for the specified number of times. It should be used if number of iteration is known otherwise use while loop.

Syntax

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

Example 8: For Loop

```
<?php  
for($n=1;$n<=10;$n++){  
    echo "$n<br/>";  
}  
?>
```

Output

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

PHP Nested For Loop

We can use for loop inside for loop in PHP, it is known as nested for loop. In case of inner or nested for loop, nested for loop is executed fully for one outer for loop. If outer for loop is to be executed for 3 times and inner for loop for 3 times, inner for loop will be executed 9 times (3 times for 1st outer loop, 3 times for 2nd outer loop and 3 times for 3rd outer loop).

Example 9: Nested For loop

```
<?php
for($i=1;$i<=3;$i++){
for($j=1;$j<=3;$j++){
echo "$i $j<br/>";
}
}
?>
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

PHP For Each Loop

PHP for each loop is used to traverse array elements. We will be covering more in Array

Syntax:

```
foreach($array_name as $key => $data )
```

PHP While Loop

PHP while loop can be used to traverse set of code like for loop. It should be used if number of iteration is not known.

Syntax:

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

PHP do while loop

PHP do while loop can be used to traverse set of code like php while loop. The PHP do-while loop is guaranteed to run at least once. It executes the code at least one time always because condition is checked after executing the code.

Syntax:

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

PHP Functions

When you're writing computer programs, laziness is a virtue. Reusing code, you've already written makes it easier to do as little work as possible. Functions are the key to code reuse. A function is a named set of statements that you can execute just by invoking the function name instead of retyping the statements. This saves time and prevents errors. Plus, functions make it easier to use code that other people have written.

Advantage of PHP Functions

- **Code Reusability:** PHP functions are defined only once and can be invoked many times, like in other programming languages.
- **Less Code:** It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.
- **Easy to understand:** PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

Creating a Function

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

- Any name ending with an open and closed parenthesis is a function.
- A function name always begins with the keyword function.
- To call a function we just need to write its name followed by the parenthesis
- A function name cannot start with a number. It can start with an alphabet or underscore.
- A function name is not case-sensitive.

Syntax:

```
function                                     function_name(){  
    //                                     executable         code;  
}
```

Calling a Function

Functions can be defined before or after they are called. The PHP engine reads the entire program file and takes care of all the function definitions before it runs any of the commands in the file.

Example 10: Creating a Function

```
<?php  
function sum() {  
    echo 'sum of 10 and 20 is: 30';  
}  
?>
```

Example 11: Defining functions before or after calling them

```
<?php
function greet()
{
    echo 'Hello';
}

greet();
print "Welcome, to the class of PHP ";
message();
function message()
{
    echo "Thanks for coming";
}
?>
```

Passing Arguments to Function

The information or variable, within the function's parenthesis, are called parameters. These are used to hold the values executable during runtime. A user is free to take in as many parameters as he wants, separated with a comma (,) operator. These parameters are used to accept inputs during runtime. While passing the values like during a function call, they are called arguments. An argument is a value passed to a function and a parameter is used to hold those arguments. In common term, both parameter and argument mean the same. We need to keep in mind that for every parameter, we need to pass its corresponding argument.

Syntax:

```
function function_name($first_parameter, $second_parameter) {
    executable code;
}
```

Example 12: Passing arguments to function

```
<?php
// function along with three parameters
function product($num1, $num2, $num3)
{
    $product = $num1 * $num2 * $num3;
    echo "The product is $product";
}

// Calling the function
// Passing three arguments
product (2, 3, 5);
?>
```

Output:

← → ↻ ⓘ localhost/book/php.php
The product is 30

Default values for function parameter

We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument. Let's see a simple example of using default argument value in PHP function.

Example 13: Default values for function parameter

```
<?php
function sayHello($name="John"){
echo "Hello $name<br/>";
}
sayHello("Romeo");
sayHello();//passing no value
sayHello("Johny");
?>
```

Output:

← → ↻ ⓘ localhost/t
Hello Romeo
Hello John
Hello Johny

Returning Values from Functions

Functions can also return values to the part of program from where it is called. The *return* keyword is used to return value back to the part of program, from where it was called. The returning value may be of any type including the arrays and objects. The return statement also marks the end of the function and stops the execution after that and returns the value.

Example 14: Returning values from Function

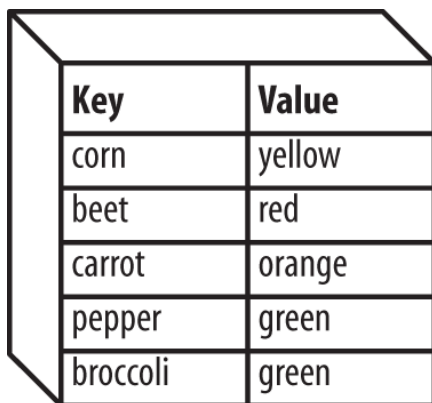
```
<?php
function cube($n){
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
```

Output

← → ↻ ⓘ localhost/
Cube of 3 is: 27

PHP Array

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data. The arrays are helpful to create a list of elements of similar types, which can be accessed using their index or key. An array is made up of elements. Each element has a key and a value. For example, an array holding information about the colors of vegetables has vegetable names for keys and colors for values, as shown



Key	Value
corn	yellow
beet	red
carrot	orange
pepper	green
broccoli	green

An array can only have one element with a given key. In the vegetable color array, there can't be another element with the key corn even if its value is blue. However, the same value can appear many times in one array. You can have green peppers, green broccoli, and green celery.

Any string or number value can be an array element key, such as corn, 4, -36, or Salt Baked Squid. Arrays and other non-scalar values can't be keys, but they can be element values. An element value can be anything: a string, a number, true, false, or a non-scalar type such as another array.

Creating an Array

To create an array, use the array() language construct. Specify a comma-delimited list of key/value pairs, with the key and the value separated by =>

Example 15: Creating an array

```
<?php
$fruit_name = array("Apple","Banana");
?>
```

A shortcut for the array() language construct is a pair of square brackets (called the short array syntax)

Example 16: Using short array syntax

```
<?php
$fruit_name = ["Apple","Banana"];
?>
```


Types of Array

There are basically three types of arrays in PHP:

- **Indexed or Numeric Arrays:** An array with a numeric index where values are stored linearly.
- **Associative Arrays:** An array with a string index where instead of linear storage, each value can be assigned a specific key.
- **Multidimensional Arrays:** An array which contains single or multiple array within it and can be accessed via multiple indices.

Numeric Array

PHP provides some shortcuts for working with arrays that have only numbers as keys. If you create an array with [] or array() by specifying only a list of values instead of key/value pairs, the PHP engine automatically assigns a numeric key to each value. The keys start at 0 and increase by one for each element

Example 17: Numeric Array

```
<?php
$season=array("summer","winter","spring","autumn");
?>
```

To print the array we use print_r() function.

Syntax: print_r(array_name)

Example 18: Printing array

```
<?php
$season=array("summer","winter","spring","autumn");
print_r($season);
?>
```

Output:

```
Array ( [0] => summer [1] => winter [2] => spring [3] => autumn )
```

Associative Array

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

Example 19: Associative Array

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

Output:

```
Array ( [Zack] => Zara [Anthony] => Any [Ram] => Rani [Salim] => Sara [Raghav] => Ravina )
```

Multidimensional Array

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

Syntax:

```
$array_name = array(array());
```

Example 20: Creating multidimensional arrays with array() and []

```
<?php
$meals = array('breakfast' => ['Walnut Bun','Coffee'],
    'lunch'    => ['Cashew Nuts', 'White Mushrooms'],
    'snack'    => ['Dried Mulberries','Salted Sesame Crab']);
print_r($meals);
echo '<br>';

$lunches = [ ['Chicken','Eggplant','Rice'],
    ['Beef','Scallions','Noodles'],
    ['Eggplant','Tofu'] ];
print_r($lunches);
```

```

        echo '<br>';

$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                'Chinese' => array('hot' => 'mustard',
                                   'pepper-salty' => 'prickly ash'));
print_r($flavors);
echo '<br>';
?>

```

Output:

```

Array
(
    [breakfast] => Array
        (
            [0] => Walnut Bun
            [1] => Coffee
        )

    [lunch] => Array
        (
            [0] => Cashew Nuts
            [1] => White Mushrooms
        )

    [snack] => Array
        (
            [0] => Dried Mulberries
            [1] => Salted Sesame Crab
        )

)
Array
(
    [0] => Array
        (
            [0] => Chicken
            [1] => Eggplant
            [2] => Rice
        )

    [1] => Array
        (
            [0] => Beef
            [1] => Scallions
            [2] => Noodles
        )

    [2] => Array
        (
            [0] => Eggplant
            [1] => Tofu
        )

)
Array
(
    [Japanese] => Array
        (
            [hot] => wasabi
            [salty] => soy sauce
        )

    [Chinese] => Array
        (
            [hot] => mustard
            [pepper-salty] => prickly ash
        )

)

```

Looping through arrays

To loop array, we use foreach loop;

Syntax

```
<?php
    foreach ($array_name as $key => $value) {
        # code...
    }
    //or
    foreach ($variable as $value) {
        # code...
    }
?>
```

Example 21: Looping with foreach

```
<?php
$meal = array('breakfast' => 'Walnut Bun',
'lunch' => 'Cashew Nuts and White Mushrooms',
'snack' => 'Dried Mulberries',
'dinner' => 'Eggplant with Chili Sauce');
foreach ($meal as $key => $value) {
    echo "Key is ".$key. " and value is " . $value;
}
?>
```

Output:

```
Key is breakfast and value is Walnut Bun
Key is lunch and value is Cashew Nuts and White Mushrooms
Key is snack and value is Dried Mulberries
Key is dinner and value is Eggplant with Chili Sauce
```

Some Array Function

The **count()** or **sizeof()** function tells you the number of elements in an array

Syntax:

count(array_name) or sizeof(array_name)

To check for an element with certain key, use **array_key_exists()**. This function returns true if an element with the provided key exists in the provided array.

Syntax:

array_key_exists(key,array_name)

To check for an element with certain key, use **in_array()**. This function returns true if it finds an element with the given value. It is case-sensitive when it compares strings.

Syntax:

```
in_array(value,array_name)
```

The **array_search()** function is similar to **in_array()**, but if it finds an element, it returns the element key instead of true

Syntax:

```
array_search(value,array_name)
```

To remove an element from an array, use **unset()**. Removing an element with **unset()** is different than just setting the element value to 0 or the empty string. When you use **unset()**, the element is no longer there when you iterate through the array or count the number of elements in the array. Using **unset()** on an array that represents a store's inventory is like saying that the store no longer carries a product. Setting the element's value to 0 or the empty string says that the item is temporarily out of stock.

Syntax:

```
unset(array[key])
```

To convert string from array, use **implode()** function. It makes a string by combining all the values in an array, putting a string delimiter between each value.

Syntax:

```
implode(separator,array)
```

Example 21: Making a string from an array with **implode()**

```
<?php
$dimsum = array('Chicken Bun','Stuffed Duck Web','Turnip Cake');
$menu = implode(' ', $dimsum);
echo $menu;
?>
```

Output:

Chicken Bun, Stuffed Duck Web, Turnip Cake

The counterpart to **implode()** is called **explode()**. It breaks a string apart into an array. The delimiter argument to **explode()** is the string it should look for to separate array elements.

Syntax:

```
explode(separator,string,limit)
```

Example 22: Turning a string into an array with explode()

```
<?php
    $fish = 'Bass, Carp, Pike, Flounder';
    $fish_list = explode(' ', $fish);
    print_r($fish_list);
?>
```

Output:

Array ([0] => Bass [1] => Carp [2] => Pike [3] => Flounder)

Sorting Arrays

There are several ways to sort arrays. Which function to use depends on how you want to sort your array and what kind of array it is. The following are the types of sorting techniques:

- `sort()`
- `asort()`
- `ksort()`
- `rsort()`
- `arsort()`
- `krsort()`

The **`sort()`** function sorts an array by its element values in ascending order. It should only be used on numeric arrays, because it resets the keys of the array when it sorts. The `rsort()` sorts an array element values in descending order

Example 23: Sorting with `sort()`

```
<?php
$dinner = array('Sweet Corn and Asparagus',
'Lemon Chicken',
'Braised Bamboo Fungus');
echo "before sorting <br>";
print_r($dinner);
echo "<br>";
sort($dinner);
echo "After sorting <br>";
print_r($dinner);
?>
```

Output:

```
before sorting
Array ( [0] => Sweet Corn and Asparagus [1] => Lemon Chicken [2] => Braised Bamboo Fungus )
After sorting
Array ( [0] => Braised Bamboo Fungus [1] => Lemon Chicken [2] => Sweet Corn and Asparagus )
```

To sort an associative array by element value in ascending order, use **`asort()`**. This keeps keys together with their values. The **`arsort()`** sorts array by element value in descending order. The values are sorted in the same way with **`asort()`** as with **`sort()`**, but this time, the keys stick around.

Example 24: sorting with `asort()`

```
<?php
$meal = array('breakfast' => 'Walnut Bun',
'lunch' => 'Cashew Nuts and White Mushrooms',
'snack' => 'Dried Mulberries',
```

```
'dinner' => 'Eggplant with Chili Sauce');
echo "before sorting </br>";
print_r($meal);
echo "</br>";
asort($meal);
echo "After sorting </br>";
print_r($meal);
?>
```

Output:

```
before sorting
Array ( [breakfast] => Walnut Bun [lunch] => Cashew Nuts and White Mushrooms [snack] => Dried Mulberries [dinner] => Eggplant with Chili Sauce )
After sorting
Array ( [lunch] => Cashew Nuts and White Mushrooms [snack] => Dried Mulberries [dinner] => Eggplant with Chili Sauce [breakfast] => Walnut Bun )
```

To sort an array by keys in ascending order, use **ksort()**. This keeps key/value pairs together, but orders them by key. The **krsort()** sorts array by keys in descending order

Example 25: Sorting with ksort()

```
<?php
$meal = array('breakfast' => 'Walnut Bun',
'lunch' => 'Cashew Nuts and White Mushrooms',
'snack' => 'Dried Mulberries',
'dinner' => 'Eggplant with Chili Sauce');
echo "before sorting </br>";
print_r($meal);
echo "</br>";
ksort($meal);
echo "After sorting </br>";
print_r($meal);
?>
```

Output:

```
before sorting
Array ( [breakfast] => Walnut Bun [lunch] => Cashew Nuts and White Mushrooms [snack] => Dried Mulberries [dinner] => Eggplant with Chili Sauce )
After sorting
Array ( [breakfast] => Walnut Bun [dinner] => Eggplant with Chili Sauce [lunch] => Cashew Nuts and White Mushrooms [snack] => Dried Mulberries )
```


Creating Class and Objects in PHP

Like C++ and Java, PHP also supports object oriented programming

1. Classes are the blueprints of objects. One of the big differences between functions and classes is that a class contains both data (variables) and functions that form a package called an: 'object'.
2. Class is a programmer-defined data type, which includes local methods and local variables.
3. Class is a collection of objects. Object has properties and behavior.

Syntax: We define our own class by starting with the keyword 'class' followed by the name you want to give your new class.

```
<?php
class person {

}
?>
```

Objects in PHP

An object is an individual instance of the data structure defined by a class. We define a class once and then make many objects that belong to it. Objects are also known as instances.

Creating _____ an _____ Object:

Following is an example of how to create object using **new** operator.

```
<?php
class Books {
    // Members of class Books
}

// Creating three objects of Books
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
?>
```

PHP Forms

Form processing is an essential component of almost any web application. Forms are how users communicate with your server: signing up for a new account, searching a forum for all the posts about a particular subject, retrieving a lost password, finding a nearby restaurant or shoemaker, or buying a book.

Using a form in a PHP program is a two-step activity. Step one is to display the form. This involves constructing HTML that has tags for the appropriate user-interface elements in it, such as text boxes, checkboxes, and buttons.

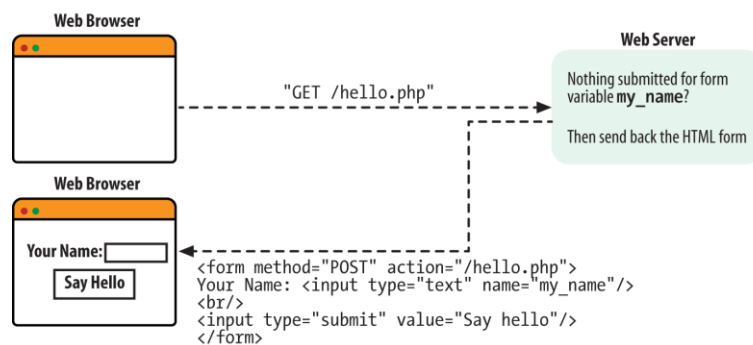
When a user sees a page with a form in it, she inputs the requested information into the form and then clicks a button or hits Enter to send the form information back to your server. Processing that submitted form information is step two of the operation.

Accessing Form Elements

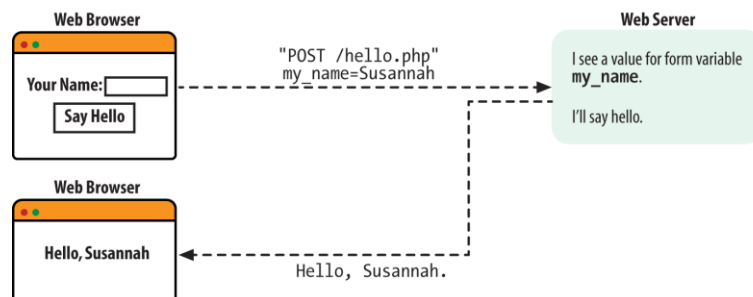
At the beginning of every request, the PHP engine sets up some auto-global arrays that contain the values of any parameters submitted in a form or passed in the URL. URL and form parameters from GET method forms are put into `$_GET`. Form parameters from POST method forms are put into `$_POST`.

The following figure indicates displaying and processing a simple form

Step 1: Retrieve and display the form



Step 2: Submit the form and display the results



The `$_SERVER` auto-global array holds a variety of information about your server and the current request the PHP engine is processing. The `PHP_SELF` element of `$_SERVER` holds the pathname part of the current request's URL, which is shown in the example below.

Example 26: Form processing in PHP with `$_SERVER` and `PHP_SELF` using here document

```
<?php
if ('POST' == $_SERVER['REQUEST_METHOD']) {
    print "Hello, ". $_POST['my_name'];
} else {
    print<<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
    Your name: <input type="text" name="my_name" >
<br>
<input type="submit" value="Say Hello">
</form>
_HTML_;
}
```

Output:

Your name:

When Say Hello button is pressed the page is processed by `$_SERVER` which holds the current request URL's using `PHP_SELF` and redirected to the page and the `$_SERVER['REQUEST_METHOD']` checks the method in the form tag and the form parameters with name having value `my_name` is put into `$_POST['my_name']` for further processing.

Useful Server Variables

In addition to **PHP_SELF** and **REQUEST_METHOD**, the **\$_SERVER** auto-global array contains a number of useful elements that provide information on the web server and the current request. The following are some of the entries that **\$_SERVER** holds

Element	Description
QUERY_STRING	The part of the URL after the question mark where the URL parameters live. The example query string shown is for the URL <i>http://www.example.com/catalog/store.php?category=kitchen&price=5</i> .
PATH_INFO	Extra path information tacked onto the end of the URL after a slash. This is a way to pass information to a script without using the query string. The example PATH_INFO shown is for the URL <i>http://www.example.com/catalog/store.php/browse</i> .
SERVER_NAME	The name of the website on which the PHP engine is running. If the web server hosts many different virtual domains, this is the name of the particular virtual domain that is being accessed.
DOCUMENT_ROOT	The directory on the web server computer that holds the documents available on the website. If the document root is <i>/usr/local/htdocs</i> for the website <i>http://www.example.com</i> , then a request for <i>http://www.example.com/catalog/store.php</i> corresponds to the file <i>/usr/local/htdocs/catalog/store.php</i> .
REMOTE_ADDR	The IP address of the user making the request to your web server.
REMOTE_HOST	If your web server is configured to translate user IP addresses into hostnames, this is the hostname of the user making the request to your web server. Because this address-to-name translation is relatively expensive (in terms of computational time), most web servers do not do it.
HTTP_REFERER	If someone clicked on a link to reach the current URL, HTTP_REFERER contains the URL of the page that contained the link. This value can be faked, so don't use it as your sole criterion for giving access to private web pages. It can, however, be useful for finding out who's linking to you.
HTTP_USER_AGENT	The web browser that retrieved the page. The example value is the signature of Firefox 37 running on OS X. Like with HTTP_REFERER, this value can be faked, but is useful for analysis.

Example 27: Accessing Form parameters using \$_POST

```
<html>
<head>
  <title>Form</title>
</head>
<body>
<form method="POST" action="process.php">

Name:<input type="text" name="name">
Email: <input type="email" name="email">
<input type="submit" name="submit">
</form>
</body>
</html>
```

Output:

Name: Email:

After the form is submitted it will redirect to the process.php as the action="process.php" in the form

The process.php page looks like :

```
<?php
echo "The name and emails are:";
echo "Name: " . $_POST['name'];
echo "Email: " . $_POST['email'];
?>
```

And the output is displayed as:

The name and emails are: Name: John Email: John@gmail.com

Form Validation

Data validation is one of the most important parts of a web application. Weird, wrong, and damaging data shows up where you least expect it. Users can be careless, malicious, and fabulously more creative (often accidentally) than you may ever imagine when you are designing your application. Validation means check the input submitted by the user. There are two types of validation are available in PHP. They are as follows:

- Client-Side Validation – Validation is performed on the client machine web browsers.
- Server Side Validation – After submitted by data, the data has sent to a server and perform validation checks in server machine.

Server Side Validation

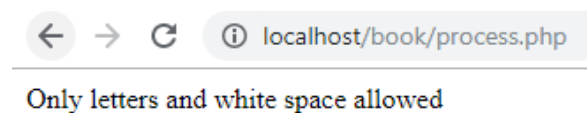
Text validation:

For validating text we use regular expression i.e **preg_match()**. This function searches string for pattern, returns true if pattern exists, otherwise returns false. Usually search starts from beginning of subject string. The optional parameter offset is used to specify the position from where to start the search.

Example 28: Text Validation with preg_match()

```
<?php
$name = $_POST['name'];
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
    echo "Only letters and white space allowed";
}
else{
    echo "The name is". $name;
}
?>
```

Output:



In the above example the preg_match() function searches in string given through a variable \$name for the pattern of a-zA-Z.

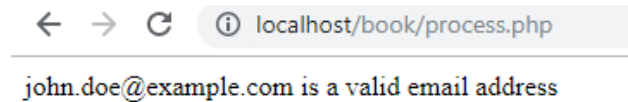
Email validation:

To validate the email, use **FILTER_VALIDATE_EMAIL**

Example 29: Email validation using FILTER_VALIDATE_EMAIL

```
<?php
// Variable to check
$email = $_POST['email'];
// Validate email
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
?>
```

Output:



The screenshot shows a web browser window with the address bar displaying 'localhost/book/process.php'. Below the address bar, the output of the script is shown: 'john.doe@example.com is a valid email address'.

Number Validation:

To validate the integer, use **FILTER_VALIDATE_INT** and to validate float, use **FILTER_VALIDATE_FLOAT**

Some useful validating functions

- **htmlspecialchars**: converts some predefined characters to HTML entities.
- **stripslashes**: function removes backslashes
- **trim**: removes the white spaces

Cookies and Session

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.

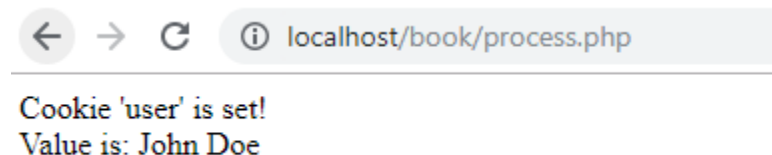
PHP Create/Retrieve a Cookie

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 30: Creating and retrieving a cookie

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); //86400 = 1 day
if(!isset($_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];
}
?>
<html>
<body>
```

Output:



Cookie 'user' is set!
Value is: John Doe

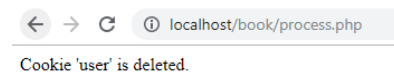
Deleting a cookie

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

Example 31: Deleting a cookie

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
echo "Cookie 'user' is deleted.";
?>
```

Output:



Cookie 'user' is deleted.

Session

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state. Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser. So; Session variables hold information about one single user, and are available to all pages in one application.

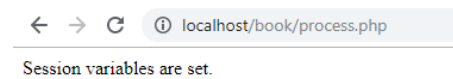
Starting PHP Session

A PHP session is easily started by making a call to the `session_start()` function. This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to `session_start()` at the beginning of the page. Session variables are stored in associative array called `$_SESSION[]`. These variables can be accessed during lifetime of a session. The following example starts a session then register a variable called counter that is incremented each time the page is visited during the session. Make use of `isset()` function to check if session variable is already set or not.

Example 32: Starting session and setting session values

```
<?php
// Start the session
session_start();
// Set session variables
$_SESSION["username"] = "john";
$_SESSION["email"] = "ducky@gmail.com";
echo "Session variables are set.";
?>
```

Output:



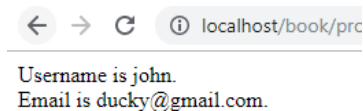
Getting PHP Session Variable values

Next, we create another example. From this example, we will access the session information we set on the previous example. Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Example 33: Getting session variables

```
<?php
session_start();
// Echo session variables that were set on previous page
echo "Username is " . $_SESSION["username"] . "<br>";
echo "Email is " . $_SESSION["email"] . ".";
?>
```

Output:



Destroy a PHP Session

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

Example 34: Destroying a session

```
<?php
session_start();
?>
<?php
// remove all session variables
session_unset();
// destroy the session
session_destroy();
?>
```

Working with Databases

With PHP, you can connect to and manipulate databases. MySQL is the most popular database system used with PHP. The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows. Databases are useful for storing information categorically. There are three ways of working with MySQL and PHP

1. MySQLi (object-oriented)
2. MySQLi (procedural)
3. PDO

Above the three ways we will be doing the procedural approach in this section

Example 35: Creating Connection using object-oriented Approach

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
// Creating connection
$conn = new mysqli($servername, $username, $password);
// Checking connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

In the above example, the variable name \$servername is the name of the server, the variable \$username is the username of the server and the variable \$password is the password used to connect the server. The **new mysqli()** function takes the parameters and creates the connection, the **connect_error()** returns the error if there is any error in the process of being connected.

Example 36: Creating Connection using procedural approach

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
// Creating connection
$conn = mysqli_connect($servername, $username, $password);
// Checking connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

In MySQLi procedural approach instead of creating an instance we can use the **mysqli_connect()** function available in PHP to establish a connection. This function takes the information as arguments such as host, username, password, database name etc. This function returns MySQL link identifier on successful connection or FALSE when failed to establish a connection.

The some of the mysqli function used in procedural approach are:

- **mysqli_connect**: opens a new connection to the MySQL server.
- **mysqli_connect_error**: generates the error while creating the connection
- **mysqli_connect_errno**: generates the error number while creating the connection
- **mysqli_query**: performs a query against the database
- **mysqli_error**: returns the last error description for the most recent function call
- **mysqli_errno**: returns the last error code for the most recent function call
- **mysqli_num_rows**: returns the number of rows in a result set.
- **mysqli_close**: closes the connection

Example 37: Creating Database using procedural approach

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// Create database
$sql = "CREATE DATABASE webTech";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Output:

Database created successfully

Example 38: Creating Table in the database

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

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// sql to create table
$sql = "CREATE TABLE Student(
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
    echo "Table Students created successfully";
} else {
    echo "Error creating table: " . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Output:

Table Students created successfully

Example 39: Inserting data into database table

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "webTech";
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "INSERT INTO Students (firstname, lastname, email)
VALUES ('John', 'Ducky', 'john@ducky.com')";

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

Output:

New record created successfully

Example 40: Inserting data into database table using form

```
<?php
if(!empty($_POST))
{
$servername='localhost';
$username='root';
$password='';
$dbname='web';
$conn=mysqli_connect($servername,$username,$password,$dbname);
if(!$conn)
{
die ("Connection error".mysqli_connect_error());
}
}
```

```

$name=$_POST['name'];
$dob=$_POST['dob'];
$age=$_POST['age'];
$sex=$_POST['sex'];
$sql="insert into qno9_data (name,age,dob,sex) values ('$name','$age','$dob','$sex')";
$result=mysqli_query($connection,$sql);
if($result)
{
echo "Data inserted Sucessfully";
}
else
{
echo "Data not inserted Xx-ERROR-xx ".mysqli_error($connection);
}
mysqli_close($connection);
}
?>
<html>
<head>
</head>
<body>
<h3> Fill in the form </h3>
<form action="<?php $_SERVER['PHP_SELF'];?>" method="post">
<label>Name:</label>
<input type="text" name="name" value='<?php echo $_POST["name"]?>'>
<label>Age: </label>
<input type="text" name="age" value='<?php echo $_POST["age"]?>'>
<label>DOB:</label>
<input type="text" name="dob" value='<?php echo $_POST["dob"]?>'>
<label>Gender:</label>
<input type="radio" name="sex" value="Male"> Male <input type="radio"
name="sex" value="Male"> Female
<input type="submit" value="Send"><input type="reset" value="Clear"></u>
</form>
</body></html>

```

Fill in the form

Name: Age: DOB: Gender: ☐ Male ☐ Female

Example 41: Displaying the data in HTML table format

```
<html>
<head>
  <title>Class data in table format</title>
</head>
<body>
  <table border="1">
    <thead>
      <tr>
        <th>ID</th>
        <th>NAME</th>
        <th>AGE</th>
        <th>GENDER</th>
        <th>COUNTRY</th>
        <th>EMAIL</th>
        <th>PHONE</th>
        <th>EDIT</th>
        <th>DELETE</th>
      </tr>
    </thead>
    <?php
    $servername="localhost";
    $username="root";
    $password="";
    $dbname="citizen";
    $conn=mysqli_connect($servername,$username,$password,$dbname);
    if(!$conn){
      die("Connection failed".mysqli_connect_error()."<br>");
    }

    $sql="SELECT ID,NAME,AGE,GENDER,COUNTRY,EMAIL,PHONE FROM personal_details";
    $result=mysqli_query($conn,$sql);
    if(mysqli_num_rows($result)>0){
      while($row=mysqli_fetch_assoc($result)){
        echo '<tr>';
        echo '<td>'.$row['ID'].'</td>';
        echo '<td>'.$row['NAME'].'</td>';
        echo '<td>'.$row['AGE'].'</td>';
        echo '<td>'.$row['GENDER'].'</td>';
        echo '<td>'.$row['COUNTRY'].'</td>';
        echo '<td>'.$row['EMAIL'].'</td>';
        echo '<td>'.$row['PHONE'].'</td>';
        echo '<td><a href=\"edit.php?id=\".$row['ID'].\"\">Edit</a></td>';
        echo '<td><a href=\"delete.php?id=\".$row['ID'].\"\">Delete</a></td>';
        echo '</tr>';
      }
    }
  </tbody>
</table>
</body>
</html>
```

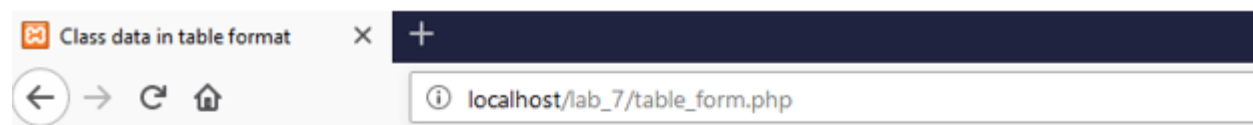
```

    }
}
mysqli_close($conn);
?>
<tbody>
</tbody>
</table>
</body>
</html>

```

The **mysqli_fetch_assoc()** function fetches a result row as an associative array.

Output:



ID	NAME	AGE	GENDER	COUNTRY	EMAIL	PHONE	EDIT	DELETE
1	sagu	21		Nepal	sajs@gmail.com	4	Edit	Delete
2	milan	21	Female	Nepal	milan@gmail.com	986128900	Edit	Delete
3	babita	21	Female	India	babita@gmail.com	98019909	Edit	Delete
4	sanaya	24	Female	India	sanayakc@gmail.com	2147483647	Edit	Delete
5	Aryan	23	Male	India	aerf2gmail.com	56789900	Edit	Delete
6	Kiran	22	Female	london	kiran@gmail.com	2147483647	Edit	Delete

Example 42: Updating the records in a table

While updating the records it requires two pages edit.php which loads the form with values another one update.php where the edit code are written

```

//edit.php
<?php
$servername="localhost";
$username="root";
$password="";
$dbname="citizen";
$conn=mysqli_connect($servername,$username,$password,$dbname);
if(!$conn){
die("Connection failed".mysqli_connect_error()."<br>");
}
$id=$_GET['id'];
$sql="SELECT * from personal_details where id=$id";

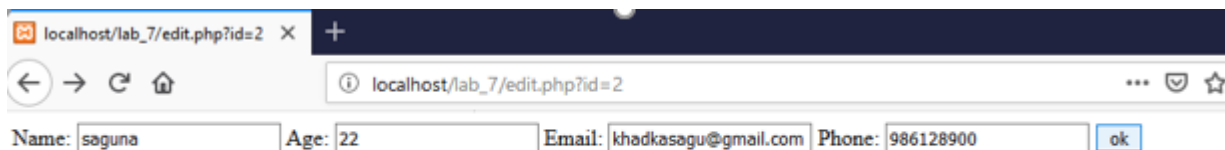
```

```

$result=mysqli_query($conn,$sql);
if(mysqli_num_rows($result)>0){
while ($row=mysqli_fetch_assoc($result)){
?>
<form action="update.php?id=<?php echo $id?>"method="post">
Name:
<input type="text" name="name" value="<?php echo $row['NAME']?>">
Age:
<input type="numeric" name="age" value="<?php echo $row['AGE']?>">
Email:
<input type="text" name="email" value="<?php echo $row['EMAIL']?>">
Phone:
<input type="text" name="phone" value="<?php echo $row['PHONE']?>">
<input type="submit" value="ok">
</form>
<?php
}}
mysqli_close($conn);
?>

```

Output:



```

//update.php
<?php
$servername="localhost";
$username="root";
$password="";
$dbname="citizen";
$conn=mysqli_connect($servername,$username,$password,$dbname);
if(!$conn){
    die("Connection failed".mysqli_connect_error()."<br>");
}
$id=$_GET['id'];
$name=$_POST['name'];
$age=$_POST['age'];
$email=$_POST['email'];
$phone=$_POST['phone'];

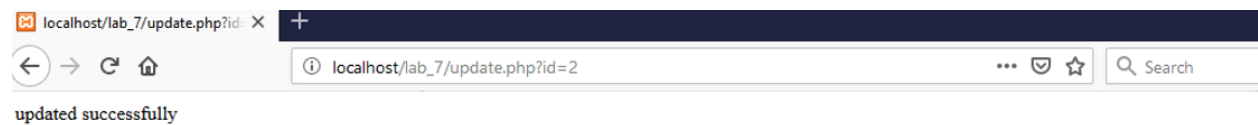
```

```

$sql="update personal_details set name='$name',age='$age',phone='$phone',email='$email' where id=$id";
$result =mysqli_query($conn,$sql);
if($result){
    echo "updated successfully";
}
else{
    echo "error".mysqli_error($conn);
}
mysqli_close($conn);
?>

```

Output:



Inserting Multiple Data

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

```

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

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

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

if (mysqli_multi_query($conn, $sql)) {
    echo "Records inserted successfully";
}

```

```
} else {  
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);  
}  
mysqli_close($conn);  
?>
```

Output

Records inserted successfully.

Date and Time functions in PHP

PHP date function is an in-built function that simplify working with date data types. The PHP date function is used to format a date or time into a human readable format. It can be used to display the date of article was published. record the last updated a data in a database.

PHP Date Syntax & Example

PHP Date the following basic syntax

```
<?php  
date(format,[timestamp]);  
?>
```

Explanation:

- “date(…)” is the function that returns the current time on the server.
- “format” is the general format which we want our output to be i.e.;
- “Y-m-d” for PHP date format YYYY-MM-DD
- “Y” to display the current year
- “[timestamp]” is optional. If no timestamp has been provided, PHP will get the use the php current date time on the server.

Introduction to PHP Framework

1. Codeigniter Framework

CodeIgniter is an Application Development Framework - a toolkit - for people who build web sites using PHP. Its goal is to enable you to develop projects much faster than you could if you were writing code from scratch, by providing a rich set of libraries for commonly needed tasks, as well as a simple interface and logical structure to access these libraries. CodeIgniter lets you creatively focus on your project by minimizing the amount of code needed for a given task.

Who is CodeIgniter For?

CodeIgniter is right for you if:

- You want a framework with a small footprint.
- You need exceptional performance.
- You need broad compatibility with standard hosting accounts that run a variety of PHP versions and configurations.
- You want a framework that requires nearly zero configuration.
- You want a framework that does not require you to use the command line.
- You want a framework that does not require you to adhere to restrictive coding rules.
- You are not interested in large-scale monolithic libraries like PEAR.
- You do not want to be forced to learn a templating language (although a template parser is optionally available if you desire one).
- You eschew complexity, favoring simple solutions.
- You need clear, thorough documentation.

To use codeigniter just go to <https://codeigniter.com/download.php#311> and download the codeigniter 3.1.11 which is the current version

Installation Instructions

CodeIgniter is installed in four steps:

- Unzip the package.
- Upload the CodeIgniter folders and files to your server. Normally the *index.php* file will be at your root.
- Open the *application/config/config.php* file with a text editor and set your base URL. If you intend to use encryption or sessions, set your encryption key.
- If you intend to use a database, open the *application/config/database.php* file with a text editor and set your database settings.

2. Laravel Framework

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable, creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.

Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, we've attempted to combine the very best of what we have seen in other web frameworks, including frameworks implemented in other languages, such as Ruby on Rails, ASP.NET MVC, and Sinatra.

Laravel is accessible, yet powerful, providing powerful tools needed for large, robust applications. A superb inversion of control container, expressive migration system, and tightly integrated unit testing support give you the tools you need to build any application with which you are tasked.

Installing Laravel

Laravel utilizes [Composer](#) to manage its dependencies. So, before using Laravel, make sure you have Composer installed on your machine.

Via Laravel Installer

First, download the Laravel installer using Composer:

```
composer global require laravel/installer
```

Make sure to place Composer's system-wide vendor bin directory in your `$PATH` so the laravel executable can be located by your system. This directory exists in different locations based on your operating system; however, some common locations include:

- macOS and GNU / Linux Distributions: `$HOME/.config/composer/vendor/bin`
- Windows: `%USERPROFILE%\AppData\Roaming\Composer\vendor\bin`

Once installed, the `laravel new` command will create a fresh Laravel installation in the directory you specify. For instance, `laravel new blog` will create a directory named `blog` containing a fresh Laravel installation with all of Laravel's dependencies already installed:

```
laravel new blog
```

Via Composer Create-Project

Alternatively, you may also install Laravel by issuing the Composer `create-project` command in your terminal:

```
composer create-project --prefer-dist laravel/laravel blog
```

Local Development Server

If you have PHP installed locally and you would like to use PHP's built-in development server to serve your application, you may use the `serve` Artisan command. This command will start a development server at `http://localhost:8000`:

```
php artisan serve
```

Introduction to Wordpress

WordPress is a free and open source content management system (CMS) based on PHP and MySQL. It is the most widely used CMS software in the world, and as of May 2019, it powers more than 30% of the top 10 million websites and has an estimated 60% market share of all websites built using a CMS.

WordPress started as a simple blogging system in 2003, but it has evolved into a full CMS with thousands of plugins, widgets, and themes. It is licensed under the General Public License (GPLv2 or later).

