

Web Programming

YJ – Oct 2015

PHP

- ❖ File Access
- ❖ XML & JSON
- ❖ REGULAR EXPRESSIONS
- ❖ State Maintaining
- ❖ Object Oriented
- ❖ MVC

File Access

Open a file

```
$filename = "/home/user/guest/tmp.txt";  
$file = fopen( $filename, "r" );
```

Mode	Purpose
r	Opens the file for reading only.Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing.Places the file pointer at the beginning of the file.
w	Opens the file for writing only.Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
w+	Opens the file for reading and writing only.Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
a	Opens the file for writing only.Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
a+	Opens the file for reading and writing only.Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

File Access

Read a file

```
$filename = "/home/user/guest/tmp.txt";  
$file = fopen( $filename, "r" );  
$filesize = filesize( $filename );  
$filetext = fread( $file, $filesize );
```

Write a file

```
fwrite( $file, "This is a simple test\n" );
```

Close a file

```
fclose( $file );
```

Upload Files

The process of uploading a file follows these steps

- ❖ The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- ❖ The user clicks the browse button and selects a file to upload from the local PC.
- ❖ The full path to the selected file appears in the text filed then the user clicks the submit button.
- ❖ The selected file is sent to the temporary directory on the server.
- ❖ The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- ❖ The PHP script confirms the success to the user.

There is one global PHP variable called **\$_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file.

- ❖ **\$_FILES['file']['tmp_name']**- the uploaded file in the temporary directory on the web server.
- ❖ **\$_FILES['file']['name']** - the actual name of the uploaded file.
- ❖ **\$_FILES['file']['size']** - the size in bytes of the uploaded file.
- ❖ **\$_FILES['file']['type']** - the MIME type of the uploaded file.
- ❖ **\$_FILES['file']['error']** - the error code associated with this file upload.

XML

- ❖ XML stands for EXtensible Markup Language
- ❖ XML is a markup language much like HTML
- ❖ XML was designed to describe data, not to display data
- ❖ XML tags are not predefined. You must define your own tags
- ❖ XML is designed to be self-descriptive

```
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>
```

PHP 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.

<code>simplexml_load_file()</code>	Converts an XML file into a SimpleXMLElement object
<code>simplexml_load_string()</code>	Converts an XML string into a SimpleXMLElement object
<code>getName()</code>	Returns the name of the XML tag referenced by the SimpleXML element
<code>getNamespaces()</code>	Returns the namespaces USED in document
<code>addAttribute()</code>	Adds an attribute to the SimpleXML element
<code>addChild()</code>	Adds a child element the SimpleXML element
<code>count()</code>	Counts the children of a specified node
<code>asXML()</code>	Returns a well-formed XML string from a SimpleXML object

PHP SimpleXML Parser

❖ Simplexml_load_string

```
$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);
```

❖ Simplexml_load_file

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


PHP SimpleXML Parser

❖ getName()

```
$xml=<<<XML
<?xml version="1.0" standalone="yes"?>
<cars>
  <car id="1">Volvo</car>
  <car id="2">BMW</car>
  <car id="3">Saab</car>
</cars>
XML;
```

```
$sxe=new SimpleXMLElement($xml);
echo $sxe->getName() . "<br>";
foreach ($sxe->children() as $child)
{
  echo $child->getName() . "<br>";
}
```

```
$ns=$sxe->getNamespaces(true);
var_dump($ns);
```

PHP SimpleXML Parser

❖ addChild() vs addAttribute()

```
$note=<<<XML  
<note>  
<to>Tove</to>  
<from>Jani</from>  
<heading>Reminder</heading>  
<body>Don't forget me this weekend!</body>  
</note>  
XML;
```

```
$xml = new SimpleXMLElement($note);  
$xml->addAttribute("type", "private");  
$xml->body->addAttribute("date", "2014-01-01");
```

```
echo $xml->asXML();
```

```
// Add a child element to the body element  
$xml->body->addChild("date", "2014-01-01");  
// Add a child element after the last element inside note  
$footer = $xml->addChild("footer", "Some footer text");
```

```
echo $xml->asXML();
```

PHP SimpleXML Parser

❖ count()

The count() function counts the children of a specified node

```
$xml=<<<XML
```

```
<cars>
```

```
<car name="Volvo">
```

```
<child/>
```

```
<child/>
```

```
<child/>
```

```
<child/>
```

```
</car>
```

```
<car name="BMW">
```

```
<child/>
```

```
<child/>
```

```
</car>
```

```
</cars>
```

```
XML;
```

```
$elem=new SimpleXMLElement($xml);
```

```
foreach ($elem as $car)
```

```
{
```

```
    printf("%s has %d children.<br>", $car['name'], $car->count());
```

```
};
```

JSON

- ❖ JSON (JavaScript Object Notation) is a lightweight data-interchange format.
- ❖ It is easy for humans to read and write. It is easy for machines to parse and generate.
- ❖ JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, properties make JSON an ideal data-interchange language.

```
{ "employees": [  
  { "firstName": "John", "lastName": "Doe" },  
  { "firstName": "Anna", "lastName": "Smith" },  
  { "firstName": "Peter", "lastName": "Jones" }  
]}
```

JSON Functions

json_encode	Returns the JSON representation of a value
json_decode	Decodes a JSON string
json_last_error	Returns the last error occurred

```
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
```

```
echo json_encode($arr);
```

```
//output
```

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

```
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
```

```
var_dump(json_decode($json));
```

```
var_dump(json_decode($json, true));
```

```
//output
```

```
object(stdClass)#1 (5) {
```

```
  ["a"] => int(1)
```

```
  ["b"] => int(2)
```

```
  ["c"] => int(3)
```

```
  ["d"] => int(4)
```

```
  ["e"] => int(5)
```

```
}
```

```
array(5) {
```

```
  ["a"] => int(1)
```

```
  ["b"] => int(2)
```

```
  ["c"] => int(3)
```

```
  ["d"] => int(4)
```

```
  ["e"] => int(5)
```

```
}
```

REGULAR EXPRESSIONS

- ❖ Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.
- ❖ Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks.
- ❖ Very useful for text processing applications – many web applications fit this pattern.
- ❖ For example, an email address follows a regular pattern.
- ❖ To check if an email address entered by a user is valid, we can check it against a regular expression that describes the pattern followed by email addresses.

REGULAR EXPRESSIONS

Expression	Description
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.

REGULAR EXPRESSIONS

Expression	Description
p^+	It matches any string containing at least one p .
p^*	It matches any string containing zero or more p 's.
$p?$	It matches any string containing zero or more p 's. This is just an alternative way to use p^* .
$p\{N\}$	It matches any string containing a sequence of N p 's
$p\{2,3\}$	It matches any string containing a sequence of two or three p 's.
$p\{2, \}$	It matches any string containing a sequence of at least two p 's.
$p\$$	It matches any string with p at the end of it.
p	It matches any string with p at the beginning of it.

REGULAR EXPRESSIONS

Expression	Description
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^. {2}\$	It matches any string containing exactly two characters.
(.*)	It matches any string enclosed within and .
p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence hp.

REGULAR EXPRESSIONS

Expression	Description
<code>[[:alpha:]]</code>	It matches any string containing alphabetic characters aA through zZ.
<code>[[:digit:]]</code>	It matches any string containing numerical digits 0 through 9.
<code>[[:alnum:]]</code>	It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
<code>[[:space:]]</code>	It matches any string containing a space.
<code>[[:upper:]]</code>	It matches an uppercase character
<code>[[:punct:]]</code>	It matches a punctuation character

REGULAR EXPRESSIONS

Character	Description
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo bar baz)	matches any of the alternatives specified

REGULAR EXPRESSIONS

You can give choices using the pipe character | (read it as OR)

For example:

(com) | (edu) | (org) | (net)

matches com or edu or org or net

If you want to match a character that has some special meaning

(. * + [] \$ ^ ()

you need to say "I mean a literal one of these, not the special meaning".

Put a backslash in front of it: \

If you want to match a backslash, you need two backslashes: \\

REGULAR EXPRESSIONS

Function	Description
<code>preg_match()</code>	The <code>preg_match()</code> function searches string for pattern, returning true if pattern exists, and false otherwise.
<code>preg_match_all()</code>	The <code>preg_match_all()</code> function matches all occurrences of pattern in string.
<code>preg_replace()</code>	The <code>preg_replace()</code> function operates just like <code>ereg_replace()</code> , except that regular expressions can be used in the pattern and replacement input parameters.
<code>preg_split()</code>	The <code>preg_split()</code> function operates exactly like <code>split()</code> , except that regular expressions are accepted as input parameters for pattern.
<code>preg_grep()</code>	The <code>preg_grep()</code> function searches all elements of <code>input_array</code> , returning all elements matching the <code>regex</code> pattern.

Error Handling

die()

```
<?php
if(!file_exists("/tmp/textNoExist.txt"))
{
    die("File not found");
}
else
{
    $file=fopen("/tmp/test.txt","r");
    print "Opend file sucessfully";
}
// Test of the code here.
?>
```

try...catch

```
<?php
try {
    $error = 'Always throw this error';
    throw new Exception($error);

    // Code following an exception is not executed.
    echo 'Never executed';

} catch (Exception $e) {
    echo 'Caught exception:', $e->getMessage(), "\n";
}

// Continue execution
echo 'Hello World';
?>
```

Function

```
<?php
```

```
/* Defining a PHP Function */
```

```
function writeMessage()
```

```
{
```

```
    echo "You are really a nice person, Have a nice time!";
```

```
}
```

```
/* Calling a PHP Function */
```

```
writeMessage();
```

```
?>
```

Maintaining State

- ❖ HTTP is a Stateless Protocol

HTTP has no built-in way of carrying state from page to page. This means that when a user clicks a link in page 1 and moves on to page 2, we have no way of knowing if it is the same user

- ❖ We would like to be able to track a user through a single session when using a web site. This can enable us to:

- > Log a user in and out
- > Track their behavior
- > Build useful and secure applications more easily

- ❖ 3 different ways:

- > Passing Hidden POST Variables
- > Cookies
- > Sessions (built into PHP language)

Passing Hidden POST Variables

You may have used this approach previously. Replace each link with a form button with action pointing to the next page and use a hidden variable.

```
<input type="hidden" name="username" value="joe" />
```

```
<?php
```

```
    $number_of_visits++;
```

```
?>
```

```
<input type="hidden" name="number_of_visits"  
    value="<?php echo number_of_visits?>" />
```

Cookies

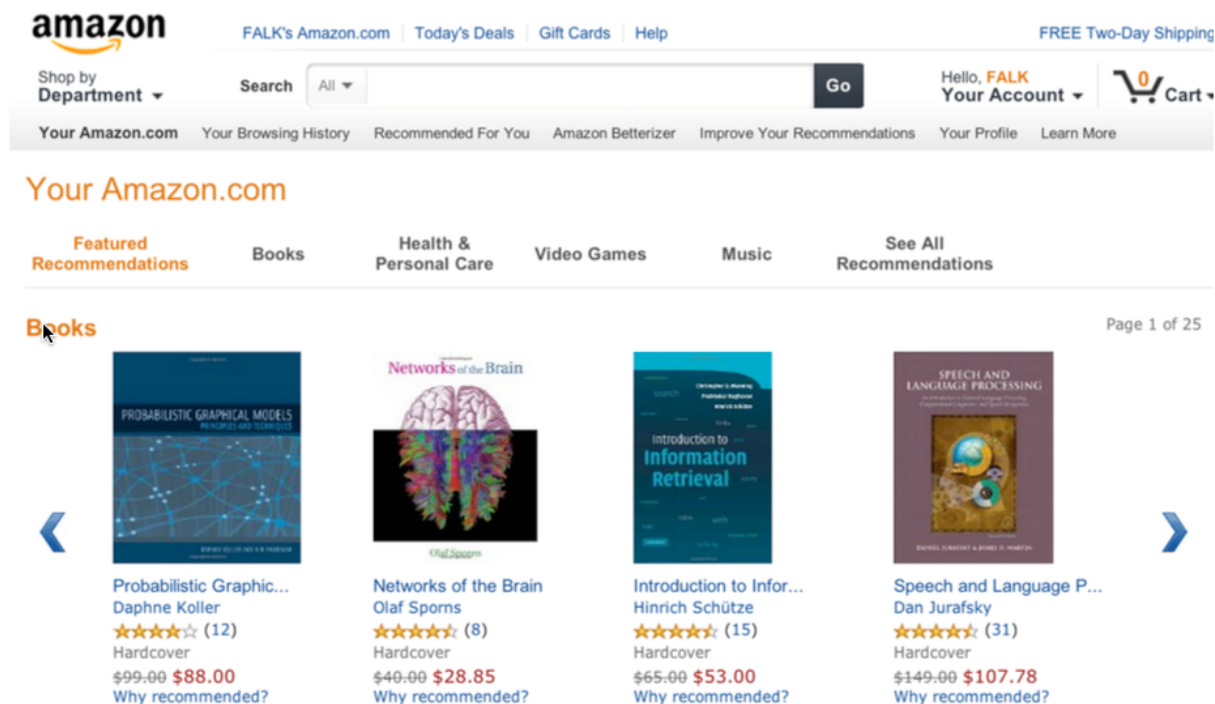
Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users:

- ❖ Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- ❖ Browser stores this information on local machine for future use.
- ❖ When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

Cookies

- ❖ When you visit certain websites (e.g. amazon.com), they can look at a unique identifier that was previously stored in a cookie on your machine.
- ❖ The website can then recognize you without you needing to re-enter a username.
- ❖ Sometimes this is convenient.



The screenshot shows the Amazon homepage. At the top, the Amazon logo is on the left, and navigation links for 'FALK's Amazon.com', 'Today's Deals', 'Gift Cards', and 'Help' are in the center. On the right, there's a link for 'FREE Two-Day Shipping'. Below the logo, there's a search bar with 'Shop by Department' on the left, 'Search' in the middle, and a 'Go' button on the right. To the right of the search bar, there's a greeting 'Hello, FALK' and a link to 'Your Account', and a shopping cart icon with '0' items. Below the search bar, there's a navigation bar with links: 'Your Amazon.com', 'Your Browsing History', 'Recommended For You', 'Amazon Betterizer', 'Improve Your Recommendations', 'Your Profile', and 'Learn More'. The main content area is titled 'Your Amazon.com' and features a 'Featured Recommendations' section. This section has tabs for 'Books', 'Health & Personal Care', 'Video Games', 'Music', and 'See All Recommendations'. The 'Books' tab is selected, showing a carousel of book recommendations. The books displayed are: 'Probabilistic Graphical Models: Principles and Techniques' by Daphne Koller, 'Networks of the Brain' by Olaf Sporns, 'Introduction to Information Retrieval' by Hinrich Schütze, and 'Speech and Language Processing' by Dan Jurafsky. Each book listing includes the title, author, star rating, number of reviews, cover type, price, and a 'Why recommended?' link. Navigation arrows are visible on the left and right sides of the book carousel.

amazon

FALK's Amazon.com | Today's Deals | Gift Cards | Help

FREE Two-Day Shipping

Shop by Department ▼ Search All ▼ Go

Hello, FALK Your Account ▼ Cart 0

Your Amazon.com Your Browsing History Recommended For You Amazon Betterizer Improve Your Recommendations Your Profile Learn More

Your Amazon.com

Featured Recommendations

Books Health & Personal Care Video Games Music See All Recommendations

Books

Page 1 of 25

Probabilistic Graphical Models: Principles and Techniques

Daphne Koller

★★★★☆ (12)

Hardcover

\$99.00 **\$88.00**

Why recommended?

Networks of the Brain

Olaf Sporns

★★★★☆ (8)

Hardcover

\$40.00 **\$28.85**

Why recommended?

Introduction to Information Retrieval

Hinrich Schütze

★★★★☆ (15)

Hardcover

\$65.00 **\$53.00**

Why recommended?

Speech and Language Processing

Dan Jurafsky

★★★★☆ (31)

Hardcover

\$149.00 **\$107.78**

Why recommended?

Cookies

There are a number of fields in a cookie:

- ❖ **Name**: what the data is called (a variable name)
- ❖ **Expires**: the date when cookie is no longer relevant
- ❖ **Domain**: the server that generated the cookie
- ❖ **Path**: used to specify a subset of URLs on the site where the cookie is relevant
- ❖ **Secure**: if this is set, the cookie will only be transmitted over HTTPS (encrypted HTTP - when you see the little padlock in the corner of your browser)

Cookies

Setting Cookies

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

`setcookie("name", "", time()- 60, "/", "", 0);`

Access Cookies

`echo $_COOKIE["name"]. "
";`

Delete Cookies

`setcookie("name", "", time()- 60, "/", "", 0);`

Cookies

Uses of Cookies

- ❖ Cookies are a flexible way of maintaining state or other information for a user.
- ❖ We can use them to store data on a user's machine for a period of time.
- ❖ A common use of cookies is with banner ads - the ad will set a cookie so the ad server knows which ads you have already seen.

Limitations of Cookies

- ❖ Could be switched off.
- ❖ Cookie data is stored on the client.

Sessions

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

When a session is started following things happen:

- ❖ PHP first creates a unique identifier for that particular session
- ❖ A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- ❖ Alternatively PHP, can automatically add the session ID to the end of urls. (You will need to compile PHP with --enable-trans-sid for this to work.)
- ❖ A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier.

Sessions

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

- ❖ Session variables contain data for a particular user – perhaps the number of items in their shopping cart, the ids of those items, or the total amount.
- ❖ The variables are actually stored at the *server*, and are accessed via the *session id* from your cookie.
- ❖ This is more secure than storing the data directly in a cookie.

Sessions

You begin a session by calling the function

session_start();

This will either:

- ❖ Create a new session, **or**
- ❖ Resume the current session, if a session id can be found from a cookie or in the URL.
- ❖ Generally, in a site that uses sessions, you will call this function at the start of every page where you need access to session variables.
- ❖ This creates the unique session id and stores it in a cookie (or adds it to the URL).
- ❖ If you reload the page, **session_start()** will find the session id and load any associated session variables.

Sessions

Setting Sessions

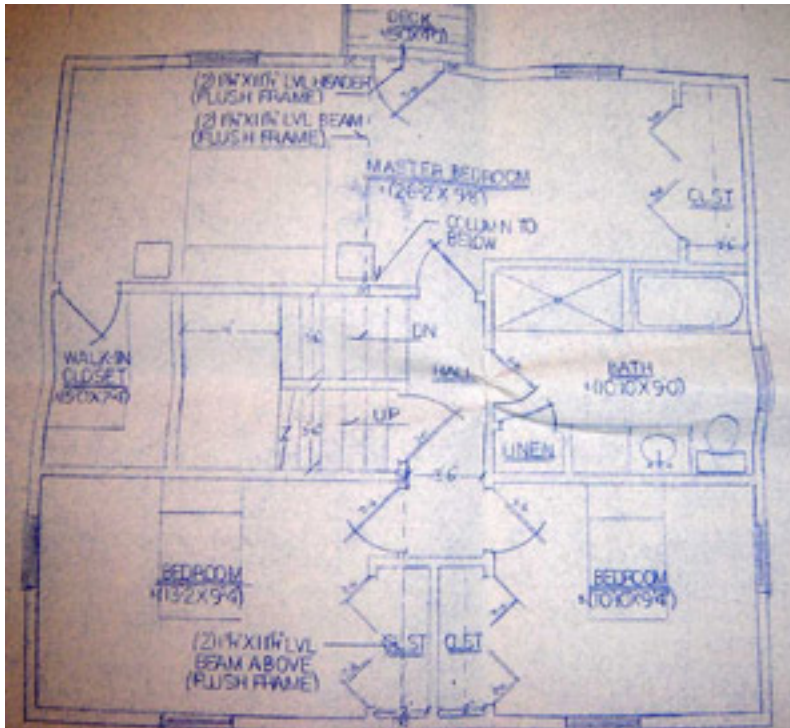
```
<?php
session_start();
if(isset($_SESSION['views'])) $_SESSION['views']=$_SESSION['views']+1;
else $_SESSION['views']=1;
echo "Views=". $_SESSION['views'];
?>
```

Destory Sessions

```
session_destroy();
//OR
unset($_SESSION['views']);
```

Object-Oriented

- ❖ Object-oriented programming is a style of coding that allows developers to group similar tasks into classes. This helps keep code following the tenet "don't repeat yourself" (DRY) and easy-to-maintain.



Class

Class is a programmer-defined datatype, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.

```
<?php
```

```
class MyClass
{
    // Class properties and methods go here
}
```

```
$obj = new MyClass;
```

```
var_dump($obj);
```

```
?>
```

Properties

Properties are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.

```
<?php
```

```
class MyClass
{
    public $prop1 = "I'm a class property!";
}
```

```
$obj = new MyClass;
```

```
echo $obj->prop1; // Output the property
```

```
?>
```

Methods

Methods are the function defined inside a class and are used to access object data.

```
<?php
class MyClass
{
    public $prop1 = "I'm a class property!";

    public function setProperty( $newval )
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

$obj = new MyClass;
echo $obj->getProperty(); // Get the property value

// Set a new one
$obj->setProperty( "I'm a new property value!" );
// Read it out again to show the change
echo $obj->getProperty();
?>
```

Constructor & Destructors

When an object is instantiated, it's often desirable to set a few things right off the bat. To handle this, PHP provides the magic method `__construct()`, which is called automatically whenever a new object is created.

To call a function when the object is destroyed, the `__destruct()` magic method is available. This is useful for class cleanup (closing a database connection, for instance).

```
<?php
class MyClass
{
    public $prop1 = "I'm a class property!";

    public function __construct()
    {
        echo 'The class ', __CLASS__, ' was initiated!<br />';
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

// Create a new object
$obj = new MyClass;
// Get the value of $prop1
echo $obj->getProperty();
// Output a message at the end of the file
echo "End of file.<br />";
?>
```

Inheritance

When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

```
class MyOtherClass extends MyClass
{
    public function newMethod()
    {
        echo "From a new method in " . __CLASS__ . "<br />";
    }
}
```


Overriding

Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.

```
class MyOtherClass extends MyClass
{
    public function __construct()
    {
        parent::__construct(); // Call the parent class's constructor
        echo "A new constructor in " . __CLASS__ . "<br />";
    }

    public function newMethod()
    {
        echo "From a new method in " . __CLASS__ . "<br />";
    }
}
```

Accessibility

Accessibility controls how and from where properties and methods can be accessed. There are three visibility keywords: public, protected, and private.

❖ Public

- > All the methods and properties you've used so far have been public. This means that they can be accessed anywhere, both within the class and externally.

❖ Protected

- > When a property or method is declared protected, **it can only be accessed within the class itself or in descendant classes** (classes that extend the class containing the protected method).

❖ Private

- > A property or method declared private is accessible **only from within the class that defines it**. This means that *even if a new class extends the class that defines a private property*, that property or method will not be available at all within the child class.

Accessibility

```
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Works
echo $obj->protected; // Fatal Error
echo $obj->private; // Fatal Error
$obj-
>printHello(); // Shows Public, Protected
and Private
```

```
class MyClass2 extends MyClass
{
    // We can redeclare the public and protected method, but not private
    protected $protected = 'Protected2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // Works
echo $obj2->protected; // Fatal Error
echo $obj2->private; // Undefined
$obj2-
>printHello(); // Shows Public, Protected2, Und
efined
```

Accessibility

```
class MyClass
{
    // Declare a public constructor
    public function __construct() { }

    // Declare a public method
    public function MyPublic() { }

    // Declare a protected method
    protected function MyProtected() { }

    // Declare a private method
    private function MyPrivate() { }

    // This is public
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Works
$myclass->MyProtected(); // Fatal Error
$myclass->MyPrivate(); // Fatal Error
$myclass->Foo();
// Public, Protected and Private work
```

```
class MyClass2 extends MyClass
{
    // This is public
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Fatal Error
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Works
$myclass2->Foo2();
// Public and Protected work,
// not Private
```

Accessibility

final keyword, which prevents child classes from overriding a method by prefixing the definition with *final*. If the class itself is being defined final then it cannot be extended.

```
<?php
class BaseClass {
    public function test() {
        echo "BaseClass::test() called\n";
    }

    final public function moreTesting() {
        echo "BaseClass::moreTesting() called\n";
    }
}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "ChildClass::moreTesting() called\n";
    }
}
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()
?>
```

Static

Declaring class members or methods as static makes them accessible without needing an instantiation of the class. A member declared as static can not be accessed with an instantiated class object (though a static method can).

```
public static function plusOne()  
{  
    return "The count is " .  
    ++self::$count . "<br />";  
}  
  
echo MyClass::plusOne();
```

```
class Foo  
{  
    public static $my_static = 'foo';  
  
    public function staticValue() {  
        return self::$my_static;  
    }  
}  
  
print Foo::$my_static . "\n";
```

Interface

Interfaces are defined to provide a common function names to the implementers.

Different implementers can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

```
interface Mail {  
    public function sendMail();  
}  
  
class Report implements Mail {  
    // sendMail() Definition goes here  
}
```

Abstract Class

An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword **abstract**, like this:

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same accessibility.

Abstract vs Interface

❖ Abstract Classes focus on a kind of things similarity.

It is the notion to extend from something, and optionally add some new feature or override some existing feature (to do differently). But using inheritance, you share a big part of code with the parent. **You are** a parent + some other things.

People are considered of type mammal and as such would not be considered of type vehicle.

❖ Interfaces focus on collation of similar function.


It is representing some abilities (we says a class is *implementing* an interface to says that it has these abilities). An interface can be implemented by 2 classes which are completely different and do not share their code (except for methods they implements).

For example: You are a human being and are of type mammal. If you want to fly then you will need to implement a flying Interface. If you want to shoot while flying, then you also need to implement the gun Interface.

MVC

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.



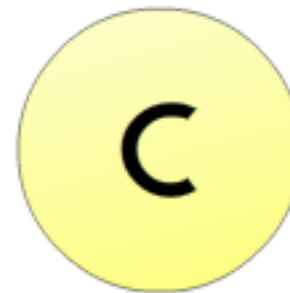
 model

database related,
not necessary
database. Data
can be XML or
even text files



 view

your website
design/ HTML
files. NO images,
css, etc.here.
only html layout



 controller

application logic,
process the user
request and get
appropriate data,
then output a design
from View

MVC

❖ **Model**

The Model is the name given to the permanent storage of the data used in the overall design. It must allow access for the data to be viewed, or collected and written to, and is the bridge between the View component and the Controller component in the overall pattern.

❖ **View**

The View is where data, requested from the Model, is viewed and its final output is determined. Traditionally in web apps built using MVC, the View is the part of the system where the HTML is generated and displayed.

❖ **Controller**

The final component of the triad is the Controller. Its job is to handle data that the user inputs or submits, and update the Model accordingly.

The Controller can be summed up simply as a collector of information, which then passes it on to the Model to be organized for storage, and does not contain any logic other than that needed to collect the input. The Controller is also only connected to a single View and to a single Model, making it a one way data flow system, with handshakes and signoffs at each point of data exchange.

MVC

