

UNIT 2

Server-Side Scripting with Database Connectivity

(15 Hrs: PHP 8 Hrs + MySQL 7 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.

PART A: PHP

2.1 Introduction to Server-Side Scripting and PHP

Server-side scripting means the script is executed on the web server; the browser receives only the resulting HTML. **PHP (PHP: Hypertext Preprocessor)** is an open-source, general-purpose, server-side scripting language created by Rasmus Lerdorf in 1994. PHP files have the extension `.php` and can mix HTML and PHP code.

- Free, open source and cross-platform (Windows, Linux, macOS).
- Easy to learn, with built-in support for databases such as MySQL.
- Works with Apache/Nginx; typical stack: **XAMPP / WAMP / LAMP** (Apache + MySQL + PHP).
- PHP is **loosely typed**, interpreted and mostly case-insensitive for function names but **case-sensitive for variables**.

How a PHP page works

1. Browser requests **index.php**. 2. Web server passes the file to the PHP engine. 3. PHP runs the code (and queries the database if needed). 4. The generated HTML is sent back to the browser.

2.2 Basic PHP Syntax and Comments

Code:

```
<?php
    echo "Hello, World!";      // statement ends with a semicolon
    print "<br>Welcome";
?>
```

Output:

```
Hello, World!
Welcome
```

Comments in PHP:

```
<?php
// single-line comment
# also a single-line comment
/* multi-line
   comment */
?>
```

2.3 Variables, Data Types and Operators

- A variable starts with \$, followed by a letter or underscore, e.g. `$name`.
- No declaration keyword is needed; the type is decided by the assigned value.
- Data types: **string, integer, float, boolean, array, object, NULL**.
- Scope: **local, global, static**; use the `global` keyword to access a global variable in a function.
- Constants: `define("PI", 3.14);` or `const PI = 3.14;`

Code:

```
<?php
$name = "Ram"; $age = 20; $pi = 3.14; $ok = true;
echo "Name: $name, Age: $age";           // double quotes parse variables
echo 'Name: $name';                     // single quotes do NOT
echo "Type: " . gettype($age);          // . is concatenation
var_dump($pi);                          // float(3.14)
?>
```

Type	Operators
Arithmetic	<code>+ - * / % **</code>
Assignment	<code>= += -= *= /= .= %=</code>
Comparison	<code>== === != !== < > <= >= <=></code> (spaceship)
Increment/Decrement	<code>++\$i \$i++ --\$i \$i--</code>
Logical	<code>&& ! and or xor</code>
String	<code>.</code> (concatenation) <code>.=</code> (append)
Null coalescing	<code>\$x ?? 'default'</code>

2.4 Control Structures

Code:

```
<?php
$n = 15;
if ($n % 2 == 0) { echo "Even"; }
elseif ($n > 10) { echo "Odd and greater than 10"; }
else { echo "Odd"; }
switch ($day) {
    case 1: echo "Sunday"; break;
    case 7: echo "Saturday"; break;
    default: echo "Other day";
}
for ($i = 1; $i <= 5; $i++) { echo $i . " "; }
$j = 1; while ($j <= 5) { echo $j++ . " "; }
$k = 1; do { echo $k++ . " "; } while ($k <= 5);
?>
```

Output:

```
Odd and greater than 10
1 2 3 4 5
```

2.5 Arrays and the foreach Loop

PHP has three kinds of arrays: **indexed** (numeric keys), **associative** (named keys) and **multidimensional** (arrays inside arrays).

Code:

```
<?php
$colors = array("Red", "Green", "Blue");           // indexed
$marks  = ["Ram" => 80, "Sita" => 92];             // associative
$table  = [[1, 2], [3, 4]];                       // multidimensional
echo $colors[1];                                   // Green
echo count($colors);                               // 3
$colors[] = "Yellow";                             // append
foreach ($colors as $c) { echo $c . "<br>"; }
foreach ($marks as $name => $m) { echo "$name got $m<br>"; }
echo $table[1][0];                                // 3
?>
```

Function	Purpose
count()	Number of elements
sort() / rsort()	Sort ascending / descending (by value)
asort() / ksort()	Sort associative array by value / by key
array_push() / array_pop()	Add / remove at end

Function	Purpose
in_array()	Check whether a value exists
array_merge()	Join arrays
array_keys() / array_values()	Get all keys / values

2.6 Functions

Code:

```
<?php
function greet($name = "Guest") {           // default parameter
    return "Hello, " . $name;
}
echo greet("Sita");                          // Hello, Sita
echo greet();                               // Hello, Guest
function addRef(&$x) { $x += 10; }           // pass by reference
$a = 5; addRef($a); echo $a;                // 15
function sum(int $a, int $b): int { return $a + $b; } // typed
?>
```

Useful built-in functions: `strlen()`, `strtoupper()`, `str_replace()`, `substr()`, `strpos()`, `trim()`, `round()`, `rand()`, `isset()`, `empty()`.

2.7 Form Handling: \$_GET, \$_POST, \$_REQUEST

PHP collects form data using **superglobal** arrays. The form's `method` attribute decides which one is used.

Feature	GET	POST
Data location	Appended to URL (visible)	Sent in HTTP body (hidden)
Size limit	Limited (about 2000 chars)	Practically unlimited
Security	Not for passwords/sensitive data	Better; still needs HTTPS
Bookmarking	Possible	Not possible
Superglobal	<code>\$_GET</code>	<code>\$_POST</code>

`$_REQUEST` contains the data of both GET, POST and cookies.

form.html:

```
<!-- form.html -->
<form action="welcome.php" method="post">
    Name: <input type="text" name="uname">
    Email: <input type="text" name="email">
```

```
<input type="submit">
</form>
```

welcome.php:

```
<?php
// welcome.php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = htmlspecialchars(trim($_POST["uname"])); // sanitize output
    $email = $_POST["email"];
    if (empty($name)) { echo "Name is required"; }
    elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) { echo "Bad email"; }
    else { echo "Welcome $name, your email is $email"; }
}
?>
```

Note: Always validate and sanitise user input on the server (htmlspecialchars(), filter_var()) to prevent XSS and injection attacks.

2.8 PHP date() Function and include Files

Code:

```
<?php
echo date("Y-m-d"); // 2026-04-14 (Y=year, m=month, d=day)
echo date("d/m/Y H:i:s"); // 14/04/2026 10:30:45
echo date("l, jS F Y"); // Tuesday, 14th April 2026
date_default_timezone_set("Asia/Kathmandu");
echo mktime(0, 0, 0, 4, 14, 2026); // Unix timestamp
?>
```

include and **require** insert the contents of another PHP file, which is useful for shared headers, footers and database connection code. If the file is missing, **include** gives a warning and continues, but **require** gives a fatal error and stops. **include_once** and **require_once** avoid inserting the same file twice.

Code:

```
<?php
include "header.php";
require_once "config.php";
echo "Main content";
include "footer.php";
?>
```

2.9 File Handling and File Uploading

Code:

```
<?php
$f = fopen("data.txt", "r"); // modes: r, w, a, r+, w+, a+
```

```

while (!feof($f)) { echo fgets($f) . "<br>"; } // read line by line
fclose($f);
$f = fopen("data.txt", "a"); // append
fwrite($f, "New line\n");
fclose($f);
echo file_get_contents("data.txt");
file_put_contents("out.txt", "Hello");
if (file_exists("data.txt")) { unlink("data.txt"); } // delete
?>

```

Mode	Meaning
r	Read only; pointer at start
w	Write only; erases file or creates a new one
a	Append; pointer at end, creates if missing
r+ / w+ / a+	Read and write variants

File Uploading

To upload a file, the form must use `method="post"` and `enctype="multipart/form-data"`. Uploaded file details are in the `$_FILES` array (name, type, tmp_name, size, error).

upload form:

```

<form action="upload.php" method="post" enctype="multipart/form-data">
  <input type="file" name="photo">
  <input type="submit" value="Upload">
</form>

```

upload.php:

```

<?php
// upload.php
$allowed = ["jpg", "jpeg", "png"];
$ext = strtolower(pathinfo($_FILES["photo"]["name"], PATHINFO_EXTENSION));
if (!in_array($ext, $allowed)) { die("Only JPG/PNG allowed"); }
if ($_FILES["photo"]["size"] > 2000000) { die("File too large (max 2MB)"); }
$target = "uploads/" . basename($_FILES["photo"]["name"]);
if (move_uploaded_file($_FILES["photo"]["tmp_name"], $target)) {
  echo "Uploaded successfully";
} else { echo "Upload failed"; }
?>

```

2.10 PHP Sessions and Cookies

A **cookie** is stored on the **client** computer, while a **session** stores data on the **server** and gives the browser only a session ID (PHPSESSID). Sessions are safer and are used for login systems and shopping carts.

Sessions:

```
<?php
// page1.php
session_start();                // must be before any output
$_SESSION["user"] = "Ram";
$_SESSION["role"] = "admin";
// page2.php
session_start();
echo "Hello " . $_SESSION["user"]; // Hello Ram
// logout.php
session_start();
session_unset();                // remove all session variables
session_destroy();              // destroy the session
?>
```

Cookies (call setcookie() before any output):

```
<?php
setcookie("theme", "dark", time() + 86400 * 30, "/"); // 30 days
if (isset($_COOKIE["theme"])) { echo $_COOKIE["theme"]; }
setcookie("theme", "", time() - 3600);                // delete cookie
?>
```

Cookie	Session
Stored on client browser	Stored on server
Less secure, user can edit	More secure
Size limited (about 4 KB)	Much larger data possible
Persists until expiry	Ends when browser closes or session is destroyed

2.11 Sending Emails

PHP sends mail with the `mail()` function. It needs a configured mail server (SMTP/sendmail); on a local XAMPP setup it is usually configured in `php.ini` / `sendmail.ini`. For real projects, libraries like PHPMailer are preferred.

Code:

```
<?php
$to      = "someone@example.com";
$subject = "Test mail";
```

```
$message = "Hello from PHP!";
$headers = "From: admin@mysite.com\r\nReply-To: admin@mysite.com";
if (mail($to, $subject, $message, $headers)) { echo "Mail sent"; }
else { echo "Mail failed"; }
?>
```

PART B: MySQL

2.12 Introduction to MySQL

MySQL is a popular open-source **relational database management system (RDBMS)** that stores data in tables made up of rows (records) and columns (fields). It uses **SQL (Structured Query Language)**. PHP talks to MySQL through the **MySQLi** extension (or PDO). The [phpMyAdmin](#) tool in XAMPP provides a graphical interface.

MySQL Data Types

Category	Types	Notes
Numeric	INT, TINYINT, BIGINT, FLOAT, DOUBLE, DECIMAL(m,d)	Whole and decimal numbers
String	CHAR(n), VARCHAR(n), TEXT, BLOB	CHAR is fixed length, VARCHAR variable
Date/Time	DATE, TIME, DATETIME, TIMESTAMP, YEAR	DATE format is YYYY-MM-DD
Other	ENUM('a','b'), SET, BOOLEAN	ENUM allows one value from a list

Create database and table:

```
CREATE DATABASE school;
USE school;
CREATE TABLE student (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INT,
    city VARCHAR(30),
    marks DECIMAL(5,2),
    dept_id INT
);
```

2.13 Connecting PHP to MySQL

Procedural MySQLi:

```
<?php
$conn = mysqli_connect("localhost", "root", "", "school");
```

```

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
// ... queries here ...
mysqli_close($conn);           // closing the connection
?>

```

Object-oriented MySQLi:

```

<?php
$conn = new mysqli("localhost", "root", "", "school");
if ($conn->connect_error) { die("Failed: " . $conn->connect_error); }
$conn->close();
?>

```

2.14 MySQL Insert, Select, Where, Update, Delete

SQL commands:

```

INSERT INTO student (name, age, city, marks, dept_id)
VALUES ('Ram', 20, 'Kathmandu', 78.5, 1);
SELECT * FROM student;
SELECT name, marks FROM student WHERE city = 'Pokhara' AND marks > 60;
SELECT * FROM student WHERE name LIKE 'R%';           -- names starting with R
SELECT * FROM student WHERE age BETWEEN 18 AND 22;
SELECT * FROM student WHERE city IN ('Kathmandu', 'Lalitpur');
UPDATE student SET marks = 85, city = 'Bhaktapur' WHERE id = 1;
DELETE FROM student WHERE id = 3;

```

Note: Always use a WHERE clause with UPDATE and DELETE; without it, all rows are changed or deleted.

PHP + MySQL CRUD example:

```

<?php
$conn = mysqli_connect("localhost", "root", "", "school");
// INSERT
$sql = "INSERT INTO student (name, age, city) VALUES ('Sita', 21, 'Pokhara')";
if (mysqli_query($conn, $sql)) { echo "Record added"; }
else { echo "Error: " . mysqli_error($conn); }
// SELECT and display in a table
$res = mysqli_query($conn, "SELECT * FROM student");
if (mysqli_num_rows($res) > 0) {
    echo "<table border='1'><tr><th>ID</th><th>Name</th><th>City</th></tr>";
    while ($row = mysqli_fetch_assoc($res)) {
        echo "<tr><td>{$row['id']}</td><td>{$row['name']}</td>"
            . "<td>{$row['city']}</td></tr>";
    }
    echo "</table>";
}
// UPDATE and DELETE
mysqli_query($conn, "UPDATE student SET city='Butwal' WHERE id=2");
mysqli_query($conn, "DELETE FROM student WHERE id=5");

```

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

Prepared statements protect against SQL injection and should be used when the query contains user input:

Code:

```
<?php  
$stmt = $conn->prepare("INSERT INTO student (name, age) VALUES (?, ?)");  
$stmt->bind_param("si", $name, $age); // s = string, i = integer  
$name = "Hari"; $age = 22;  
$stmt->execute();  
$stmt->close();  
?>
```

2.15 Aggregate Functions, ORDER BY and GROUP BY

Function	Returns
COUNT()	Number of rows
SUM()	Total of a numeric column
AVG()	Average value
MIN() / MAX()	Smallest / largest value

Code:

```
SELECT COUNT(*) FROM student;  
SELECT SUM(marks), AVG(marks), MAX(marks), MIN(marks) FROM student;  
SELECT * FROM student ORDER BY marks DESC;           -- highest first  
SELECT * FROM student ORDER BY city ASC, name ASC;  
SELECT city, COUNT(*) AS total, AVG(marks) AS avg_marks  
FROM student  
GROUP BY city  
HAVING COUNT(*) > 1                                -- filter groups  
ORDER BY total DESC;
```

Note: WHERE filters individual rows before grouping; HAVING filters groups after grouping.

2.16 MySQL Subqueries and Joins

A **subquery** is a query nested inside another query, enclosed in parentheses.

Code:

```
-- students scoring above the average  
SELECT name, marks FROM student  
WHERE marks > (SELECT AVG(marks) FROM student);
```

```
-- students living in cities where a department exists
SELECT name FROM student
WHERE dept_id IN (SELECT id FROM department WHERE name = 'BCA');
```

A **join** combines rows from two or more tables using a related column. Assume tables `student(id, name, dept_id)` and `department(id, dname)`.

Join	Result
INNER JOIN	Only rows that match in both tables
LEFT JOIN	All rows of left table + matching rows of right (else NULL)
RIGHT JOIN	All rows of right table + matching rows of left
CROSS JOIN	Cartesian product of both tables

Code:

```
SELECT s.name, d.dname
FROM student s
INNER JOIN department d ON s.dept_id = d.id;
SELECT s.name, d.dname
FROM student s
LEFT JOIN department d ON s.dept_id = d.id; -- keeps students with no dept
```

Example: Find the number of students in each department

Tables: `student(id, name, dept_id)`, `department(id, dname)`.

Solution

```
SELECT d.dname, COUNT(s.id) AS total FROM department d LEFT JOIN
student s ON s.dept_id = d.id GROUP BY d.dname;
```

Key Takeaways

- ✓ PHP runs on the server; variables start with \$, strings join with a dot, and code sits in `<?php ... ?>`.
- ✓ Use `$_GET` / `$_POST` / `$_REQUEST` for form data, and always validate and sanitise input.
- ✓ include/require reuse files; fopen/fread/fwrite handle files; `$_FILES` + `move_uploaded_file()` handle uploads.
- ✓ Sessions live on the server (`session_start`), cookies on the client (`setcookie`).
- ✓ MySQL steps in PHP: connect → query → fetch results → close. Prefer prepared statements.
- ✓ Know INSERT, SELECT, WHERE, UPDATE, DELETE, aggregate functions, ORDER BY, GROUP BY/HAVING, subqueries and joins.

— *End of Unit 2* —

bcalink.com