

UNIT 3

Advanced Server-Side Scripting

(15 Hrs) — Scripting Language (CACS254)

⚠ Important Note

These notes may contain errors, typos or omissions. Please double-check definitions, syntax and code examples against your textbook, class notes and the official TU syllabus before relying on them for exams. Test all code yourself before submitting it as practical work.

3.1 Object Oriented Programming in PHP

Object-Oriented Programming (OOP) organises code into **classes** (blueprints) and **objects** (instances). It improves reusability, maintainability and security of large applications.

Classes, Objects, Properties and Methods

Code:

```
<?php
class Student {
    public $name;           // properties
    public $marks;
    public function setData($n, $m) { // method
        $this->name = $n;
        $this->marks = $m;
    }
    public function show() {
        echo "Name: {$this->name}, Marks: {$this->marks}<br>";
    }
}
$s1 = new Student();           // creating an object
$s1->setData("Ram", 80);
$s1->show();
?>
```

Output:

```
Name: Ram, Marks: 80
```

- **\$this** refers to the current object; members are accessed with the arrow operator **->**.
- Class members are declared with visibility: **public** (everywhere), **protected** (class + child classes), **private** (only inside the class).

Constructors and Destructors

The **constructor** `__construct()` runs automatically when an object is created and is used to initialise properties. The **destructor** `__destruct()` runs when the object is destroyed or the script ends.

Code:

```
<?php
class Book {
    public $title;
    function __construct($t) {
        $this->title = $t;
        echo "Book created: $t<br>";
    }
    function __destruct() {
        echo "Book destroyed: {$this->title}<br>";
    }
}
$b = new Book("PHP Basics");
?>
```

Output:

```
Book created: PHP Basics
Book destroyed: PHP Basics
```

Encapsulation

Encapsulation means wrapping data and the methods that work on it into one unit and restricting direct access to the data. Properties are made **private** and are read or changed only through public **getter** and **setter** methods.

Code:

```
<?php
class Account {
    private $balance = 0;
    public function deposit($amt) {
        if ($amt > 0) { $this->balance += $amt; }
    }
    public function getBalance() { return $this->balance; }
}
$a = new Account();
$a->deposit(500);
echo $a->getBalance(); // 500
// echo $a->balance; // Fatal error: cannot access private property
?>
```

Inheritance and Method Overriding

Inheritance lets a child class reuse the properties and methods of a parent class using the keyword `extends`. **Method overriding** means the child class redefines a method that already exists in the parent; the parent version can still be called with `parent::method()`.

Code:

```
<?php
class Person {
    protected $name;
    function __construct($n) { $this->name = $n; }
    function intro() { echo "I am {$this->name}<br>"; }
}
class Teacher extends Person {
    private $subject;
    function __construct($n, $s) {
        parent::__construct($n);          // call parent constructor
        $this->subject = $s;
    }
    function intro() {                    // overriding
        parent::intro();
        echo "I teach {$this->subject}<br>";
    }
}
$t = new Teacher("Hari", "PHP");
$t->intro();
?>
```

Output:

```
I am Hari
I teach PHP
```

*Note: PHP supports only **single inheritance** of classes; multiple behaviours can be added through **interfaces** and **traits**. Use the `final` keyword to prevent a class from being extended or a method from being overridden.*

Polymorphism

Polymorphism ('many forms') means the same method call behaves differently depending on the object. In PHP it is achieved through method overriding with **abstract classes** or **interfaces**.

Code:

```
<?php
abstract class Shape {
    abstract function area();          // must be implemented by children
}
class Circle extends Shape {
    function __construct(private $r) {}
    function area() { return 3.14159 * $this->r * $this->r; }
}
```

```

}
class Rectangle extends Shape {
    function __construct(private $l, private $b) {}
    function area() { return $this->l * $this->b; }
}
$shapes = [new Circle(2), new Rectangle(3, 4)];
foreach ($shapes as $s) { echo $s->area() . "<br>"; } // 12.56636, 12
?>

```

Static Members

Static properties and methods belong to the class itself, not to any object. They are accessed with **ClassName::member** (or **self::** inside the class), and no object is required.

Code:

```

<?php
class Counter {
    public static $count = 0;
    public static function increase() { self::$count++; }
}
Counter::increase();
Counter::increase();
echo Counter::$count; // 2
?>

```

Exception Handling

An **exception** is an error condition that interrupts normal flow. PHP handles it with **try**, **catch**, **finally** and **throw**. This separates error-handling code from normal code.

Code:

```

<?php
function divide($a, $b) {
    if ($b == 0) { throw new Exception("Division by zero"); }
    return $a / $b;
}
try {
    echo divide(10, 2) . "<br>";
    echo divide(5, 0); // throws
} catch (Exception $e) {
    echo "Error: " . $e->getMessage(); // also getCode(), getLine(), getFile()
} finally {
    echo "<br>Always executed";
}
?>

```

Output:

```

5
Error: Division by zero

```

Custom exceptions are created by extending the built-in class: `class MyException extends Exception {}`.

OOP concept	One-line meaning
Class / Object	Blueprint / instance of the blueprint
Encapsulation	Hide data using private members and getters/setters
Inheritance	Child class reuses parent (extends)
Polymorphism	Same method, different behaviour
Abstraction	Show essential features only (abstract class / interface)
Static	Belongs to class, accessed by ::

3.2 AJAX (Asynchronous JavaScript and XML)

AJAX is a technique that lets a web page exchange data with the server **in the background** and update part of the page **without reloading** the entire page. It is not a language but a combination of: HTML/CSS, the DOM, JavaScript, the **XMLHttpRequest** object, and a server-side script (PHP) using XML/JSON/text.

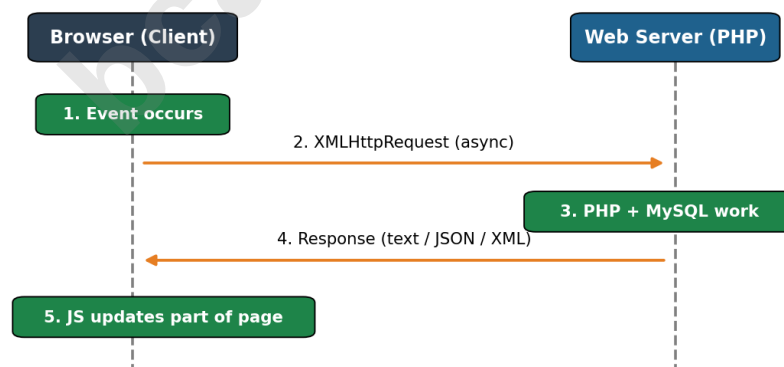


Figure 3.1: AJAX request/response cycle

- **Advantages:** faster, better user experience, less bandwidth, no full page reload.
- **Disadvantages:** back button and bookmarks may not work, needs JavaScript enabled, harder to debug, search engines may not index dynamic content.

XMLHttpRequest member	Description
open(method, url, async)	Prepares the request (GET/POST)

XMLHttpRequest member	Description
send() / send(data)	Sends the request (data only for POST)
onreadystatechange	Function called whenever readyState changes
readyState	0 unsent, 1 opened, 2 headers received, 3 loading, 4 done
status	HTTP status code (200 = OK, 404 = Not Found)
responseText / responseXML	Server response as text / XML

AJAX using PHP

Client side (HTML + JavaScript):

```
<!-- index.html: suggest a name while typing -->
<input type="text" id="q" onkeyup="hint(this.value)">
<p>Suggestions: <span id="out"></span></p>
<script>
function hint(str) {
    if (str.length == 0) {
        document.getElementById("out").innerHTML = "";
        return;
    }
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("out").innerHTML = this.responseText;
        }
    };
    xhr.open("GET", "gethint.php?q=" + encodeURIComponent(str), true);
    xhr.send();
}
</script>
```

Server side (PHP):

```
<?php
// gethint.php
$names = ["Anil", "Anita", "Binod", "Bishal", "Ram", "Rina", "Sita"];
$q = strtolower($_GET["q"]);
$hint = "";
foreach ($names as $n) {
    if (strpos(strtolower($n), $q) === 0) { // starts with typed text
        $hint .= ($hint == "" ? "" : ", ") . $n;
    }
}
echo $hint == "" ? "no suggestion" : $hint;
?>
```

AJAX using PHP + MySQL

Here the PHP file queries the database and returns the result. Example: choose a student and show the details without reloading.

Client side:

```
<select onchange="getStudent(this.value)">
  <option value="">Select</option>
  <option value="1">Ram</option><option value="2">Sita</option>
</select>
<div id="detail"></div>
<script>
function getStudent(id) {
  var xhr = new XMLHttpRequest();
  xhr.onload = function () {
    document.getElementById("detail").innerHTML = this.responseText;
  };
  xhr.open("POST", "getstudent.php");
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhr.send("id=" + id);
}
</script>
```

Server side:

```
<?php
// getstudent.php
$conn = new mysqli("localhost", "root", "", "school");
$id = (int) $_POST["id"]; // cast to avoid injection
$stmt = $conn->prepare("SELECT name, age, city FROM student WHERE id = ?");
$stmt->bind_param("i", $id);
$stmt->execute();
$row = $stmt->get_result()->fetch_assoc();
if ($row) {
  echo "<table border='1'><tr><td>Name</td><td>{$row['name']}</td></tr>"
    . "<tr><td>Age</td><td>{$row['age']}</td></tr>"
    . "<tr><td>City</td><td>{$row['city']}</td></tr></table>";
} else { echo "No record found"; }
$conn->close();
?>
```

3.3 jQuery

jQuery is a fast, small, cross-browser JavaScript library that simplifies DOM manipulation, event handling, animation and AJAX. Its motto is '*write less, do more*'. It is included from a CDN or a local file.

Including jQuery:

```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
<script>
$(document).ready(function () { // run when the DOM is ready
```

```
// code here
});
</script>
```

The basic syntax is `$(selector).action()`: the `$` defines jQuery, the **selector** finds HTML elements (same as CSS selectors) and the **action** is performed on them.

Playing with Elements

Category	Examples
Selectors	<code>\$('p')</code> , <code>\$('#id')</code> , <code>\$('.class')</code> , <code>\$(ul li)</code> , <code>\$(input[type=text])</code>
Get/Set content	<code>.text()</code> , <code>.html()</code> , <code>.val()</code> , <code>.attr('src')</code> , <code>.css('color','red')</code>
Add content	<code>.append()</code> , <code>.prepend()</code> , <code>.after()</code> , <code>.before()</code>
Remove	<code>.remove()</code> , <code>.empty()</code>
Classes	<code>.addClass()</code> , <code>.removeClass()</code> , <code>.toggleClass()</code>
Effects	<code>.hide()</code> , <code>.show()</code> , <code>.toggle()</code> , <code>.fadeIn()</code> , <code>.fadeOut()</code> , <code>.slideUp()</code> , <code>.slideDown()</code> , <code>.animate()</code>

Code:

```
$("#btn").click(function () {
    $("p").css("color", "blue");
    $("#box").html("<b>Changed</b>");
    $("ul").append("<li>New item</li>");
    $(".note").addClass("highlight");
});
$("input").on("keyup", function () { console.log($(this).val()); });
```

Hiding and Unhiding Images

Example: Hide, show and toggle an image using buttons

Three buttons control one image with jQuery effects.

Code:

```

<button id="hideBtn">Hide</button>
<button id="showBtn">Show</button>
<button id="toggleBtn">Toggle</button>
<button id="fadeBtn">Fade</button>
<script>
$(document).ready(function () {
    $("#hideBtn").click(function () { $("#pic").hide(1000); }); // 1 second
    $("#showBtn").click(function () { $("#pic").show(1000); });
    $("#toggleBtn").click(function () { $("#pic").toggle(); });
    $("#fadeBtn").click(function () { $("#pic").fadeToggle("slow"); });
});
```

```
});
</script>
```

Explanation

Each button has an id; `$('#hideBtn').click(...)` attaches a click handler.

`hide()` / `show()` change visibility (optional speed in milliseconds), `toggle()` flips the current state, and `fadeToggle()` does it with a fading effect.

jQuery UI

jQuery UI is a curated set of user-interface widgets, effects and interactions built on top of jQuery. It needs its own CSS and JS files in addition to jQuery.

Category	Components
Widgets	Datepicker, Accordion, Tabs, Dialog, Autocomplete, Slider, Progressbar, Menu, Button
Interactions	Draggable, Droppable, Resizable, Selectable, Sortable
Effects	Explode, bounce, shake, highlight, blind, fold

Code:

```
<link rel="stylesheet"
  href="https://code.jquery.com/ui/1.13.2/themes/base/jquery-ui.css">
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
<script src="https://code.jquery.com/ui/1.13.2/jquery-ui.min.js"></script>
<input type="text" id="dob">
<div id="tabs">
  <ul><li><a href="#t1">One</a></li><li><a href="#t2">Two</a></li></ul>
  <div id="t1">First</div><div id="t2">Second</div>
</div>
<div id="box" style="width:100px;height:100px;background:orange"></div>
<script>
$(function () {
  $("#dob").datepicker({ dateFormat: "yy-mm-dd" }); // calendar pop-up
  $("#tabs").tabs(); // tabbed panel
  $("#box").draggable(); // drag with mouse
});
</script>
```

3.4 Joomla (CMS)

Introduction to CMS

A **Content Management System (CMS)** is software that lets users create, manage and publish website content without writing code from scratch. Content is stored in a database and displayed using templates. Popular CMSs: **Joomla, WordPress, Drupal**.

Joomla is a free, open-source CMS written in **PHP** using a **MySQL** database and following the **MVC** pattern. It is used for corporate sites, e-commerce, community portals and blogs.

Installation

- Install a local server (XAMPP) and start Apache and MySQL.
- Download the Joomla package and extract it in `htdocs/joomla`.
- Create an empty database (e.g. `joomla_db`) in phpMyAdmin.
- Open `http://localhost/joomla` and complete the web installer: site name, admin username/password/email, then database type (MySQLi), host (localhost), DB user, password and DB name.
- Finish and remove/rename the installation folder; log in at `http://localhost/joomla/administrator`.

Handling the Joomla Back End and Customization

The **back end (Administrator)** is the control panel used to manage the site. Main menus are:

Menu	Purpose
System	Global configuration, users' settings, cache, updates, information
Users	Users, groups, access levels, permissions
Menus	Create menus and menu items (links)
Content	Articles, categories, featured articles, media
Components	Banners, contacts, news feeds, etc.
Extensions	Install and manage modules, plugins, templates, languages
Help	Documentation

Customization is done by changing the template style, adding menus and modules, editing articles and categories, changing the site logo/colours in template options, and installing extensions.

Extensions

Extensions add features to Joomla. There are five types:

Type	Description
Components	Main mini-applications shown in the content area (e.g. Contacts, Banners)
Modules	Small blocks displayed in template positions (e.g. Login form, Menu, Latest articles)
Plugins	Event-driven code that runs in the background (e.g. editors, authentication)
Templates	Control the look and layout of the site
Languages	Translations of the interface

Installing an extension: Administrator → System → Install → Extensions → upload the ZIP package (or install from URL) → the extension appears in its manager where it can be enabled and configured. Modules also need a *position* and *menu assignment*.

Template Development in Joomla

A template is a folder in `/templates/name` with these key files:

File	Purpose
templateDetails.xml	Manifest: template name, author, version, files and module positions
index.php	Main layout in HTML + PHP; includes module positions and the component output
css/template.css	Stylesheet
images/, js/	Images and scripts
template_preview.png	Preview image

Skeleton of index.php:

```
<?php
defined('_JEXEC') or die;                // security check
?>
<!DOCTYPE html>
<html>
<head><jdoc:include type="head" /></head>
<body>
  <header><jdoc:include type="modules" name="header" style="none" /></header>
  <nav><jdoc:include type="modules" name="menu" /></nav>
  <main><jdoc:include type="component" /></main>
  <footer><jdoc:include type="modules" name="footer" /></footer>
</body>
</html>
```

Artisteer (written 'Artiseer' in the syllabus) is a visual **design tool (IDE)** that generates Joomla templates automatically. The designer selects layout, colours, fonts and header images, and the tool exports the finished template package that is then installed like any other extension.

Module Development in Joomla

A module shows small pieces of output (e.g. a greeting or latest news). A simple module named `mod_hello` needs:

File	Purpose
<code>mod_hello.xml</code>	Manifest describing the module and its files/parameters
<code>mod_hello.php</code>	Entry point, gets data and loads the layout
<code>helper.php</code>	Optional class with data-fetching functions
<code>tmpl/default.php</code>	Layout (HTML output)

`mod_hello.php`:

```
<?php
// mod_hello.php
defined('_JEXEC') or die;
$greeting = "Hello from my first module!";
require JModuleHelper::getLayoutPath('mod_hello');
```

`tmpl/default.php`:

```
<?php
// tmpl/default.php
defined('_JEXEC') or die;
echo '<div class="hello">' . $greeting . '</div>';
```

Zip the folder, install it via the Extension Manager, then create a module instance (Content → Site Modules), pick a template position and assign it to menu items.

Component Development in Joomla

A **component** is the main application of a page. It has a **site (front-end)** part and an **administrator (back-end)** part. A minimal component named `com_hello` has:

- `hello.xml` – manifest file (name, version, files, admin menu).
- `hello.php` – entry point that loads the controller.
- `controller.php` – controller class handling the request.
- `models/` – classes that access the database.
- `views/` with `tmpl/default.php` – output for the user.

Entry point of a legacy-style component:

```
<?php
// site/hello.php (entry point)
defined('_JEXEC') or die;
$controller = JControllerLegacy::getInstance('Hello');
$controller->execute(JFactory::getApplication()->input->get('task'));
$controller->redirect();
```

Note: Class names and APIs differ between Joomla versions (3.x uses names such as JFactory; 4.x/5.x use namespaces). Check the version taught in your class.

Introduction to MVC (Model, View, Controller)

MVC is a software design pattern that separates an application into three parts, which makes code organised, reusable and easy to maintain. Joomla components follow this pattern.

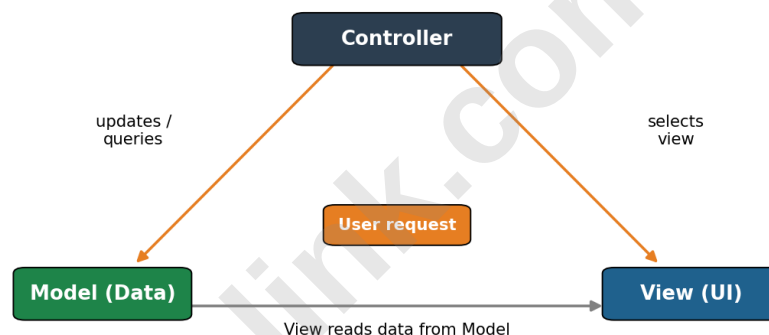


Figure 3.2: Model-View-Controller pattern

Part	Responsibility	Example in Joomla
Model	Handles data and business logic; talks to the database	models/hello.php
View	Presents data to the user (HTML output)	views/hello/tmpl/default.php
Controller	Receives user requests, calls the model and picks the view	controller.php

3.5 WordPress (Administrator Level)

WordPress is the world's most widely used open-source CMS, built with PHP and MySQL. It is ideal for blogs, business websites and online stores. The admin dashboard is opened at yoursite.com/wp-admin.

Dashboard menu	Use
Posts	Write and manage blog posts, categories and tags
Media	Upload and manage images, videos and files
Pages	Static pages such as Home, About, Contact
Comments	Approve, reply to or delete comments
Appearance	Themes, Customize, Widgets, Menus, Theme editor
Plugins	Install, activate and manage plugins
Users	Add users and set roles (Admin, Editor, Author, Contributor, Subscriber)
Settings	General, Writing, Reading, Permalinks

Theme Integration

A **theme** controls the design of the website. To add one: *Appearance* → *Themes* → *Add New*, search a free theme or *Upload Theme* (ZIP) → **Install** → **Activate**. Options can be changed with *Appearance* → *Customize*. A custom theme is a folder in `wp-content/themes/` containing at least `style.css` (with a theme header comment) and `index.php`.

style.css header:

```
/*
Theme Name: My First Theme
Author: Student
Version: 1.0
*/
```

Basic index.php (The Loop):

```
<?php get_header(); ?>
<?php if (have_posts()) : while (have_posts()) : the_post(); ?>
    <h2><?php the_title(); ?></h2>
    <?php the_content(); ?>
<?php endwhile; endif; ?>
<?php get_footer(); ?>
```

Creating Pages and Managing Posts

- **Create a page:** Pages → Add New → enter title and content in the block editor → choose a template/parent if needed → **Publish**.
- **Add a post:** Posts → Add New → write content, add a featured image, choose **category** and **tags** → Publish (or Save Draft / Schedule).

- **Manage posts:** edit, quick edit, trash, restore, change status (published, draft, pending, private) and use bulk actions.
- **Pages vs posts:** posts are time-based and appear in the blog feed with categories/tags; pages are static and hierarchical.

Managing Widgets

Widgets are small content blocks (search box, recent posts, categories, calendar, text, custom HTML) placed in **widget areas** such as the sidebar and footer defined by the theme. Go to *Appearance* → *Widgets* (or *Customize* → *Widgets*), pick a widget area, click **+** to add a block/widget, configure it, drag to re-order, and press **Update**.

Key Takeaways

- ✓ OOP in PHP: class, object, constructor/destructor, encapsulation (private + getters/setters), inheritance (extends), polymorphism, static (::), and try-catch-finally exceptions.
- ✓ AJAX updates part of a page using XMLHttpRequest without reloading; PHP (and MySQL) work on the server, and the response is placed in the DOM.
- ✓ jQuery: `$(selector).action()`; `hide()`, `show()`, `toggle()` and fade effects manage elements; jQuery UI adds datepicker, tabs, draggable, etc.
- ✓ Joomla is a PHP/MySQL CMS with back-end administration, five types of extensions, templates (index.php + XML), modules and MVC-based components.
- ✓ MVC separates Model (data), View (output) and Controller (request handling).
- ✓ WordPress admin tasks: install/activate themes, create pages, manage posts, and place widgets.

— *End of Unit 3* —