Mahatma Gandhi Vidyamandir's

Panchavati College of Management & Computer Science

E-Content

Department of BBA/BBA(CA)

Class: TYBBA-CA **Subject: Advanced Web Technology(601)**

Created by: Chaitali Shinde

Unit 1

Introduction to Object Oriented Programming in PHP

What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Classes and Objects

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

OOP Case

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.

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

Define a Class

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods go inside the braces:

Syntax

```php
<?php
class Fruit {
  // code goes here...
}
?>
```

Below we declare a class named Fruit consisting of two properties ($name and $color) and two methods set_name() and get_name() for setting and getting the $name property:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Define Objects

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class is created using the new keyword.

In the example below, $apple and $banana are instances of the class Fruit:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

In the example below, we add two more methods to class Fruit, for setting and getting the $color property:

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

PHP - The $this Keyword

The $this keyword refers to the current object, and is only available inside methods.

Look at the following example:

Example

```
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
?>
```

So, where can we change the value of the $name property? There are two ways:

1. Inside the class (by adding a set_name() method and use $this):

Example

```
<?php
class Fruit {
  public $name;
  function set_name($name) {
    $this->name = $name;
  }
}
$apple = new Fruit();
$apple->set_name("Apple");
?>
```

2. Outside the class (by directly changing the property value):

Example

```
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
$apple->name = "Apple";
?>
```

PHP - instanceof

You can use the instanceof keyword to check if an object belongs to a specific class:

Example

```
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

Inheritance

Inheritance in OOP = When a class derives from another class.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

An inherited class is defined by using the extends keyword.

Let's look at an example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}
```

```
// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

Example Explained

The Strawberry class is inherited from the Fruit class.

This means that the Strawberry class can use the public $name and $color properties as well as the public __construct() and intro() methods from the Fruit class because of inheritance.

The Strawberry class also has its own method: message().

PHP - Inheritance and the Protected Access Modifier

In the previous chapter we learned that protected properties or methods can be accessed within the class and by classes derived from that class. What does that mean?

Let's look at an example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}

// Try to call all three methods from outside class
$strawberry = new Strawberry("Strawberry", "red");  // OK. __construct() is public
$strawberry->message(); // OK. message() is public
$strawberry->intro(); // ERROR. intro() is protected
?>
```

In the example above we see that if we try to call a protected method (intro()) from outside the class, we will receive an error. public methods will work fine!

Let's look at another example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
    // Call protected method from within derived class - OK
    $this -> intro();
  }
}

$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() is public
$strawberry->message(); // OK. message() is public and it calls intro() (which is protected)
from within the derived class
?>
```

In the example above we see that all works fine! It is because we call the protected method (intro()) from inside the derived class.

PHP - Overriding Inherited Methods

Inherited methods can be overridden by redefining the methods (use the same name) in the child class.

Look at the example below. The __construct() and intro() methods in the child class (Strawberry) will override the __construct() and intro() methods in the parent class (Fruit):

Example

```
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public $weight;
  public function __construct($name, $color, $weight) {
    $this->name = $name;
    $this->color = $color;
    $this->weight = $weight;
  }
  public function intro() {
    echo "The fruit is {$this->name}, the color is {$this->color}, and the weight is {$this->weight} gram.";
  }
}

$strawberry = new Strawberry("Strawberry", "red", 50);
$strawberry->intro();
?>
```

PHP - The final Keyword

The final keyword can be used to prevent class inheritance or to prevent method overriding.

The following example shows how to prevent class inheritance:

Example

```php
<?php
final class Fruit {
  // some code
}

// will result in error
class Strawberry extends Fruit {
  // some code
}
?>
```

The following example shows how to prevent method overriding:

Example

```php
<?php
class Fruit {
  final public function intro() {
    // some code
  }
}

class Strawberry extends Fruit {
  // will result in error
  public function intro() {
    // some code
  }
}
?>
```

Interfaces

Interfaces allow you to specify what methods a class should implement.

Interfaces make it easy to use a variety of different classes in the same way. When one or more classes use the same interface, it is referred to as "polymorphism".

Interfaces are declared with the interface keyword:

Syntax

```php
<?php
interface InterfaceName {
  public function someMethod1();
  public function someMethod2($name, $color);
  public function someMethod3() : string;
}
?>
```

PHP - Interfaces vs. Abstract Classes

Interface are similar to abstract classes. The difference between interfaces and abstract classes are:

- Interfaces cannot have properties, while abstract classes can
- All interface methods must be public, while abstract class methods is public or protected
- All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary
- Classes can implement an interface while inheriting from another class at the same time

PHP - Using Interfaces

To implement an interface, a class must use the implements keyword.

A class that implements an interface must implement **all** of the interface's methods.

Example

```
<?php
interface Animal {
  public function makeSound();
}

class Cat implements Animal {
  public function makeSound() {
    echo "Meow";
  }
}

$animal = new Cat();
$animal->makeSound();
?>
```

From the example above, let's say that we would like to write software which manages a group of animals. There are actions that all of the animals can do, but each animal does it in its own way.

Using interfaces, we can write some code which can work for all of the animals even if each animal behaves differently:

Example

```php
<?php
// Interface definition
interface Animal {
  public function makeSound();
}

// Class definitions
class Cat implements Animal {
  public function makeSound() {
    echo " Meow ";
  }
}

class Dog implements Animal {
  public function makeSound() {
    echo " Bark ";
  }
}

class Mouse implements Animal {
  public function makeSound() {
    echo " Squeak ";
  }
}

// Create a list of animals
$cat = new Cat();
$dog = new Dog();
$mouse = new Mouse();
$animals = array($cat, $dog, $mouse);

// Tell the animals to make a sound
foreach($animals as $animal) {
  $animal->makeSound();
}
?>
```

Example Explained

Cat, Dog and Mouse are all classes that implement the Animal interface, which means that all of them are able to make a sound using the makeSound() method. Because of this, we can loop through all of the animals and tell them to make a sound even if we don't know what type of animal each one is.

Since the interface does not tell the classes how to implement the method, each animal can make a sound in its own way.

Abstract Classes

What are Abstract Classes and Methods?

Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

An abstract class or method is defined with the abstract keyword:

Syntax

```
<?php
abstract class ParentClass {
  abstract public function someMethod1();
  abstract public function someMethod2($name, $color);
  abstract public function someMethod3() : string;
}
?>
```

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method
- The child class method must be defined with the same or a less restricted access modifier
- The number of required arguments must be the same. However, the child class may have optional arguments in addition

Let's look at an example:

Example

```
<?php
// Parent class
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
  public function intro() : string {
    return "Choose German quality! I'm an $this->name!";
  }
}

class Volvo extends Car {
  public function intro() : string {
    return "Proud to be Swedish! I'm a $this->name!";
  }
}

class Citroen extends Car {
  public function intro() : string {
    return "French extravagance! I'm a $this->name!";
  }
}

// Create objects from the child classes
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();
?>
```

Example Explained

The Audi, Volvo, and Citroen classes are inherited from the Car class. This means that the Audi, Volvo, and Citroen classes can use the public $name property as well as the public __construct() method from the Car class because of inheritance.

But, intro() is an abstract method that should be defined in all the child classes and they should return a string.

PHP - More Abstract Class Examples

Let's look at another example where the abstract method has an argument:

Example

```
<?php
abstract class ParentClass {
  // Abstract method with an argument
  abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
  public function prefixName($name) {
    if ($name == "John Doe") {
      $prefix = "Mr.";
    } elseif ($name == "Jane Doe") {
      $prefix = "Mrs.";
    } else {
      $prefix = "";
    }
    return "{$prefix} {$name}";
  }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

Let's look at another example where the abstract method has an argument, and the child class has two optional arguments that are not defined in the parent's abstract method:

Example

```
<?php
abstract class ParentClass {
  // Abstract method with an argument
  abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
  // The child class may define optional arguments that are not in the parent's abstract method
  public function prefixName($name, $separator = ".", $greet = "Dear") {
    if ($name == "John Doe") {
      $prefix = "Mr";
    } elseif ($name == "Jane Doe") {
      $prefix = "Mrs";
    } else {
      $prefix = "";
    }
    return "{$greet} {$prefix}{$separator} {$name}";
  }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

Constructor

PHP - The __construct Function

A constructor allows you to initialize an object's properties upon creation of the object.

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

Notice that the construct function starts with two underscores (__)!

We see in the example below, that using a constructor saves us from calling the set_name() method which reduces the amount of code:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
```

```
$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

Another example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function get_name() {
    return $this->name;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

PHP - The __construct Function

A constructor allows you to initialize an object's properties upon creation of the object.

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

Notice that the construct function starts with two underscores (__)!

We see in the example below, that using a constructor saves us from calling the set_name() method which reduces the amount of code:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

Another example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function get_name() {
    return $this->name;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

Destructor

PHP - The __destruct Function

A destructor is called when the object is destructed or the script is stopped or exited.

If you create a __destruct() function, PHP will automatically call this function at the end of the script.

Notice that the destruct function starts with two underscores (__)!

The example below has a __construct() function that is automatically called when you create an object from a class, and a __destruct() function that is automatically called at the end of the script:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}

$apple = new Fruit("Apple");
?>
```

Another example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function __destruct() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}
$apple = new Fruit("Apple", "red");
?>
```

Introspection in PHP:

Introspection in PHP offers the useful ability to examine an object's characteristics, such as its name, parent class (if any) properties, classes, interfaces, and methods.
PHP offers a large number of functions that you can use to accomplish the task.
The following are the functions to extract basic information about classes such as their name, the name of their parent class and so on.
In-built functions in PHP Introspection :

Function	**Description**
class_exists()	Checks whether a class has been defined.
et_class()	Returns the class name of an object.
get parent_class()	Returns the class name of a Return object's parent class.
is_subclass_of()	Checks whether an object has a given parent class.
get_declared_classes()	Returns a list of all declared classes.
get_class_methods()	Returns the names of the class methods.
get_class_vars()	Returns the default properties of a class
interface_exists()	Checks whether the interface is defined.
method_exists()	Checks whether an object defines a method.

Example 1:

```
<?php
if(class_exists('cwipedia'))
{
 $ob=new cwipedia();
 echo "This is cwipedia.in";
}
else
{
 echo "Not exist";
}
?>
```

Output: Not exist

Example 2:

```
<?php
class cwipedia
{
 //decl
}
if(class_exists('cwipedia'))
{
 $ob=new cwipedia();
 echo "This is cwipedia.in";
}
else
{
 echo "Not exist";
}

?>
```

Output: This is cwipedia.in

Serialization in PHP:

Serialization is a technique used by programmers to preserve their working data in a format that can later be restored to its previous form.

Serializing an object means converting it to a byte stream representation that can be stored in a file.

Serialization in PHP is mostly automatic, it requires little extra work from you, beyond calling the serialize() and unserialize() functions.

Serialize():

The serialize() converts a storable representation of a value.
The serialize() function accepts a single parameter which is the data we want to serialize and returns a serialized string
A serialize data means a sequence of bits so that it can be stored in a file, a memory buffer, or transmitted across a network connection link. It is useful for storing or passing PHP values around without losing their type and structure.

Syntax:
serialize(value);

unserialize():

unserialize() can use string to recreate the original variable values i.e. converts actual data from serialized data.

Syntax:
unserialize(string);

Example:

```
<?php
$a=array('Shivam','Rahul','Vilas');
$s=serialize($a);
print_r($s);
$s1=unserialize($s);
echo "<br>";
print_r($s1);
?>
```

Output:

```
a:3:{i:0;s:6:"Shivam";i:1;s:5:"Rahul";i:2;s:5:"Vilas";}
Array ( [0] => Shivam [1] => Rahul [2] => Vilas )
```

Unit 2-Web Techniques

Global Variables – Superglobals/Web Variables

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

The PHP superglobal variables are:

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

PHP $GLOBALS

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

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

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

Example

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

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

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

PHP $_SERVER

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

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

Example

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

PHP $_REQUEST

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

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

Example

```
<html>
<body>

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

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

</body>
</html>
```

PHP $_POST

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

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

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
```

```
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

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

</body>
</html>
```

PHP $_GET

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

$_GET can also collect data sent in the URL.

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

```
<html>
<body>

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

</body>
</html>
```

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

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

Example

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

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

</body>
</html>
```

PHP Form Processing

we will discuss how to process form in PHP. HTML forms are used to send the user information to the server and returns the result back to the browser. For example, if you want to get the details of visitors to your website, and send them good thoughts, you can collect the user information by means of form processing. Then, the information can be validated either at the client-side or on the server-side. The final result is sent to the client through the respective web browser. To create a HTML form, **form** tag should be used.

PHP Form Validation

Think SECURITY when processing PHP forms!

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:

The validation rules for the form above are as follows:

Field	**Validation Rules**
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)

Gender	Required. Must select one

First we will look at the plain HTML code for the form:

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
```

The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"
>
```

When the form is submitted, the form data is sent with method="post".

What is the $_SERVER["PHP_SELF"] variable?

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the htmlspecialchars() function?

The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The $_SERVER["PHP_SELF"] variable can be used by hackers!

If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

So far, so good.

However, consider that a user enters the following URL in the address bar:

http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"
>
```

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/scr
ipt&gt;">
```

The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.

When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

```
<script>location.href('http://www.hacked.com')</script>
```

- this would not be executed, because it would be saved as HTML escaped code, like this:

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test_input().

Now, we can check each $_POST variable with the test_input() function, and the script looks like this:

Example

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

Notice that at the start of the script, we check whether the form has been submitted using $_SERVER["REQUEST_METHOD"]. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters, dashes, apostrophes and whitespaces. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.

In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

PHP - Validate Name, E-mail, and URL

Now, the script looks like this:

Example

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression also allows dashes in the
URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP Cookies

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";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<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>
```

Note: The `setcookie()` function must appear BEFORE the <html> tag.

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use `setrawcookie()` instead).

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the `setcookie()` function:

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>
```

PHP Sessions

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

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

What is a PHP 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.

Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

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

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

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Note: The `session_start()` function must be the very first thing in your document. Before any HTML tags.

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

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()`).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

Another way to show all the session variable values for a user session is to run the following code:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>
```

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Destroy a PHP Session

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

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>
</body>
</html>
```

Unit 3-Databases

PHP MySQL Database

MySQL is the most popular database system used with PHP.

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

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

- Employees
- Products
- Customers
- Orders

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

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

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

Should I Use MySQLi or PDO?

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

Both MySQLi and PDO have their advantages:

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

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

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

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

MySQL Examples in Both MySQLi and PDO

Syntax

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

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

MySQLi Installation

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

For installation details, go to: http://php.net/manual/en/mysqli.installation.php

PDO Installation

For installation details, go to: http://php.net/manual/en/pdo.installation.php

Open a Connection to MySQL

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

Example (MySQLi Object-Oriented)

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

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

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

```
echo "Connected successfully";
?>
```

Note on the object-oriented example above:

$connect_error was broken until PHP 5.2.9 and 5.3.0. If you need to ensure compatibility with PHP versions prior to 5.2.9 and 5.3.0, use the following code instead:

```
// Check connection
if (mysqli_connect_error()) {
  die("Database connection failed: " . mysqli_connect_error());
}
```

Example (MySQLi Procedural)

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

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

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

Example (PDO)

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

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

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

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

Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

MySQLi Object-Oriented:

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

MySQLi Procedural:

```
mysqli_close($conn);
```

PDO:

```
$conn = null;
```

PHP Create a MySQL Database

A database consists of one or more tables.

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

Create a MySQL Database Using MySQLi and PDO

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

The following examples create a database named "myDB":

Example (MySQLi Object-oriented)

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

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

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

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

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

Tip: If you have to use a specific port, add an empty string for the database-name argument, like this: new mysqli("localhost", "username", "password", "", port)

Example (MySQLi Procedural)

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

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

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

```
mysqli_close($conn);
?>
```

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

Example (PDO)

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

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

$conn = null;
?>
```

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

PHP MySQL Create Table

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

Create a MySQL Table Using MySQLi and PDO

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

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

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

Notes on the table above:

The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go to our Data Types reference.

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

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

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

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

Example (MySQLi Object-oriented)

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

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

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

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

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

Example (MySQLi Procedural)

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

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

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

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

mysqli_close($conn);
?>
```

Example (PDO)

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

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

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

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

$conn = null;
?>
```

PHP MySQL Insert Data

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();
?>
```

Get ID of The Last Inserted Record

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

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

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

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

Example (MySQLi Object-oriented)

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

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

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

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

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

Insert Multiple Records

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

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

Example (MySQLi Object-oriented)

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

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

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

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

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

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

MySQL Select Data

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
```

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

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
$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();
?>
```

Code lines to explain from the example above:

First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the `function num_rows()` checks if there are more than zero rows returned.

If there are more than zero rows returned, the function `fetch_assoc()` puts all the results into an associative array that we can loop through. The `while()` loop loops through the result set and outputs the data from the id, firstname and lastname columns.

Select and Filter Data From a MySQL Database

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

SELECT column_name(s) FROM table_name WHERE column_name operator value

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

Select and Filter Data With MySQLi

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

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 = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
$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();
?>
```

Run example »

Code lines to explain from the example above:

First, we set up the SQL query that selects the id, firstname and lastname columns from the MyGuests table where the lastname is "Doe". The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the `function num_rows()` checks if there are more than zero rows returned.

If there are more than zero rows returned, the function `fetch_assoc()` puts all the results into an associative array that we can loop through. The `while()` loop loops through the result set and outputs the data from the id, firstname and lastname columns.

Select and Order Data From a MySQL Database

The ORDER BY clause is used to sort the result-set in ascending or descending order.

The ORDER BY clause sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC

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

Select and Order Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table. The records will be ordered by the lastname column:

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 = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$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();
?>
```

Code lines to explain from the example above:

First, we set up the SQL query that selects the id, firstname and lastname columns from the MyGuests table. The records will be ordered by the lastname column. The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the `function num_rows()` checks if there are more than zero rows returned.

If there are more than zero rows returned, the function `fetch_assoc()` puts all the results into an associative array that we can loop through. The `while()` loop loops through the result set and outputs the data from the id, firstname and lastname columns.

Delete Data From a MySQL Table Using MySQLi

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

```
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!

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

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

The following examples delete the record with id=3 in 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 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();
?>
```

After the record is deleted, the table will look like this:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15

2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

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!

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

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in 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 = "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();
?>
```

After the record is updated, the table will look like this:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Doe	mary@example.com	2014-10-23 10:22:30

Limit Data Selections From a MySQL Database

MySQL provides a LIMIT clause that is used to specify the number of records to return.

The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.

Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

When the SQL query above is run, it will return the first 30 records.

What if we want to select records 16 - 25 (inclusive)?

Mysql also provides a way to handle this: by using OFFSET.

The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

You could also use a shorter syntax to achieve the same result:

```
$sql = "SELECT * FROM Orders LIMIT 15, 10";
```

Notice that the numbers are reversed when you use a comma.

PHP MySQLi Functions

Function	Description
affected_rows()	Returns the number of affected rows in the previous MySQL operation
autocommit()	Turns on or off auto-committing database modifications
begin_transaction()	Starts a transaction
change_user()	Changes the user of the specified database connection
character_set_name()	Returns the default character set for the database connection
close()	Closes a previously opened database connection
commit()	Commits the current transaction
connect()	Opens a new connection to the MySQL server

connect_errno()	Returns the error code from the last connection error
connect_error()	Returns the error description from the last connection error
data_seek()	Adjusts the result pointer to an arbitrary row in the result-set
debug()	Performs debugging operations
dump_debug_info()	Dumps debugging info into the log
errno()	Returns the last error code for the most recent function call
error()	Returns the last error description for the most recent function call
error_list()	Returns a list of errors for the most recent function call
fetch_all()	Fetches all result rows as an associative array, a numeric array, or both
fetch_array()	Fetches a result row as an associative, a numeric array, or both
fetch_assoc()	Fetches a result row as an associative array
fetch_field()	Returns the next field in the result-set, as an object

fetch_field_direct()	Returns meta-data for a single field in the result-set, as an object
fetch_fields()	Returns an array of objects that represent the fields in a result-set
fetch_lengths()	Returns the lengths of the columns of the current row in the result-set
fetch_object()	Returns the current row of a result-set, as an object
fetch_row()	Fetches one row from a result-set and returns it as an enumerated array
field_count()	Returns the number of columns for the most recent query
field_seek()	Sets the field cursor to the given field offset
get_charset()	Returns a character set object
get_client_info()	Returns the MySQL client library version
get_client_stats()	Returns statistics about client per-process
get_client_version()	Returns the MySQL client library version as an integer
get_connection_stats()	Returns statistics about the client connection

get_host_info()	Returns the MySQL server hostname and the connection type
get_proto_info()	Returns the MySQL protocol version
get_server_info()	Returns the MySQL server version
get_server_version()	Returns the MySQL server version as an integer
info()	Returns information about the last executed query
init()	Initializes MySQLi and returns a resource for use with real_connect()
insert_id()	Returns the auto-generated id from the last query
kill()	Asks the server to kill a MySQL thread
more_results()	Checks if there are more results from a multi query
multi_query()	Performs one or more queries on the database
next_result()	Prepares the next result-set from multi_query()
options()	Sets extra connect options and affect behavior for a connection

ping()	Pings a server connection, or tries to reconnect if the connection has gone down
poll()	Polls connections
prepare()	Prepares an SQL statement for execution
query()	Performs a query against a database
real_connect()	Opens a new connection to the MySQL server
real_escape_string()	Escapes special characters in a string for use in an SQL statement
real_query()	Executes a single SQL query
reap_async_query()	Returns result from an async SQL query
refresh()	Refreshes/flushes tables or caches, or resets the replication server information
rollback()	Rolls back the current transaction for the database
select_db()	Select the default database for database queries
set_charset()	Sets the default client character set

set_local_infile_default()	Unsets user defined handler for load local infile command
set_local_infile_handler()	Set callback function for LOAD DATA LOCAL INFILE command
sqlstate()	Returns the SQLSTATE error code for the error
ssl_set()	Used to establish secure connections using SSL
stat()	Returns the current system status
stmt_init()	Initializes a statement and returns an object for use with stmt_prepare()
store_result()	Transfers a result-set from the last query
thread_id()	Returns the thread ID for the current connection
thread_safe()	Returns whether the client library is compiled as thread-safe
use_result()	Initiates the retrieval of a result-set from the last query executed

Sample Application Program for Login System

we will create four files here for the login system.

1. **index.html -** This file is created for the GUI view of the login page and empty field validation.

2. **style.css -** This file is created for the attractive view of the login form.
3. **connection.php -** Connection file contains the connection code for database connectivity.
4. **authentication.php -** This file validates the form data with the database which is submitted by the user.

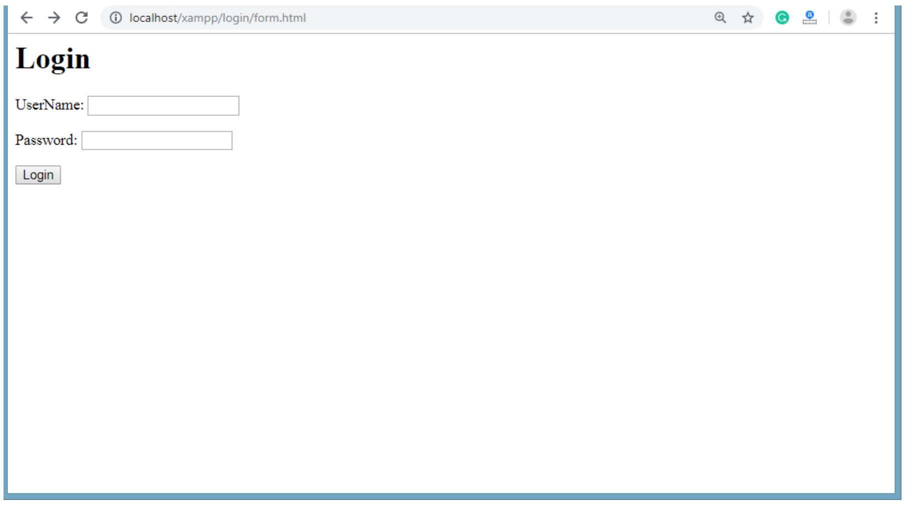
index.html

First, we need to design the login form for the website user to interact with it. This login form is created using html and also contains the empty field validation, which is written in JavaScript. The code for the index.html file is given below:

```
<html>
<head>
   <title>PHP login system</title>
   // insert style.css file inside index.html
   <link rel = "stylesheet" type = "text/css" href = "style.css">
</head>
<body>
   <div id = "frm">
      <h1>Login</h1>
      <form name="f1" action = "authentication.php" onsubmit = "return validation()" method =
"POST">
         <p>
            <label> UserName: </label>
            <input type = "text" id ="user" name  = "user" />
         </p>
         <p>
            <label> Password: </label>
            <input type = "password" id ="pass" name  = "pass" />
         </p>
         <p>
            <input type =  "submit" id = "btn" value = "Login" />
         </p>
      </form>
   </div>
   // validation for empty field
   <script>
         function validation()
         {
            var id=document.f1.user.value;
            var ps=document.f1.pass.value;
            if(id.length=="" && ps.length=="") {
               alert("User Name and Password fields are empty");
```

```
                    return false;
                }
                else
                {
                    if(id.length=="") {
                        alert("User Name is empty");
                        return false;
                    }
                    if (ps.length=="") {
                    alert("Password field is empty");
                    return false;
                    }
                }
            }
        </script>
</body>
</html>
```

After executing the above code on the browser, the login page will appear as below if it does not contain style.css file.


localhost/xampp/login/form.html

Login

UserName:

Password:

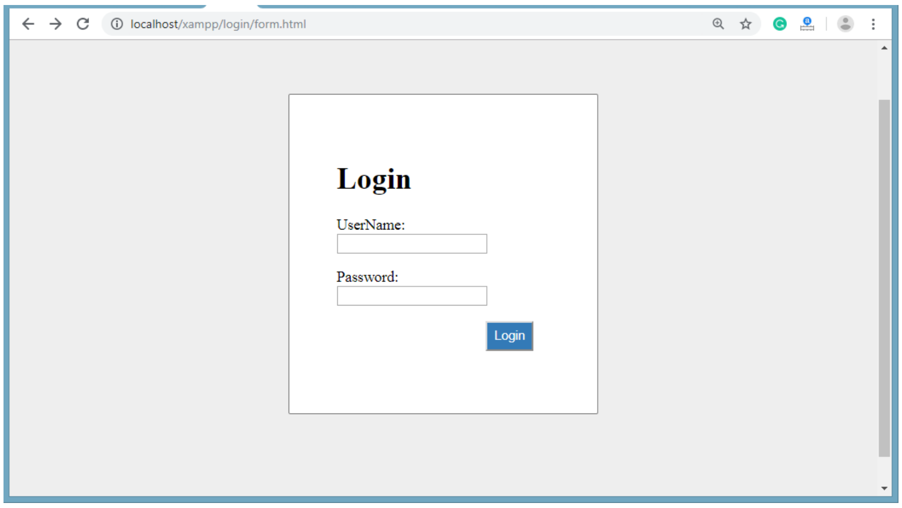
Login

style.css

Now, we will create style.css file to provide a more attractive view to the login form. The CSS code for the style.css file is given below:

```
body{
    background: #eee;
}
#frm{
    border: solid gray 1px;
    width:25%;
    border-radius: 2px;
    margin: 120px auto;
    background: white;
    padding: 50px;
}
#btn{
    color: #fff;
    background: #337ab7;
    padding: 7px;
    margin-left: 70%;
}
```

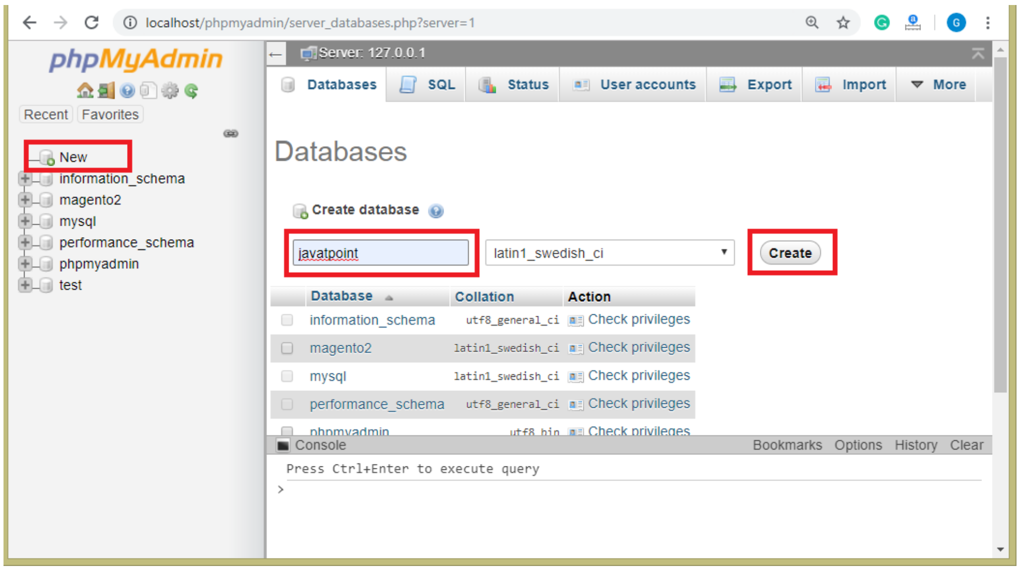
After including above CSS file in index.html, the login form will be like –



Database and Table Creation

Now, the next step is to create the database and the login table inside the database.

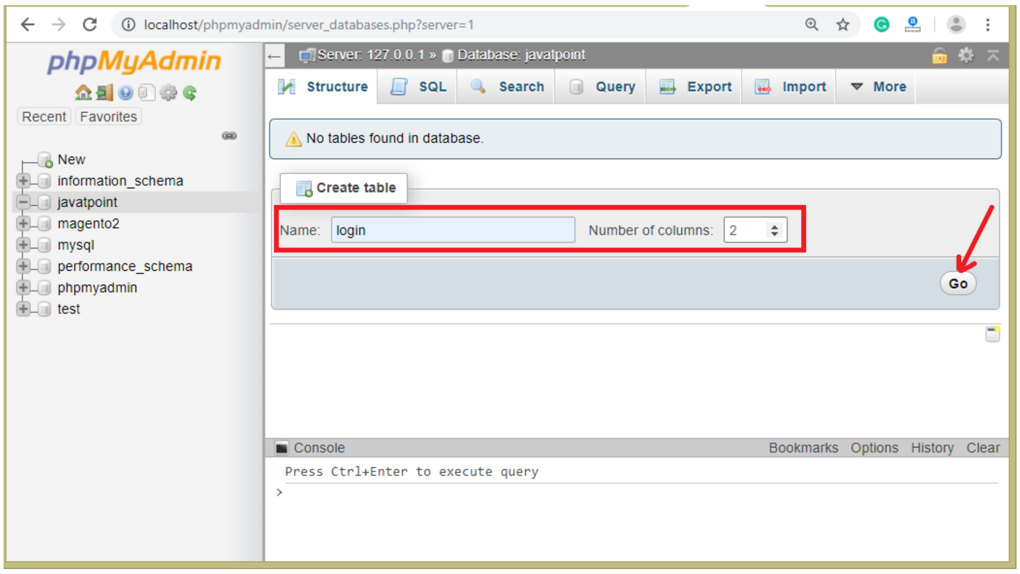
- Access the phpMyAdmin on the browser using **localhost/phpmyadmin/** and create a table in the database. Here we will create a database and table using GUI based phpMyAdmin rather than queries execution.
- Click on **New** and enter the database name and then click on **Create** button.



Database	Collation	Action
information_schema	utf8_general_ci	Check privileges
magento2	latin1_swedish_ci	Check privileges
mysql	latin1_swedish_ci	Check privileges
performance_schema	utf8_general_ci	Check privileges
phpmyadmin	utf8_bin	Check privileges

```
Press Ctrl+Enter to execute query
>
```

- Now we will create a login table in the database. Create a table by name **login** in the database which you have created earlier.

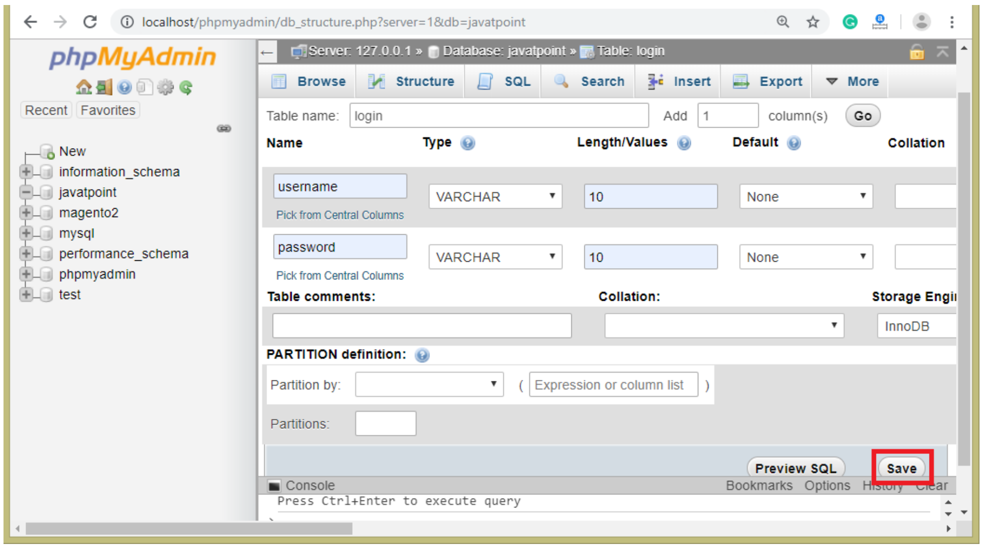


No tables found in database.

Name: login   Number of columns: 2

```
Press Ctrl+Enter to execute query
>
```

- Specify the column **Name** and their **Type** and **Length** in the table in which we will store the **username** and **password** for the different users and save it by clicking on the **save** button.


localhost/phpmyadmin/db_structure.php?server=1&db=javatpoint

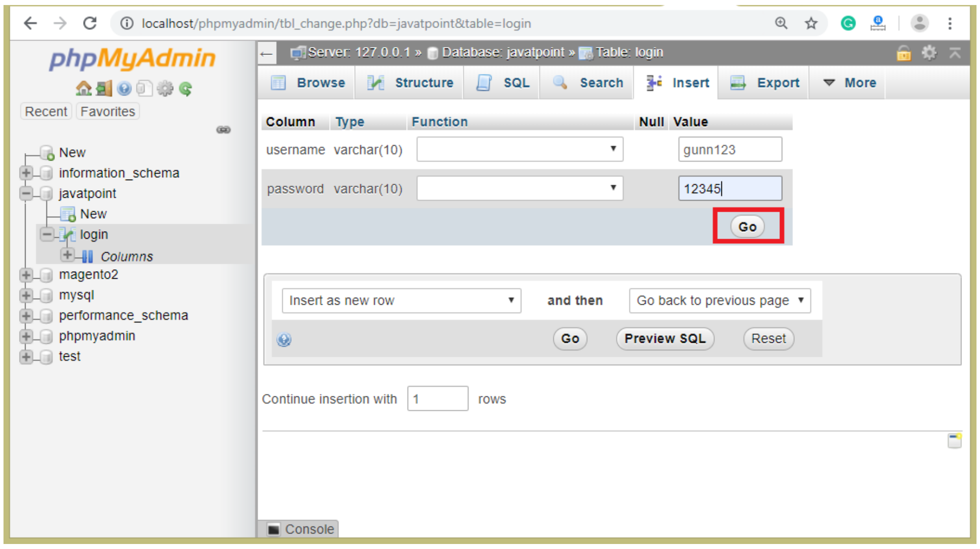
Server: 127.0.0.1 » Database: javatpoint » Table: login

Table name: login Add 1 column(s)

Name	Type	Length/Values	Default
username	VARCHAR	10	None
password	VARCHAR	10	None

Storage Engine: InnoDB

- Click on the insert, from where we can **insert** the records in columns. So insert the **username** and **password** here and click on **Go** button to save the record.


localhost/phpmyadmin/tbl_change.php?db=javatpoint&table=login

Server: 127.0.0.1 » Database: javatpoint » Table: login

Column	Type	Function	Null	Value
username	varchar(10)			gunn123
password	varchar(10)			12345

Insert as new row and then Go back to previous page

Continue insertion with 1 rows

connection.php

Next step is to do the connectivity of login form with the database which we have created in the previous steps. We will create connection.php file for which code is given below:

```
<?php
   $host = "localhost";
   $user = "root";
   $password = '';
   $db_name = "javatpoint";

   $con = mysqli_connect($host, $user, $password, $db_name);
   if(mysqli_connect_errno()) {
      die("Failed to connect with MySQL: ". mysqli_connect_error());
   }
```

```
?>
```

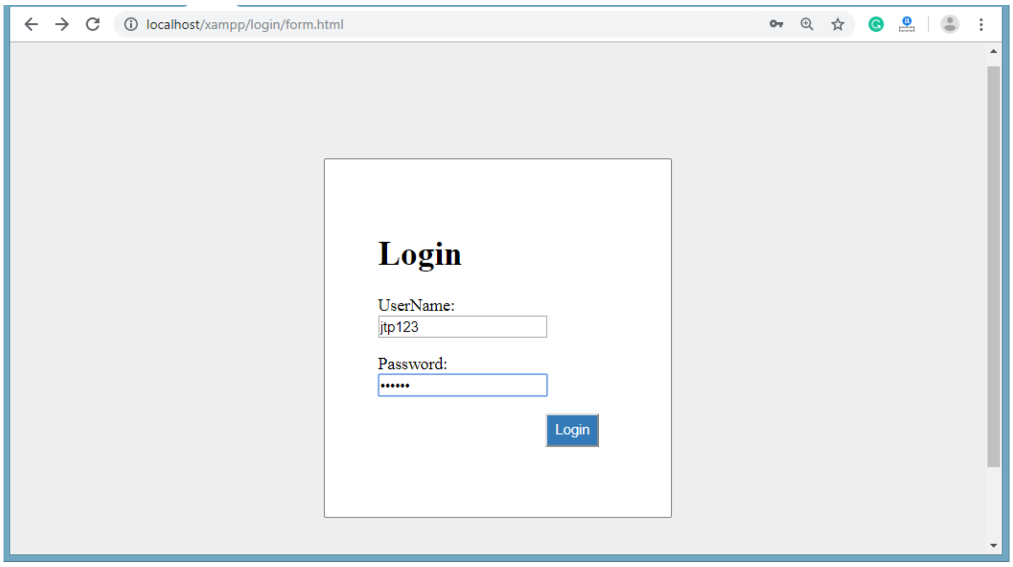
authentication.php

Now, we have our database setup, so we can go with the authentication of the user. This file handles the login form data that sent through the index.html file. It validates the data sent through the login form, if the username and password match with the database, then the login will be successful otherwise login will be failed.

```
<?php
   include('connection.php');
   $username = $_POST['user'];
   $password = $_POST['pass'];

      //to prevent from mysqli injection
      $username = stripcslashes($username);
      $password = stripcslashes($password);
      $username = mysqli_real_escape_string($con, $username);
      $password = mysqli_real_escape_string($con, $password);

      $sql = "select *from login where username = '$username' and password = '$password'";
      $result = mysqli_query($con, $sql);
      $row = mysqli_fetch_array($result, MYSQLI_ASSOC);
      $count = mysqli_num_rows($result);

      if($count == 1){
         echo "<h1><center> Login successful </center></h1>";
      }
      else{
         echo "<h1> Login failed. Invalid username or password.</h1>";
      }
?>
```

How to run the login form?

- To run the login form, open the xampp control panel and run the apache server and PHP.
- Now, type localhost/xampp/folder name/file name in the browser and press Enter key.
- All setup is done now. Enter the username and password in the login form and click the login button.


localhost/xampp/login/form.html

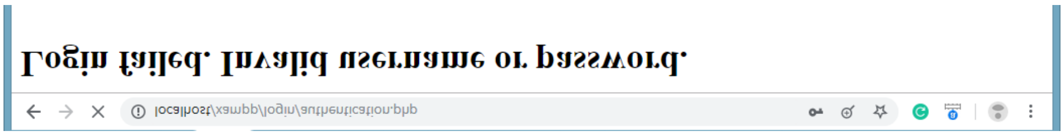
Login

UserName: jtp123

Password: ••••••

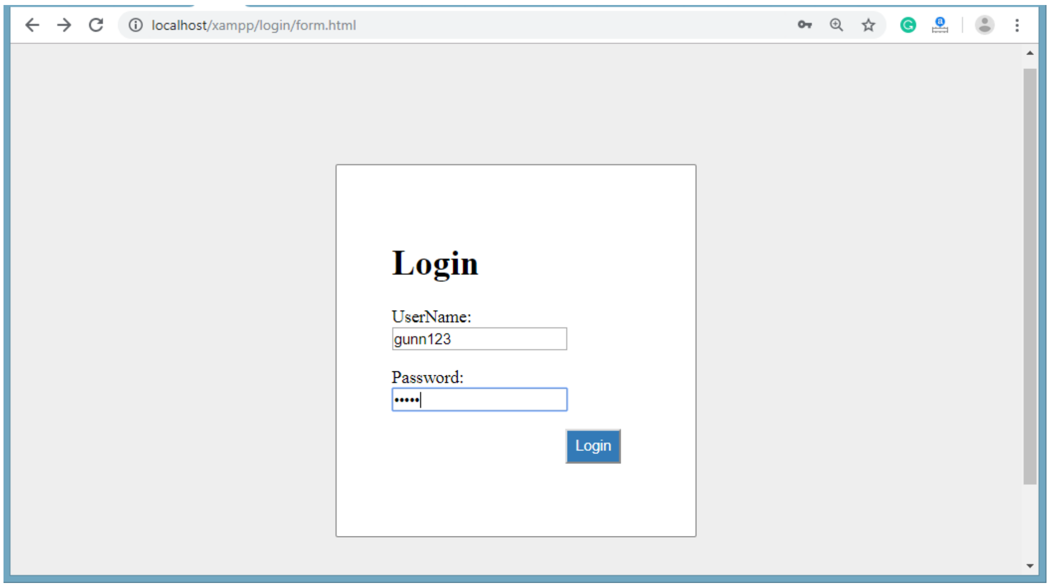
- Here, we have inserted an incorrect username, so the user is unable to log in, and it will give the login failed error.

Output:


localhost/xampp/login/authentication.php

Login failed. Invalid username or password.

- Now, we will provide correct value in the username and password. So, the user will be successfully logged in. See in the below example.

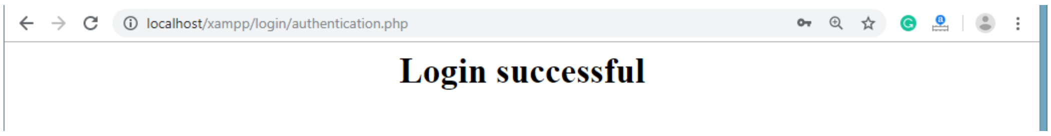

localhost/xampp/login/form.html

Login

UserName: gunn123

Password: •••••

Output


localhost/xampp/login/authentication.php

Login successful

Unit 4- XML

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation
- XML was designed to be both human- and machine-readable.
- XML is a software- and hardware-independent tool for storing and transporting data.

Differentiate between XML and HTML

Parameter	**XML**	**HTML**
Type of language	XML is a framework for specifying markup languages.	HTML is predefined markup language.
Language type	Case sensitive	Case insensitive
Structural details	It is provided	It is not provided.
Purpose	Transfer of data	Presentation of the data
Coding Errors	No coding errors are allowed.	Small errors are ignored.
Whitespace	You can use whitespaces in your code.	You can't use white spaces in your code.
Nesting	Should be done appropriately.	Does not have any effect on the code.

Driven by	XML is content driven	HTML is format driven
End of tags	The closing tag is essential in a well-formed XML document.	The closing tag is not always required. <HTML> tag needs an equivalent </HTML> tag but tag does not require </br> tag
Quotes	Quotes required around XML attribute values.	Quotes are not required for the values of attributes.
Object support	Objects have to be expressed by conventions. Mostly using attributes and elements.	Offers native object support
Null support	Need to use xsi:nil on elements in an XML instance document and also need to import the corresponding namespace.	Natively recognizes the null value.
Namespaces	XML provides support for namespaces. It helps you to remove the risk of name collisions when combining with other documents.	Does not support the concept of namespaces. Naming collisions can be avoided either using a prefix in an object member name or by nesting objects.
Formatting decisions	Require more significant effort to map application types to XML elements and attributes.	Provides direct mapping for application data.
Size	Documents are mostly lengthy in size, especially when an element-centric approach used in formatting.	The syntax is very brief and yields formatted text.
Parsing in	Requires an XML DOM implementation and application code	No extra application code required to parse text. For this

Javascript	to map text back into JavaScript objects.	purpose, you can use the eval function of JavaScript.
Learning curve	Very hard as you need to learn technologies like XPath, XML Schema, DOM, etc.	HTML is a simple technology stack that is familiar to developers.

Advantages of using XML

Here, are significant advantages of using XML:

- Makes documents transportable across systems and applications. With the help of XML, you can exchange data quickly between different platforms.
- XML separates the data from HTML
- XML simplifies platform change process

Disadvantages of using XML

Here, are few drawbacks of using XML:

- XML requires a processing application
- The XML syntax is very similar to other alternative 'text-based' data transmission formats which is sometimes confusing
- No intrinsic data type support
- The XML syntax is redundant
- Does not allow the user to create his tags.

XML Document Structure

XML documents must contain one **root** element that is the **parent** of all other elements:

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML Prolog

This line is called the XML **prolog**:

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

The XML prolog is optional. If it exists, it must come first in the document.

XML documents can contain international characters, like Norwegian øæå or French êèé.

To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.

UTF-8 is the default character encoding for XML documents.

UTF-8 is also the default encoding for HTML5, CSS, JavaScript, PHP, and SQL.

Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>salary < 1000</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>salary &lt; 1000</message>
```

There are 5 pre-defined entity references in XML:

&lt;	<	less than
&gt;	>	greater than

&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Only < and & are strictly illegal in XML, but it is a good habit to replace > with &gt; as well.

Comments in XML

The syntax for writing comments in XML is similar to that of HTML:

```
<!-- This is a comment -->
```

XML Elements

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

```
<price>29.99</price>
```

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

In the example above:

<title>, <author>, <year>, and <price> have **text content** because they contain text (like 29.99).

<bookstore> and <book> have **element contents**, because they contain elements.

<book> has an **attribute** (category="children").

XML Attributes

XML elements can have attributes, just like HTML.

Attributes are designed to contain data related to a specific element.

XML Attributes Must be Quoted

Attribute values must always be quoted. Either single or double quotes can be used.

For a person's gender, the <person> element can be written like this:

```
<person gender="female">
```

or like this:

```
<person gender='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

or you can use character entities:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

XML Elements vs. Attributes

Take a look at these examples:

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <gender>female</gender>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

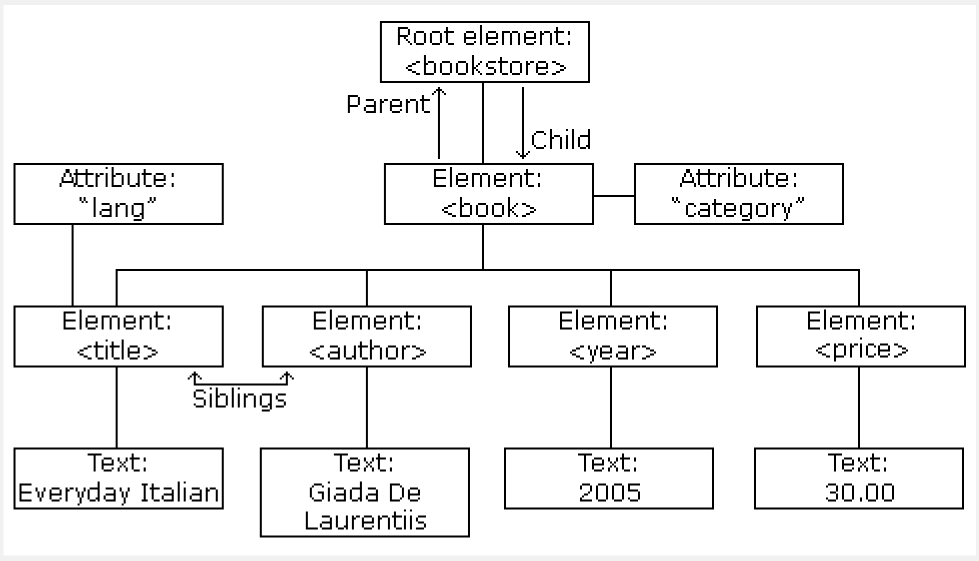
In the first example gender is an attribute. In the last, gender is an element. Both examples provide the same information.

There are no rules about when to use attributes or when to use elements in XML.

XML Tree

XML documents form a tree structure that starts at "the root" and branches to "the leaves".

XML Tree Structure



An Example XML Document

The image above represents books in this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML HttpRequest

The XMLHttpRequest Object

The XMLHttpRequest object can be used to request data from a web server.

The XMLHttpRequest object is **a developers dream**, because you can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

XMLHttpRequest Example

When you type a character in the input field below, an XMLHttpRequest is sent to the server, and some name suggestions are returned (from the server):

Sending an XMLHttpRequest

A common JavaScript syntax for using the XMLHttpRequest object looks much like this:

Example

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
      // Typical action to be performed when the document is ready:
      document.getElementById("demo").innerHTML = xhttp.responseText;
   }
};
```

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

Example Explained

The first line in the example above creates an **XMLHttpRequest** object:

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

The **onreadystatechange** property specifies a function to be executed every time the status of the XMLHttpRequest object changes:

```
xhttp.onreadystatechange = function()
```

When **readyState** property is 4 and the **status** property is 200, the response is ready:

```
if (this.readyState == 4 && this.status == 200)
```

The **responseText** property returns the server response as a text string.

The text string can be used to update a web page:

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

Old Versions of Internet Explorer (IE5 and IE6)

Old versions of Internet Explorer (IE5 and IE6) do not support the XMLHttpRequest object.

To handle IE5 and IE6, check if the browser supports the XMLHttpRequest object, or else create an ActiveXObject:

Example

```
if (window.XMLHttpRequest) {
   // code for modern browsers
   xmlhttp = new XMLHttpRequest();
 } else {
   // code for old IE browsers
   xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```

XML Parser

The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.

However, before an XML document can be accessed, it must be loaded into an XML DOM object.

All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

Parsing a Text String

This example parses a text string into an XML DOM object, and extracts the info from it with JavaScript:

Example

```
<html>
<body>

<p id="demo"></p>

<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```

Example Explained

A text string is defined:

```
text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";
```

An XML DOM parser is created:

```
parser = new DOMParser();
```

The parser creates a new XML DOM object using the text string:

```
xmlDoc = parser.parseFromString(text,"text/xml");
```

Old Versions of Internet Explorer

Old versions of Internet Explorer (IE5, IE6, IE7, IE8) do not support the DOMParser object.

To handle older versions of Internet Explorer, check if the browser supports the DOMParser object, or else create an ActiveXObject:

Example

```
if (window.DOMParser) {
  // code for modern browsers
  parser = new DOMParser();
  xmlDoc = parser.parseFromString(text,"text/xml");
} else {
  // code for old IE browsers
xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async = false;
  xmlDoc.loadXML(text);
}
```

What is an XML Parser?

To read and update, create and manipulate an XML document, you will need an XML parser.

In PHP there are two major types of XML parsers:

- Tree-Based Parsers
- Event-Based Parsers

Tree-Based Parsers

Tree-based parsers holds the entire document in Memory and transforms the XML document into a Tree structure. It analyzes the whole document, and provides access to the Tree elements (DOM).

This type of parser is a better option for smaller XML documents, but not for large XML document as it causes major performance issues.

Example of tree-based parsers:

- SimpleXML
- DOM

Event-Based Parsers

Event-based parsers do not hold the entire document in Memory, instead, they read in one node at a time and allow you to interact with in real time. Once you move onto the next node, the old one is thrown away.

This type of parser is well suited for large XML documents. It parses faster and consumes less memory.

Example of event-based parsers:

- XMLReader
- XML Expat Parser

The SimpleXML Parser

SimpleXML is a tree-based parser.

SimpleXML provides an easy way of getting an element's name, attributes and textual content if you know the XML document's structure or layout.

SimpleXML turns an XML document into a data structure you can iterate through like a collection of arrays and objects.

Compared to DOM or the Expat parser, SimpleXML takes a fewer lines of code to read text data from an element.

PHP SimpleXML - Read From String

The PHP `simplexml_load_string()` function is used to read XML data from a string.

Assume we have a variable that contains XML data, like this:

```
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>";
```

The example below shows how to use the `simplexml_load_string()` function to read XML data from a string:

Example

```
<?php
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>";

$xml=simplexml_load_string($myXMLData) or die("Error: Cannot create object");
print_r($xml);
?>
```

The output of the code above will be:

```
SimpleXMLElement Object ( [to] => Tove [from] => Jani [heading] => Reminder [body] =>
Don't forget me this weekend! )
```

Error Handling Tip: Use the libxml functionality to retrieve all XML errors when loading the document and then iterate over the errors. The following example tries to load a broken XML string:

Example

```
<?php
libxml_use_internal_errors(true);
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<document>
<user>John Doe</wronguser>
<email>john@example.com</wrongemail>
```

```
</document>";

$xml = simplexml_load_string($myXMLData);
if ($xml === false) {
  echo "Failed loading XML: ";
  foreach(libxml_get_errors() as $error) {
    echo "<br>", $error->message;
  }
} else {
  print_r($xml);
}
?>
```

The output of the code above will be:

```
Failed loading XML:
Opening and ending tag mismatch: user line 3 and wronguser
Opening and ending tag mismatch: email line 4 and wrongemail
```

PHP SimpleXML - Read From File

The PHP `simplexml_load_file()` function is used to read XML data from a file.

Assume we have an XML file called "note.xml", that looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The example below shows how to use the `simplexml_load_file()` function to read XML data from a file:

Example

```
<?php
$xml=simplexml_load_file("note.xml") or die("Error: Cannot create object");
print_r($xml);
?>
```

The output of the code above will be:

```
SimpleXMLElement Object ( [to] => Tove [from] => Jani [heading] => Reminder [body] =>
Don't forget me this weekend! )
```

Tip: The next chapter shows how to get/retrieve node values from an XML file with SimpleXML!

XML Expat Parser

The Expat parser is an event-based parser.

Look at the following XML fraction:

```
<from>Jani</from>
```

An event-based parser reports the XML above as a series of three events:

- Start element: from
- Start CDATA section, value: Jani
- Close element: from

The XML Expat Parser functions are part of the PHP core. There is no installation needed to use these functions.

The XML File

The XML file "note.xml" will be used in the example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Initializing the XML Expat Parser

We want to initialize the XML Expat Parser in PHP, define some handlers for different XML events, and then parse the XML file.

Example

```
<?php
// Initialize the XML parser
$parser=xml_parser_create();

// Function to use at the start of an element
function start($parser,$element_name,$element_attrs) {
 switch($element_name) {
   case "NOTE":
   echo "-- Note --<br>";
   break;
   case "TO":
   echo "To: ";
   break;
   case "FROM":
   echo "From: ";
   break;
   case "HEADING":
   echo "Heading: ";
   break;
   case "BODY":
   echo "Message: ";
 }
}

// Function to use at the end of an element
function stop($parser,$element_name) {
 echo "<br>";
}

// Function to use when finding character data
function char($parser,$data) {
 echo $data;
}

// Specify element handler
xml_set_element_handler($parser,"start","stop");

// Specify data handler
xml_set_character_data_handler($parser,"char");

// Open XML file
$fp=fopen("note.xml","r");

// Read data
while ($data=fread($fp,4096)) {
 xml_parse($parser,$data,feof($fp)) or
 die (sprintf("XML Error: %s at line %d",
 xml_error_string(xml_get_error_code($parser)),
 xml_get_current_line_number($parser)));
}

// Free the XML parser
xml_parser_free($parser);
?>
```

Example explained:

1. Initialize the XML parser with the xml_parser_create() function
2. Create functions to use with the different event handlers
3. Add the xml_set_element_handler() function to specify which function will be executed when the parser encounters the opening and closing tags
4. Add the xml_set_character_data_handler() function to specify which function will execute when the parser encounters character data
5. Parse the file "note.xml" with the xml_parse() function
6. In case of an error, add xml_error_string() function to convert an XML error to a textual description
7. Call the xml_parser_free() function to release the memory allocated with the xml_parser_create() function

The XMLHttpRequest Object

The XMLHttpRequest Object has a built in XML Parser.

The **responseText** property returns the response as a string.

The **responseXML** property returns the response as an XML DOM object.

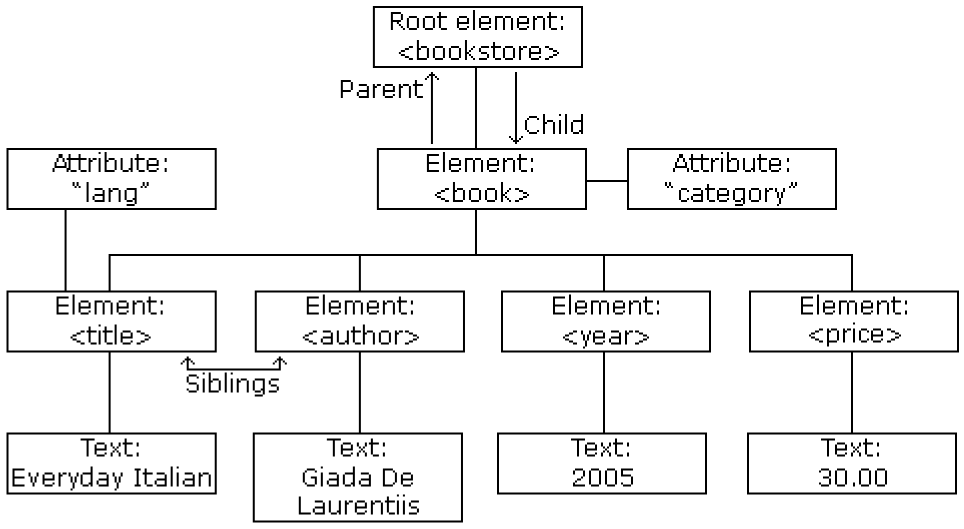
If you want to use the response as an XML DOM object, you can use the responseXML property.

Example

Request the file cd_catalog.xml and use the response as an XML DOM object:

```
xmlDoc = xmlhttp.responseXML;
txt = "";
x = xmlDoc.getElementsByTagName("ARTIST");
for (i = 0; i < x.length; i++) {
    txt += x[i].childNodes[0].nodeValue + "<br>";
}
document.getElementById("demo").innerHTML = txt;
```

XML DOM



What is the DOM?

The DOM defines a standard for accessing and manipulating documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.

The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.

The HTML DOM

All HTML elements can be accessed through the HTML DOM.

This example changes the value of an HTML element with id="demo":

Example

```
<h1 id="demo">This is a Heading</h1>

<button type="button"
```

```
onclick="document.getElementById('demo').innerHTML = 'Hello World!'">Click Me!
</button>
```

The XML DOM

All XML elements can be accessed through the XML DOM.

Books.xml

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

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

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

</bookstore>
```

This code retrieves the text value of the first <title> element in an XML document:

Example

```
txt = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
```

The XML DOM is a standard for how to get, change, add, and delete XML elements.

This example loads a text string into an XML DOM object, and extracts the info from it with JavaScript:

Example

```
<html>
<body>

<p id="demo"></p>
```

```
<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```

Unit 5-Web services

Web services are web application components.

Web services can be published, found, and used on the Web.

This tutorial introduces WSDL, SOAP, RDF, and RSS.

WSDL

- WSDL stands for Web Services Description Language
- WSDL is an XML-based language for describing Web services.
- WSDL is a W3C recommendation

SOAP

- SOAP stands for Simple Object Access Protocol
- SOAP is an XML based protocol for accessing Web Services.
- SOAP is based on XML
- SOAP is a W3C recommendation

RDF

- RDF stands for Resource Description Framework

- RDF is a framework for describing resources on the web
- RDF is written in XML
- RDF is a W3C Recommendation

RSS

- RSS stands for Really Simple Syndication
- RSS allows you to syndicate your site content
- RSS defines an easy way to share and view headlines and content
- RSS files can be automatically updated
- RSS allows personalized views for different sites
- RSS is written in XML

What You Should Already Know

Before you study web services you should have a basic understanding of XML and XML Namespaces.

Web Services

- Web services are application components
- Web services communicate using open protocols
- Web services are self-contained and self-describing
- Web services can be discovered using UDDI
- Web services can be used by other applications
- HTTP and XML is the basis for Web services

Interoperability has Highest Priority

When all major platforms could access the Web using Web browsers, different platforms couldn't interact. For these platforms to work together, Web-applications were developed.

Web-applications are simply applications that run on the web. These are built around the Web browser standards and can be used by any browser on any platform.

Web Services take Web-applications to the Next Level

By using Web services, your application can publish its function or message to the rest of the world.

Web services use XML to code and to decode data, and SOAP to transport it (using open protocols).

With Web services, your accounting department's Win 2k server's billing system can connect with your IT supplier's UNIX server.

Web Services have Two Types of Uses

Reusable application-components.

There are things applications need very often. So why make these over and over again?

Web services can offer application-components like: currency conversion, weather reports, or even language translation as services.

Connect existing software.

Web services can help to solve the interoperability problem by giving different applications a way to link their data.

With Web services you can exchange data between different applications and different platforms.

Any application can have a Web Service component.

Web Services can be created regardless of programming language.

A Web Service Example

In the following example we will use ASP.NET to create a simple Web Service that converts the temperature from Fahrenheit to Celsius, and vice versa:

```
<%@ WebService Language="VBScript" Class="TempConvert" %>

Imports System
Imports System.Web.Services

Public Class TempConvert :Inherits WebService

<WebMethod()> Public Function FahrenheitToCelsius(ByVal Fahrenheit As String) As String
  dim fahr
  fahr=trim(replace(Fahrenheit,",","."))
  if fahr="" or IsNumeric(fahr)=false then return "Error"
  return ((((fahr) - 32) / 9) * 5)
end function

<WebMethod()> Public Function CelsiusToFahrenheit(ByVal Celsius As String) As String
  dim cel
  cel=trim(replace(Celsius,",","."))
  if cel="" or IsNumeric(cel)=false then return "Error"
  return ((((cel) * 9) / 5) + 32)
end function

end class
```

This document is saved as an .asmx file. This is the ASP.NET file extension for XML Web Services.

Example Explained

Note: To run this example, you will need a .NET server.

The first line in the example states that this is a Web Service, written in VBScript, and has the class name "TempConvert":

```
<%@ WebService Language="VBScript" Class="TempConvert" %>
```

The next lines import the namespace "System.Web.Services" from the .NET framework:

```
Imports System
Imports System.Web.Services
```

The next line defines that the "TempConvert" class is a WebService class type:

```
Public Class TempConvert :Inherits WebService
```

The next steps are basic VB programming. This application has two functions. One to convert from Fahrenheit to Celsius, and one to convert from Celsius to Fahrenheit.

The only difference from a normal application is that this function is defined as a "WebMethod()".

Use "WebMethod()" to convert the functions in your application into web services:

```
<WebMethod()> Public Function FahrenheitToCelsius(ByVal Fahrenheit As String) As String
  dim fahr
  fahr=trim(replace(Fahrenheit,",","."))
  if fahr="" or IsNumeric(fahr)=false then return "Error"
  return ((((fahr) - 32) / 9) * 5)
end function

<WebMethod()> Public Function CelsiusToFahrenheit(ByVal Celsius As String) As String
  dim cel
  cel=trim(replace(Celsius,",","."))
  if cel="" or IsNumeric(cel)=false then return "Error"
  return ((((cel) * 9) / 5) + 32)
end function
```

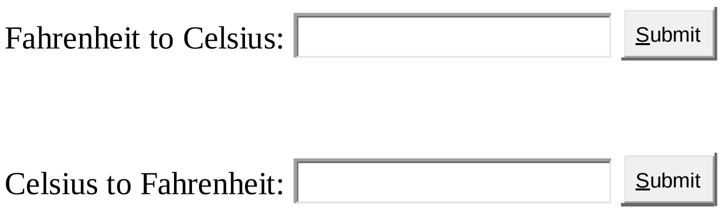
Then, end the class:

```
end class
```

Publish the .asmx file on a server with .NET support, and you will have your first working Web Service.

Put the Web Service on Your Web Site

Using a form and the HTTP POST method, you can put the web service on your site, like this:


Fahrenheit to Celsius:

Celsius to Fahrenheit:

How To Do It

Here is the code to add the Web Service to a web page:

```
<form action='tempconvert.asmx/FahrenheitToCelsius'
method="post" target="_blank">
<table>
  <tr>
    <td>Fahrenheit to Celsius:</td>
    <td>
    <input class="frmInput" type="text" size="30" name="Fahrenheit">
    </td>
  </tr>
  <tr>
    <td></td>
    <td align="right">
     <input type="submit" value="Submit" class="button">
     </td>
  </tr>
</table>
</form>

<form action='tempconvert.asmx/CelsiusToFahrenheit'
method="post" target="_blank">
<table>
  <tr>
    <td>Celsius to Fahrenheit:</td>
    <td>
    <input class="frmInput" type="text" size="30" name="Celsius">
    </td>
  </tr>
  <tr>
    <td></td>
    <td align="right">
    <input type="submit" value="Submit" class="button">
    </td>
  </tr>
</table>
</form>
```

XML WSDL

- WSDL stands for Web Services Description Language
- WSDL is used to describe web services
- WSDL is written in XML
- WSDL is a W3C recommendation from 26. June 2007

WSDL Documents

An WSDL document describes a web service. It specifies the location of the service, and the methods of the service, using these major elements:

Element	Description
<types>	Defines the (XML Schema) data types used by the web service
<message>	Defines the data elements for each operation
<portType>	Describes the operations that can be performed and the messages involved.
<binding>	Defines the protocol and data format for each port type

The main structure of a WSDL document looks like this:

```
<definitions>

<types>
  data type definitions........
```

```
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations......
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```

WSDL Example

This is a simplified fraction of a WSDL document:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

In this example the **<portType>** element defines "glossaryTerms" as the name of a **port**, and "getTerm" as the name of an **operation**.

The "getTerm" operation has an **input message** called "getTermRequest" and an **output message** called "getTermResponse".

The **<message>** elements define the **parts** of each message and the associated data types.

The <portType> Element

The <portType> element defines **a web service**, the **operations** that can be performed, and the **messages** that are involved.

The request-response type is the most common operation type, but WSDL defines four types:

Type	Definition
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

---

WSDL One-Way Operation

A one-way operation example:

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

In the example above, the portType "glossaryTerms" defines a one-way operation called "setTerm".

The "setTerm" operation allows input of new glossary terms messages using a "newTermValues" message with the input parameters "term" and "value". However, no output is defined for the operation.

WSDL Request-Response Operation

A request-response operation example:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

In the example above, the portType "glossaryTerms" defines a request-response operation called "getTerm".

The "getTerm" operation requires an input message called "getTermRequest" with a parameter called "term", and will return an output message called "getTermResponse" with a parameter called "value".

WSDL Binding to SOAP

WSDL bindings defines the message format and protocol details for a web service.

A request-response operation example:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
```

```
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

The **binding** element has two attributes - name and type.

The name attribute (you can use any name you want) defines the name of the binding, and the type attribute points to the port for the binding, in this case the "glossaryTerms" port.

The **soap:binding** element has two attributes - style and transport.

The style attribute can be "rpc" or "document". In this case we use document. The transport attribute defines the SOAP protocol to use. In this case we use HTTP.

The **operation** element defines each operation that the portType exposes.

XML Soap

- SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol
- SOAP is an application communication protocol
- SOAP is a format for sending and receiving messages
- SOAP is platform independent
- SOAP is based on XML
- SOAP is a W3C recommendation

Why SOAP?

It is important for web applications to be able to communicate over the Internet.

The best way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.

SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

SOAP Building Blocks

A SOAP message is an ordinary XML document containing the following elements:

- An Envelope element that identifies the XML document as a SOAP message
- A Header element that contains header information
- A Body element that contains call and response information
- A Fault element containing errors and status information

All the elements above are declared in the default namespace for the SOAP envelope:

http://www.w3.org/2003/05/soap-envelope/

and the default namespace for SOAP encoding and data types is:

http://www.w3.org/2003/05/soap-encoding

Syntax Rules

Here are some important syntax rules:

- A SOAP message MUST be encoded using XML
- A SOAP message MUST use the SOAP Envelope namespace
- A SOAP message must NOT contain a DTD reference
- A SOAP message must NOT contain XML Processing Instructions

Skeleton SOAP Message

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

The SOAP Envelope Element

The required SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message.

Example

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

The xmlns:soap Namespace

Notice the xmlns:soap namespace in the example above. It should always have the value of: "http://www.w3.org/2003/05/soap-envelope/".

The namespace defines the Envelope as a SOAP Envelope.

If a different namespace is used, the application generates an error and discards the message.

The encodingStyle Attribute

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and applies to the element's contents and all child elements.

A SOAP message has no default encoding.

Syntax

soap:encodingStyle="*URI*"

Example

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

The SOAP Header Element

The optional SOAP Header element contains application-specific information (like authentication, payment, etc) about the SOAP message.

If the Header element is present, it must be the first child element of the Envelope element.

Note: All immediate child elements of the Header element must be namespace-qualified.

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="https://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

The example above contains a header with a "Trans" element, a "mustUnderstand" attribute with a value of 1, and a value of 234.

SOAP defines three attributes in the default namespace. These attributes are: mustUnderstand, actor, and encodingStyle.

The attributes defined in the SOAP Header defines how a recipient should process the SOAP message.

The mustUnderstand Attribute

The SOAP mustUnderstand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process.

If you add mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it will fail when processing the Header.

Syntax

```
soap:mustUnderstand="0|1"
```

Example

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="https://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

The actor Attribute

A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. However, not all parts of a SOAP message may be intended for the ultimate endpoint, instead, it may be intended for one or more of the endpoints on the message path.

The SOAP actor attribute is used to address the Header element to a specific endpoint.

Syntax

soap:actor="*URI*"

Example

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="https://www.w3schools.com/transaction/"
  soap:actor="https://www.w3schools.com/code/">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

The encodingStyle Attribute

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements.

A SOAP message has no default encoding.

Syntax

soap:encodingStyle="*URI*"

The SOAP Body Element

The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.

Immediate child elements of the SOAP Body element may be namespace-qualified.

Example

```
<?xml version="1.0"?>

<soap:Envelope
```

```
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="https://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

The example above requests the price of apples. Note that the m:GetPrice and the Item elements above are application-specific elements. They are not a part of the SOAP namespace.

A SOAP response could look something like this:

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="https://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

The SOAP Fault Element

The optional SOAP Fault element is used to indicate error messages.

The SOAP Fault element holds errors and status information for a SOAP message.

If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.

The SOAP Fault element has the following sub elements:

Sub Element	Description
<faultcode>	A code for identifying the fault
<faultstring>	A human readable explanation of the fault
<faultactor>	Information about who caused the fault to happen
<detail>	Holds application specific error information related to the Body element

SOAP Fault Codes

The faultcode values defined below must be used in the faultcode element when describing faults:

Error	Description
VersionMismatch	Found an invalid namespace for the SOAP Envelope element
MustUnderstand	An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
Client	The message was incorrectly formed or contained incorrect information
Server	There was a problem with the server so the message could not

	proceed

The HTTP Protocol

HTTP communicates over TCP/IP. An HTTP client connects to an HTTP server using TCP. After establishing a connection, the client can send an HTTP request message to the server:

```
POST /item HTTP/1.1
Host: 189.123.255.239
Content-Type: text/plain
Content-Length: 200
```

The server then processes the request and sends an HTTP response back to the client. The response contains a status code that indicates the status of the request:

```
200 OK
Content-Type: text/plain
Content-Length: 200
```

In the example above, the server returned a status code of 200. This is the standard success code for HTTP.

If the server could not decode the request, it could have returned something like this:

```
400 Bad Request
Content-Length: 0
```

SOAP Binding

The SOAP specification defines the structure of the SOAP messages, not how they are exchanged. This gap is filled by what is called "SOAP Bindings". SOAP bindings are mechanisms which allow SOAP messages to be effectively exchanged using a transport protocol.

Most SOAP implementations provide bindings for common transport protocols, such as HTTP or SMTP.

HTTP is synchronous and widely used. A SOAP HTTP request specifies at least two HTTP headers: Content-Type and Content-Length.

SMTP is asynchronous and is used in last resort or particular cases.

Java implementations of SOAP usually provide a specific binding for the JMS (Java Messaging System) protocol.

Content-Type

The Content-Type header for a SOAP request and response defines the MIME type for the message and the character encoding (optional) used for the XML body of the request or response.

Syntax

```
Content-Type: MIMEType; charset=character-encoding
```

Example

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
```

Content-Length

The Content-Length header for a SOAP request and response specifies the number of bytes in the body of the request or response.

Syntax

```
Content-Length: bytes
```

Example

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 250
```

A SOAP Example

In the example below, a GetStockPrice request is sent to a server. The request has a StockName parameter, and a Price parameter that will be returned in the response. The namespace for the function is defined in "http://www.example.org/stock".

A SOAP request:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

The SOAP response:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Unit 6- AJAX

AJAX is a developer's dream, because you can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

AJAX Example

HTML Page

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

<div id="demo">
  <h2>Let AJAX change this text</h2>
  <button type="button" onclick="loadDoc()">Change Content</button>
</div>

</body>
</html>
```

The HTML page contains a <div> section and a <button>.

The <div> section is used to display information from a server.

The <button> calls a function (if it is clicked).

The function requests data from a web server and displays it:

Function loadDoc()

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

"ajax_info.txt" looks like this:

```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
<p>AJAX is a technique for accessing web servers from a web page.</p>
<p>AJAX stands for Asynchronous JavaScript And XML.</p>
```

What is AJAX?

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

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

AJAX - The XMLHttpRequest Object

The keystone of AJAX is the XMLHttpRequest object.

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

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

Create an XMLHttpRequest Object

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

Syntax for creating an XMLHttpRequest object:

variable = new XMLHttpRequest();

Example

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

Access Across Domains

For security reasons, modern browsers do not allow access across domains.

This means that both the web page and the XML file it tries to load, must be located on the same server.

The examples on W3Schools all open XML files located on the W3Schools domain.

If you want to use the example above on one of your own web pages, the XML files you load must be located on your own server.

Old Versions of Internet Explorer (IE5 and IE6)

Old versions of Internet Explorer (IE5 and IE6) use an ActiveX object instead of the XMLHttpRequest object:

variable = new ActiveXObject("Microsoft.XMLHTTP");

To handle IE5 and IE6, check if the browser supports the XMLHttpRequest object, or else create an ActiveX object:

Example

```
if (window.XMLHttpRequest) {
   // code for modern browsers
   xmlhttp = new XMLHttpRequest();
 } else {
   // code for old IE browsers
   xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
```

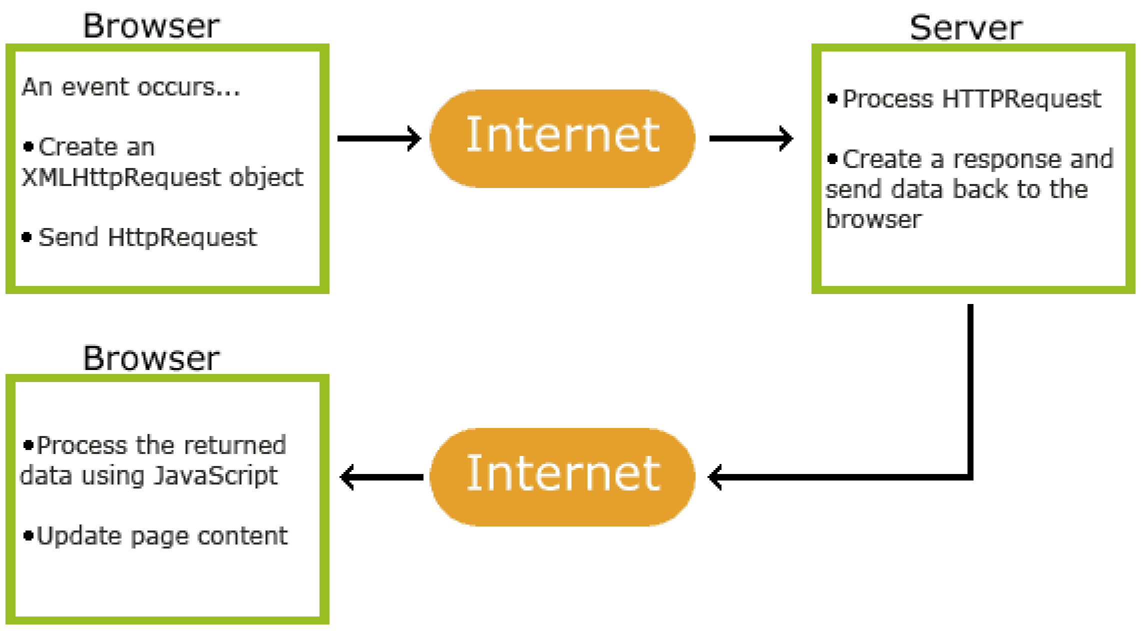
XMLHttpRequest Object Methods

Method	**Description**
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method*,*url*,*async*,*user*,*psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

XMLHttpRequest Object Properties

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

How AJAX Works


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

- 1. An event occurs in a web page (the page is loaded, a button is clicked)
- 2. An XMLHttpRequest object is created by JavaScript
- 3. The XMLHttpRequest object sends a request to a web server
- 4. The server processes the request
- 5. The server sends a response back to the web page
- 6. The response is read by JavaScript
- 7. Proper action (like page update) is performed by JavaScript

AJAX - Send a Request To a Server

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

Send a Request To a Server

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

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

Method	**Description**

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

GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

- A cached file is not an option (update a file or database on the server).
- Sending a large amount of data to the server (POST has no size limitations).
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

GET Requests

A simple GET request:

Example

```
xhttp.open("GET", "demo_get.asp", true);
xhttp.send();
```

In the example above, you may get a cached result. To avoid this, add a unique ID to the URL:

Example

```
xhttp.open("GET", "demo_get.asp?t=" + Math.random(), true);
xhttp.send();
```

If you want to send information with the GET method, add the information to the URL:

Example

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);
xhttp.send();
```

POST Requests

A simple POST request:

Example

```
xhttp.open("POST", "demo_post.asp", true);
xhttp.send();
```

To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify the data you want to send in the send() method:

Example

```
xhttp.open("POST", "demo_post2.asp", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

Method	**Description**
setRequestHeader(*header*, *value*)	Adds HTTP headers to the request *header*: specifies the header name *value*: specifies the header value

---

The url - A File On a Server

The url parameter of the open() method, is an address to a file on a server:

```
xhttp.open("GET", "ajax_test.asp", true);
```

The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

Asynchronous - True or False?

Server requests should be sent asynchronously.

The async parameter of the open() method should be set to true:

```
xhttp.open("GET", "ajax_test.asp", true);
```

By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:

- execute other scripts while waiting for server response
- deal with the response after the response is ready

The onreadystatechange Property

With the XMLHttpRequest object you can define a function to be executed when the request receives an answer.

The function is defined in the **onreadystatechange** property of the XMLHttpResponse object:

Example

```
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML = this.responseText;
  }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

You will learn more about onreadystatechange in a later chapter.

Synchronous Request

To execute a synchronous request, change the third parameter in the open() method to false:

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

Sometimes async = false are used for quick testing. You will also find synchronous requests in older JavaScript code.

Since the code will wait for server completion, there is no need for an onreadystatechange function:

Example

```
<!DOCTYPE html>

<html>

<body>

<h1>The XMLHttpRequest Object</h1>

<p id="demo">Let AJAX change this text.</p>

<button type="button" onclick="loadDoc()">Change Content</button>

<script>

function loadDoc() {

  var xhttp = new XMLHttpRequest();

  xhttp.open("GET", "ajax_info.txt", false);

  xhttp.send();

  document.getElementById("demo").innerHTML = xhttp.responseText;

}

</script>

</body>

</html>
```

Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready. If the server is busy or slow, the application will hang or stop.

Synchronous XMLHttpRequest is in the process of being removed from the web standard, but this process can take many years.

Modern developer tools are encouraged to warn about using synchronous requests and may throw an InvalidAccessError exception when it occurs.

AJAX - Server Response

The onreadystatechange Property

The **readyState** property holds the status of the XMLHttpRequest.

The **onreadystatechange** property defines a function to be executed when the readyState changes.

The **status** property and the **statusText** property holds the status of the XMLHttpRequest object.

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The onreadystatechange function is called every time the readyState changes.

When readyState is 4 and status is 200, the response is ready:

Example

```
function loadDoc() {
   var xhttp = new XMLHttpRequest();
   xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
         document.getElementById("demo").innerHTML =
         this.responseText;
      }
   };
   xhttp.open("GET", "ajax_info.txt", true);
   xhttp.send();
}
```

The onreadystatechange event is triggered four times (1-4), one time for each change in the readyState.

Using a Callback Function

A callback function is a function passed as a parameter to another function.

If you have more than one AJAX task in a website, you should create one function for executing the XMLHttpRequest object, and one callback function for each AJAX task.

The function call should contain the URL and what function to call when the response is ready.

Example

```
loadDoc("url-1", myFunction1);

loadDoc("url-2", myFunction2);

function loadDoc(url, cFunction) {
  var xhttp;
  xhttp=new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      cFunction(this);
    }
 };
  xhttp.open("GET", url, true);
  xhttp.send();
}

function myFunction1(xhttp) {
  // action goes here
}
function myFunction2(xhttp) {
  // action goes here
}
```

Server Response Properties

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

Server Response Methods

Method	Description
getResponseHeader()	Returns specific header information from the server resource
getAllResponseHeaders()	Returns all the header information from the server resource

The responseText Property

The **responseText** property returns the server response as a JavaScript string, and you can use it accordingly:

Example

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

The responseXML Property

The XML HttpRequest object has an in-built XML parser.

The **responseXML** property returns the server response as an XML DOM object.

Using this property you can parse the response as an XML DOM object:

Example

Request the file cd_catalog.xml and parse the response:

```
xmlDoc = xhttp.responseXML;
txt = "";
x = xmlDoc.getElementsByTagName("ARTIST");
for (i = 0; i < x.length; i++) {
 txt += x[i].childNodes[0].nodeValue + "<br>";
 }
document.getElementById("demo").innerHTML = txt;
xhttp.open("GET", "cd_catalog.xml", true);
xhttp.send();
```

The getAllResponseHeaders() Method

The **getAllResponseHeaders()** method returns all header information from the server response.

Example

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status == 200) {
   document.getElementById("demo").innerHTML =
   this.getAllResponseHeaders();
 }
};
```

The getResponseHeader() Method

The **getResponseHeader()** method returns specific header information from the server response.

Example

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML =
    this.getResponseHeader("Last-Modified");
  }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

AJAX XML Example

AJAX can be used for interactive communication with an XML file.

AJAX XML Example

The following example will demonstrate how a web page can fetch information from an XML file with AJAX:

Example

Get CD info

Example Explained

When a user clicks on the "Get CD info" button above, the loadDoc() function is executed.

The loadDoc() function creates an XMLHttpRequest object, adds the function to be executed when the server response is ready, and sends the request off to the server.

When the server response is ready, an HTML table is built, nodes (elements) are extracted from the XML file, and it finally updates the element "demo" with the HTML table filled with XML data:

LoadXMLDoc()

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
```

```
    myFunction(this);
    }
  };
  xhttp.open("GET", "cd_catalog.xml", true);
  xhttp.send();
}
function myFunction(xml) {
  var i;
  var xmlDoc = xml.responseXML;
  var table="<tr><th>Title</th><th>Artist</th></tr>";
  var x = xmlDoc.getElementsByTagName("CD");
  for (i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
```

AJAX PHP Example

AJAX is used to create more interactive applications.

AJAX PHP Example

The following example demonstrates how a web page can communicate with a web server while a user types characters in an input field:

Example

Start typing a name in the input field below:

First name: [   ]   Suggestions:

Example Explained

In the example above, when a user types a character in the input field, a function called "showHint()" is executed.

The function is triggered by the onkeyup event.

Here is the HTML code:

Example

```
<html>
<head>
<script>
function showHint(str) {
   if (str.length == 0) {
      document.getElementById("txtHint").innerHTML = "";
      return;
   } else {
      var xmlhttp = new XMLHttpRequest();
      xmlhttp.onreadystatechange = function() {
         if (this.readyState == 4 && this.status == 200) {
            document.getElementById("txtHint").innerHTML = this.responseText;
         }
      };
      xmlhttp.open("GET", "gethint.php?q=" + str, true);
      xmlhttp.send();
   }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field below:</b></p>
<form>
First name: <input type="text" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Code explanation:

First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint placeholder and exit the function.

However, if the input field is not empty, do the following:

- Create an XMLHttpRequest object

- Create the function to be executed when the server response is ready
- Send the request off to a PHP file (gethint.php) on the server
- Notice that q parameter is added gethint.php?q="+str
- The str variable holds the content of the input field

The PHP File - "gethint.php"

The PHP file checks an array of names, and returns the corresponding name(s) to the browser:

```
<?php
// Array with names
$a[] = "Anna";
$a[] = "Brittany";
$a[] = "Cinderella";
$a[] = "Diana";
$a[] = "Eva";
$a[] = "Fiona";
$a[] = "Gunda";
$a[] = "Hege";
$a[] = "Inga";
$a[] = "Johanna";
$a[] = "Kitty";
$a[] = "Linda";
$a[] = "Nina";
$a[] = "Ophelia";
$a[] = "Petunia";
$a[] = "Amanda";
$a[] = "Raquel";
$a[] = "Cindy";
$a[] = "Doris";
$a[] = "Eve";
$a[] = "Evita";
$a[] = "Sunniva";
$a[] = "Tove";
$a[] = "Unni";
$a[] = "Violet";
$a[] = "Liza";
$a[] = "Elizabeth";
$a[] = "Ellen";
$a[] = "Wenche";
$a[] = "Vicky";

// get the q parameter from URL
$q = $_REQUEST["q"];

$hint = "";
```

```
// lookup all hints from array if $q is different from ""
if ($q !== "") {
   $q = strtolower($q);
   $len=strlen($q);
   foreach($a as $name) {
       if (stristr($q, substr($name, 0, $len))) {
           if ($hint === "") {
               $hint = $name;
           } else {
               $hint .= ", $name";
           }
       }
   }
}

// Output "no suggestion" if no hint was found or output correct values
echo $hint === "" ? "no suggestion" : $hint;
?>
```

AJAX Database Example

AJAX can be used for interactive communication with a database.

AJAX Database Example

The following example will demonstrate how a web page can fetch information from a database with AJAX:

Example

Customer info will be listed here...

Example Explained - The showCustomer() Function

When a user selects a customer in the dropdown list above, a function called "showCustomer()" is executed. The function is triggered by the "onchange" event:

showCustomer

```
function showCustomer(str) {
  var xhttp;
  if (str == "") {
    document.getElementById("txtHint").innerHTML = "";
    return;
  }
  xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
    document.getElementById("txtHint").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "getcustomer.php?q="+str, true);
  xhttp.send();
}
```

The showCustomer() function does the following:

- Check if a customer is selected
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The AJAX Server Page

The page on the server called by the JavaScript above is an PHP file called "getcustomer.php".

The source code in "getcustomer.php" runs a query against a database, and returns the result in an HTML table:

```
<?php
$mysqli = new mysqli("servername", "username", "password", "dbname");
if($mysqli->connect_error) {
  exit('Could not connect');
}

$sql = "SELECT customerid, companyname, contactname, address, city, postalcode, country
FROM customers WHERE customerid = ?";

$stmt = $mysqli->prepare($sql);
$stmt->bind_param("s", $_GET['q']);
$stmt->execute();
$stmt->store_result();
```

```
$stmt->bind_result($cid, $cname, $name, $adr, $city, $pcode, $country);
$stmt->fetch();
$stmt->close();

echo "<table>";
echo "<tr>";
echo "<th>CustomerID</th>";
echo "<td>" . $cid . "</td>";
echo "<th>CompanyName</th>";
echo "<td>" . $cname . "</td>";
echo "<th>ContactName</th>";
echo "<td>" . $name . "</td>";
echo "<th>Address</th>";
echo "<td>" . $adr . "</td>";
echo "<th>City</th>";
echo "<td>" . $city . "</td>";
echo "<th>PostalCode</th>";
echo "<td>" . $pcode . "</td>";
echo "<th>Country</th>";
echo "<td>" . $country . "</td>";
echo "</tr>";
echo "</table>";
?>
```

XML Applications

Display XML Data in an HTML Table

This example loops through each <CD> element, and displays the values of the <ARTIST> and the <TITLE> elements in an HTML table:

Example

```
<html>
<head>
<style>
table, th, td {
  border: 1px solid black;
  border-collapse:collapse;
}
th, td {
  padding: 5px;
}
</style>
```

```
</head>
<body>

<table id="demo"></table>

<script>
function loadXMLDoc() {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      myFunction(this);
    }
  };
  xmlhttp.open("GET", "cd_catalog.xml", true);
  xmlhttp.send();
}
function myFunction(xml) {
  var i;
  var xmlDoc = xml.responseXML;
  var table="<tr><th>Artist</th><th>Title</th></tr>";
  var x = xmlDoc.getElementsByTagName("CD");
  for (i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```

Display the First CD in an HTML div Element

This example uses a function to display the first CD element in an HTML element with id="showCD":

Example

```
displayCD(0);

function displayCD(i) {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            myFunction(this, i);
        }
    };
    xmlhttp.open("GET", "cd_catalog.xml", true);
    xmlhttp.send();
}

function myFunction(xml, i) {
    var xmlDoc = xml.responseXML;
    x = xmlDoc.getElementsByTagName("CD");
    document.getElementById("showCD").innerHTML =
    "Artist: " +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "<br>Title: " +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "<br>Year: " +
    x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```

Navigate Between the CDs

To navigate between the CDs, in the example above, add a next() and previous() function:

Example

```
function next() {
  // display the next CD, unless you are on the last CD
  if (i < x.length-1) {
    i++;
    displayCD(i);
  }
}

function previous() {
  // display the previous CD, unless you are on the first CD
  if (i > 0) {
  i--;
  displayCD(i);
  }
}
```

Show Album Information When Clicking On a CD

The last example shows how you can display album information when the user clicks on a CD:

Example

```
function displayCD(i) {
    document.getElementById("showCD").innerHTML =
    "Artist: " +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "<br>Title: " +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "<br>Year: " +
    x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```