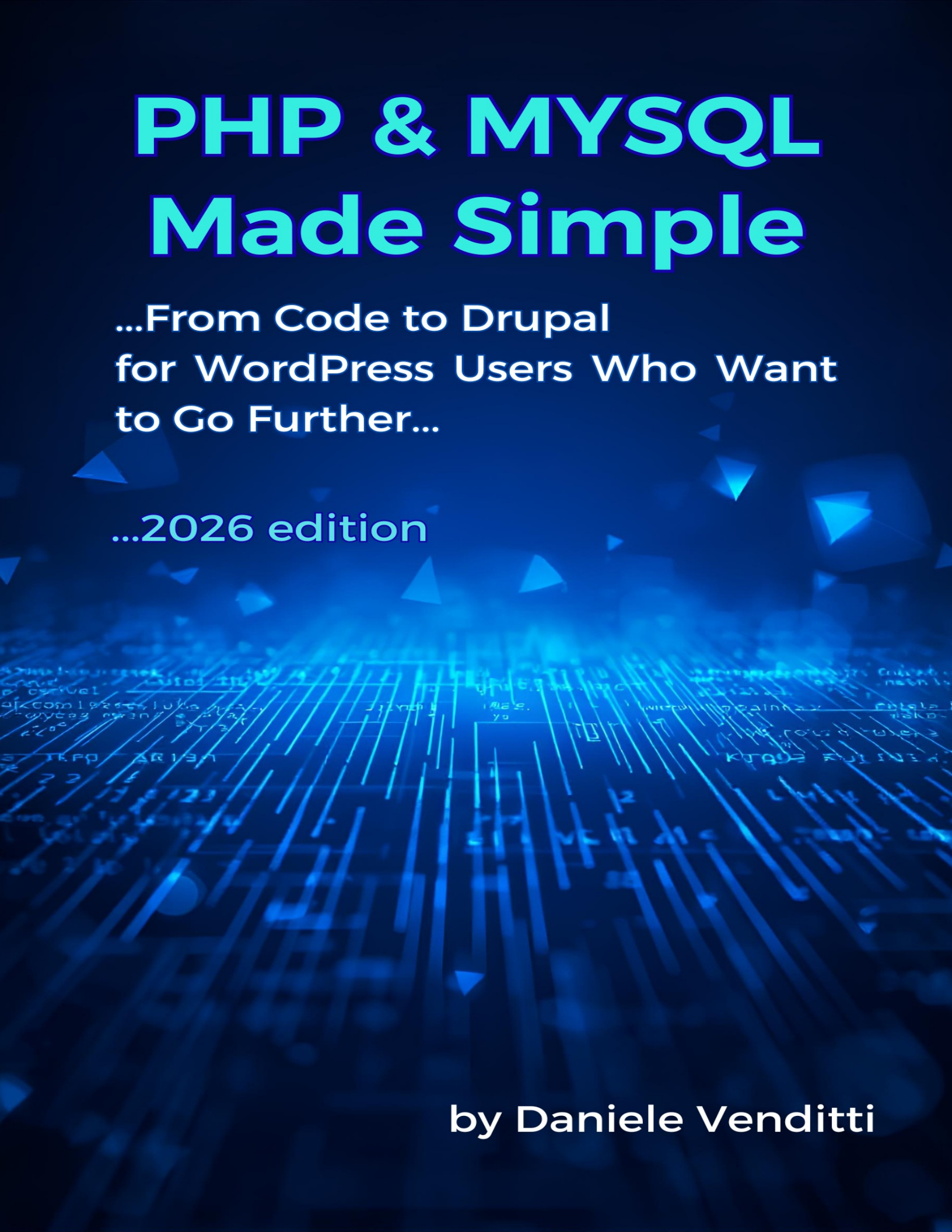
PHP & MYSQL Made Simple

...From Code to Drupal for WordPress Users Who Want to Go Further...

...2026 edition



by Daniele Venditti

PHP & MYSQL Made Simple: 100% Practical Course, From Code to Drupal, for WordPress Users Who Want to Go Further

Daniele Venditti

PHP & MYSQL Made Simple: 100% Practical Course

From Code to Drupal, for WordPress Users Who Want to Go Further

Daniele Venditti *progettimultimediali.com*

Introduction

You started with WordPress. Probably like everyone else.

One day you installed a theme, then a plugin, then another. The site worked, the client was happy, and you felt like a web professional. And you were, there's nothing wrong with that.

But at some point, something changed.

Maybe you noticed that the sites you build are starting to look the same. Maybe a plugin broke after an update and you spent hours figuring out why. Maybe you saw artificial intelligence arrive and thought: "*If anyone can create a website with a prompt, what's left of my work?*"

These questions don't come out of nowhere. They come up because you're a curious person who wants to genuinely understand how things work. And that's exactly the quality that brought you here.

What WordPress Never Showed You

WordPress is an excellent tool. I say that without irony. It's allowed millions of people to build websites without writing a single line of code, and that was revolutionary.

But there's a price to pay for all that convenience: **you've never seen what happens under the hood.**

Every time you write an article and click “Publish,” WordPress automatically runs dozens of PHP and MySQL operations. It takes your text, inserts it into a database, links it to categories and tags, updates the internal search indexes. All in milliseconds, all invisible to you.

This book shows you exactly what happens in those milliseconds.

Not to make things more complicated. But because **understanding the engine makes you a better driver.** And more importantly, it means you can choose a different car when the one you have isn’t enough anymore.

Why Right Now

The web is changing fast. Artificial intelligence generates content, and the “jack-of-all-trades” webmaster role is under pressure like never before.

In that situation, there are two paths forward.

The first is to stay where you are, add plugins, hope the next update doesn’t break anything, and compete in a market that’s increasingly crowded and increasingly less rewarding.

The second is to truly understand how the web works, build solid skills that no AI can replace, and move up to professional tools that put you in a different league.

This book is for those who choose the second path.

How This Book Is Structured

The journey is divided into nine practical lessons.

In the first eight you'll learn PHP and MySQL from scratch, or close to it. If you already have some knowledge of HTML you won't have any trouble. If you've never written a line of code, don't worry: I wrote this book with exactly you in mind.

Each lesson builds on the previous ones. You won't find theory for the sake of theory. Every concept is explained and immediately applied with a real example you can try on your computer.

The ninth lesson is my favorite. We'll bring together everything you've learned and I'll show you how PHP and MySQL are the engine powering a professional CMS like **Drupal**, a tool used today by universities, governments and large companies around the world, and the natural next step for anyone who wants to work on the web at a higher level.

The Tools We'll Use

Unlike the first edition of this book, we won't be using an external hosting space for the exercises. All the work will take place on your computer, locally, thanks to **XAMPP** (Windows) or **MAMP** (Mac), free software that installs a complete web server with PHP and MySQL directly on your machine.

This means you can work anywhere, without depending on internet connections or external accounts. When you're ready to take your work online, the transition is simple.

To write the code we'll use a text editor. I recommend **Visual Studio Code**, free and available for all operating systems, or the classic **Notepad++** if you're already used to it.

> ***A note on code blocks*** *In some lessons the PHP files are long and are shown in multiple separate blocks to make reading easier. When you find the note (same file …) between two blocks, it means the code continues in the same file, copy everything in sequence into your editor without interruptions.*

A Personal Note

I wrote the first version of this book in 2014. At the time it worked well: many readers wrote to tell me it had been the starting point for a career on the web.

Today I'm rewriting it from scratch, not only to update the code, but because the context has completely changed. Anyone working on the web in 2026 needs different skills, a broader vision, and tools that keep pace with a constantly evolving industry.

What hasn't changed is the approach: **100% practical**, no fluff, with real examples that actually work.

Happy reading and happy coding.

Lesson 1

Why Learn PHP and MySQL in 2026

Let’s start with a simple question: why should you learn to program in PHP and MySQL today, when there are tools that do everything automatically?

The answer is just as simple. Because those tools are built with exactly PHP and MySQL. And anyone who doesn’t understand how they work is always at the mercy of those who built them.

Think about WordPress. Every time you install a plugin, every time you publish an article, every time a user registers on your site, WordPress runs dozens of PHP and MySQL operations. It runs them for you, automatically, without you having to do anything. Convenient, sure. But when something breaks, and sooner or later something always breaks, you don’t know where to start looking.

This course gives you the knowledge to understand what’s happening under the hood. Not just in WordPress, but in any modern web application, including **Drupal**, which will be our final destination.

How a Dynamic Website Works

Before writing the first line of code it’s useful to understand the system we’re working in. A dynamic website is made up of three elements that communicate with each other.

The first is the **Browser**, the program you use to browse the internet, whether that's Chrome, Firefox, Safari or Edge. The browser is the starting point: the user types an address and the browser makes a request.

The second is the **Server**, a computer that's always on and connected to the internet that receives the browser's request, processes it and returns a response. Inside the server runs software called **Apache**, which is the engine that interprets the PHP code and prepares the page to return to the browser.

The third is the **MySQL Database**, an organized archive where all the site's information is stored: articles, users, settings, comments. When the server processes a request, it queries the database for the information it needs to build the page.

The request-response flow diagram between Browser, Apache Server, PHP and MySQL Database.

The flow is always the same: the browser asks, the server processes, the database responds, the server assembles the page and sends it back to the browser. All of this happens in milliseconds.

> ***WordPress does it for you… but do you know how?*** *When a visitor opens a page on your WordPress site, the server launches dozens of MySQL queries to retrieve the content, theme, widgets and settings. PHP assembles everything and returns the page to the browser. From now on, when you see a WordPress page load, you'll know exactly what's happening behind the scenes.*

In Drupal variables work the same way, but you can see them.

When you develop a theme or a module for Drupal, you work directly with PHP variables that contain the content data. For example, to display the title of a Drupal page you write `$variables['node']->getTitle()`, which is nothing more than a variable containing the title retrieved from the database. In WordPress that variable exists too, but it's hidden inside functions like `the_title()` which prints it for you without showing it. Drupal puts you in front of the code, WordPress hides it from you.

Static Site and Dynamic Site

These two concepts are worth keeping straight, because they come up all the time.

A **static site** is made up of HTML files already prepared on the server. When the browser requests them, the server returns them exactly as they are, without processing anything. Fast, but rigid: to change content you have to edit the file by hand.

A **dynamic site**, on the other hand, has no pre-built pages. Pages are constructed on the fly, every time a user requests them. The server pulls data from the database, processes it with PHP and assembles the page in real time. More complex, but infinitely more flexible.

All modern CMS platforms, WordPress, Drupal, Joomla, work dynamically. The database contains the content, PHP assembles it, the browser displays it.

Installing XAMPP on Your Computer

To run our tests we don’t need a real server on the internet. We install everything locally, directly on our computer. The tool we’ll use is called **XAMPP**.

XAMPP is a free package that installs everything we need in one go: the Apache server, PHP and the MySQL database. Once installed, your computer becomes a fully functional web server, accessible only by you locally.

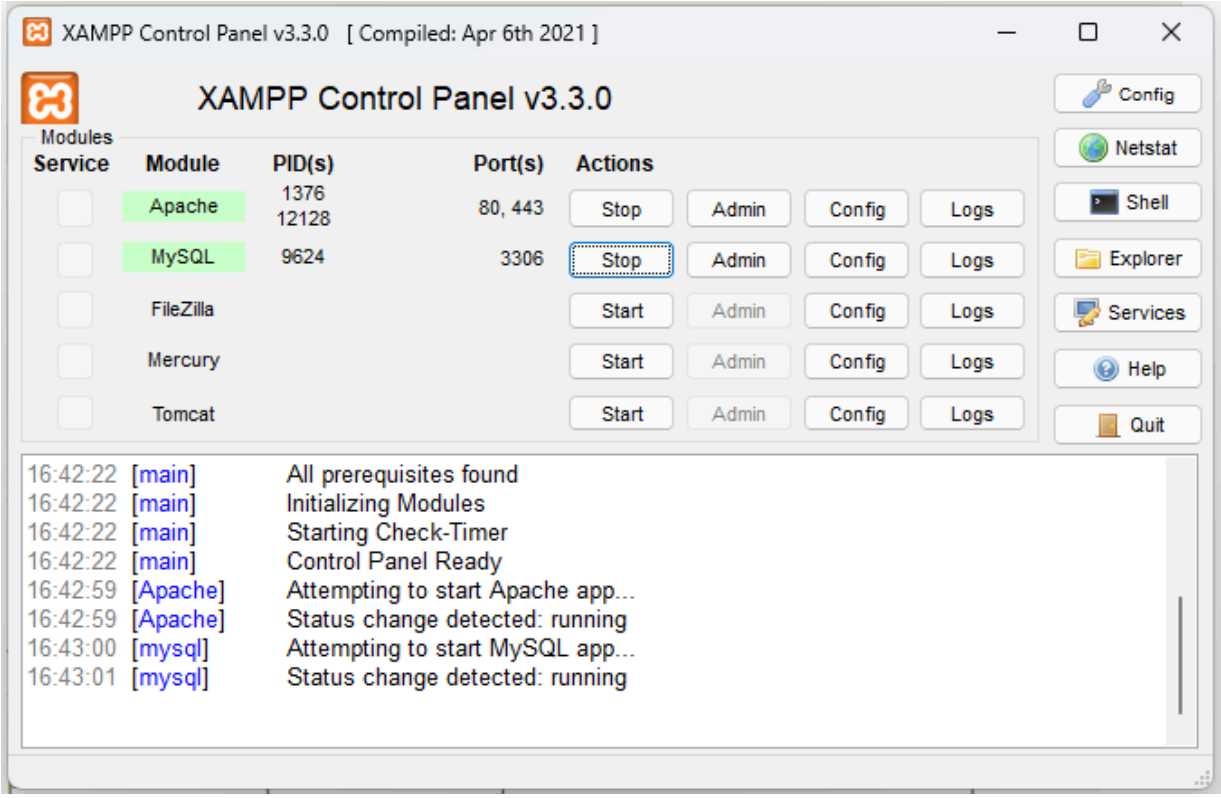
XAMPP is available for three operating systems. **Windows** is the version we’ll use as our reference in this book. On **Mac** the equivalent package is called **MAMP** and is just as simple to use. On **Linux** XAMPP exists but it’s more common to install Apache, PHP and MySQL separately via the terminal.

To download XAMPP go to:

https://www.apachefriends.org

Download the Windows version and run the installer. During the installation leave everything selected as default: we need Apache, MySQL and PHP. The installation will be complete in a few minutes.

At the end the **XAMPP control panel** opens automatically. Click **Start** next to Apache and then **Start** next to MySQL. If everything works the two rows turn green.



The XAMPP control panel with Apache and MySQL services running (green rows).

To verify that everything works open the browser and type:

http://localhost

If you see the XAMPP welcome page you're ready to begin.

Where to Put the Files

All the PHP files we'll create during the course must be saved in a specific XAMPP folder. On Windows this folder is located at:

C:/xampp/htdocs

This is the root folder of your local server. Everything you put here is accessible from the browser by typing http://localhost followed by the file name.

I recommend creating a subfolder called **php-course** inside htdocs. Your files will all go there and you'll reach them from the browser with:

http://localhost/php-course/filename.php

Your Text Editor

To write PHP code you need an editor designed for code, not Word or Pages. I recommend **Visual Studio Code**, which can be downloaded for free from:

https://code.visualstudio.com

It's lightweight, free and available for Windows, Mac and Linux. If you're already used to **Notepad++** you can continue using it without any issues.

Your First PHP Page

We're ready to write the first line of code. Open your editor, create a new file and write this code:

```
<html>
<head>
    <title>My First PHP Page</title>
</head>
<body>
    <?php echo "My first web page with PHP"; ?>
```

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

Save the file with the name `first.php` inside the folder `C:\xampp\htdocs\php-course`.

Open the browser and type:

http://localhost/php-course/first.php

You should see the text: My first web page with PHP

It might seem like a trivial thing, but at this moment you've just run PHP code on a web server. It's a good start.

What's Happening Behind the Scenes

The outer HTML structure will be familiar: the `<html>`, `<head>`, `<body>` tags are the basic structure of any web page.

The new part is this:

```
<?php echo "My first web page with PHP"; ?>
```

The tag `<?php` tells the server: PHP code starts here. The tag `?>` says: PHP code ends here, return to normal HTML. Everything found between these two tags is processed by the server before being sent to the browser. The browser never sees the PHP code: it only receives the result, which is the final HTML already prepared.

The `echo` instruction is used to display text on screen. It's one of the instructions you'll use most throughout the entire course.

If you right-click on the page in the browser and choose "View page source," you won't find any trace of the PHP code. You'll only see HTML. The server processed the PHP and returned only HTML to the browser.

> ***Note:*** *PHP files must always be saved with the* `.php` *extension and in lowercase letters. Linux servers, which are the ones you'll find on most professional hosting platforms, are case-sensitive with file names. Get used to working in lowercase right away to avoid problems when you take your work online.*

Exercise

Before moving on to the next lesson, try modifying the text inside the quotes of the `echo` instruction and reload the page in the browser. Also try adding a second `echo` with a different text and see what happens.

In the next lesson we'll get into the heart of programming with variables and flow control, the tools that allow you to write code that reasons and makes decisions.

Lesson 2

Variables, Arrays and Loops: The Essential Tools

In the previous lesson we wrote our first PHP page and saw how the server processes the code before sending it to the browser. Now let’s build the toolbox you’ll use in every subsequent lesson: variables, arrays and loops.

These three concepts aren’t theory. They’re the gears that power every web application. Every time WordPress retrieves a list of articles, every time Drupal displays a product catalog, the same mechanism is at work underneath: a variable holds the data, an array organizes it, a loop processes it one item at a time.

Variables

A variable is a named container. It stores a piece of information, a text, a number, a result, that you can retrieve and reuse in different parts of the code.

In PHP every variable starts with the `$` symbol followed by the name you want to give it. Let’s look at a practical example right away. Create a file `variables.php` in the folder `C:\xampp\htdocs\php-course`:

```
<?php
$first_name = "Daniele";
$last_name  = "Venditti";
$website    = "progettimultimediali.com";

echo "Welcome to the website of "
```

```
    . $first_name . " " . $last_name
    . " - " . $website;
?>
```

Open it in the browser at **http://localhost/php-course/variables.php** and you'll see the string assembled by the code. The dot `.` is used to concatenate multiple pieces of text into a single output.

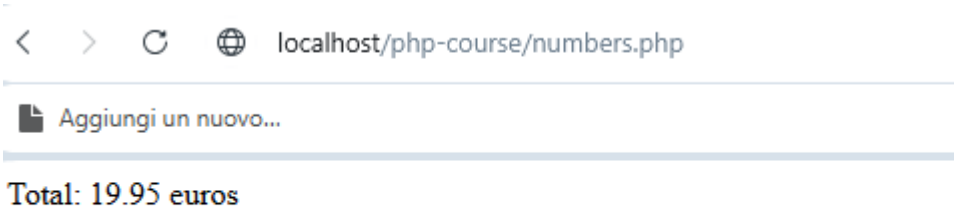
The browser displays the result of concatenating the variables $first_name, $last_name and $website.

Variables don't just hold text. They can also hold numbers, and in that case no quotes are used. Create a file `numbers.php`:

```
<?php
$price    = 3.99;
$quantity = 5;
$total    = $price * $quantity;

echo "Total: " . $total . " euros";
?>
```

PHP runs the calculation `$price * $quantity` automatically and assigns the result to `$total`. The `echo` instruction displays the final result.



The browser displays the total calculated by multiplying $price by $quantity.

When you name a variable follow these rules: don't start with a number, don't use spaces (use underscores instead: `$user_name`), no special characters like accents. Use clear and descriptive names, `$n` says nothing, `$user_name` is immediately clear.

> ***WordPress does it for you… but do you know how?*** *Every time WordPress retrieves a user's name, the title of an article, or the content of a widget, it stores them in PHP variables exactly like the ones you just learned. The* `$current_user` *variable, for example, contains all the data of the currently logged-in user. It's a variable just like* `$first_name`*, only filled automatically by WordPress instead of by you.*

In Drupal variables work the same way, but you can see them.

When you develop a Drupal theme, you work directly with variables that contain the content data. To display the title of a page you write `$variables['node']->getTitle()`, a variable with the title retrieved from the database. In WordPress that same variable exists, but it's hidden

inside the `the_title()` function that prints it for you. Drupal puts you in front of the code, WordPress hides it from you.

Arrays: Variables That Hold Multiple Values

A regular variable holds one value at a time: a name, a number, a price. But when you need to manage a list of things, ten listings, a hundred users, a series of cities, you can't open a hundred separate variables. You'd use an **array**.

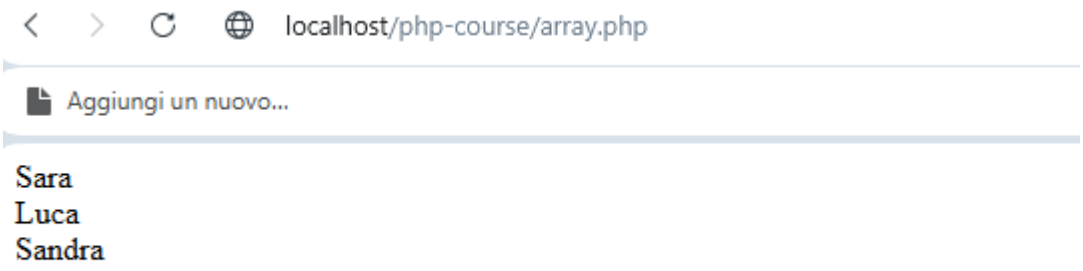
An array is a special variable that holds multiple values organized in sequence. Think of a row of numbered drawers: each drawer has its number (the **index**) and inside there's a value.

Create a file `arrays.php`:

```php
<?php
$names = array('Sara', 'Luca', 'Sandra');

echo $names[0] . '<br>';
echo $names[1] . '<br>';
echo $names[2] . '<br>';
?>
```

PHP automatically numbers the drawers starting from zero: `$names[0]` is "Sara", `$names[1]` is "Luca", `$names[2]` is "Sandra". The number in square brackets is the index that indicates which element we want to use.



The browser displays the three values of the $names array, each on a separate line thanks to the HTML `<br>` tag.

> ***If you know the first edition of this book*** *In the 2014 version the `MYSQL_FETCH_ARRAY` function was explained. The concept was right: it took data from a MySQL table and transformed it into a PHP array that could be read with a `while` loop. That function no longer exists, though, PHP removed it in version 7. Today the same result is achieved with PDO and `fetchAll(PDO::FETCH_ASSOC)`, which you'll see from Lesson 4 onwards. The array you receive is structured in exactly the same way: one element per table row, with keys equal to the column names. The syntax changes, the concept is identical.*

The for Loop: Iterating Through an Array Automatically

In the previous example we wrote three separate `echo` statements, one for each name. If the array had a hundred elements we'd need a hundred `echo` statements. That's clearly not the right approach. That's exactly why **loops** exist: blocks of code that repeat automatically.

The `for` loop is designed for numeric arrays. Create a file `for-loop.php`:

```php
<?php
$names = array('Sara', 'Luca', 'Sandra');
$count = count($names);

for ($i = 0; $i < $count; $i++) {
    echo $names[$i] . '<br>';
}
?>
```

The result is identical to the previous example, but now the code works with any number of names: you can add a hundred to the array without touching the loop.

Let's analyze the key line piece by piece:

```php
for ($i = 0; $i < $count; $i++)
```

The `for` loop has three parts separated by semicolons. `$i = 0` declares the counter and sets it to zero, the index of the first element. `$i < $count` is the continuation condition: the loop runs while the counter is less than the number of elements, which `count($names)` calculates for us (in our case it returns 3). `$i++` increments the counter by 1 each round. When `$i` reaches 3, the condition `$i < 3` becomes false and the loop stops.

Associative Arrays: Keys Instead of Numbers

In the previous example the indexes were numbers: 0, 1, 2. But in PHP indexes can also be text. This type of structure is called an **associative**

array and it's fundamental: it's exactly the form in which PHP receives data from MySQL.

When you read a row from the `listings` table in the database, PHP hands it to you like this:

```
$listing = [
    'title'    => 'Sunny apartment in Rome',
    'city'     => 'Rome',
    'price'    => 250000,
    'category' => 'Apartment',
];
```

The keys (`title` , `city` , `price` , `category`) correspond to the column names in the table. To access a value you use its key in square brackets:

```
echo $listing['title'];   // Sunny apartment in Rome
echo $listing['city'];    // Rome
echo $listing['price'];   // 250000
```

Create the file `associative-array.php` and try this code in the browser to see how it works.

The foreach Loop: The Most Natural Way to Read an Array

The `for` loop works well with numeric indexes, but becomes awkward with associative arrays. For these there's a tool made for the purpose: the **`foreach`** loop, which works with any type of array, numeric or associative.

The full form shows both the key and the value:

```
<?php
$listing = [
```

```
    'title'    => 'Sunny apartment in Rome',
    'city'     => 'Rome',
    'price'    => 250000,
    'category' => 'Apartment',
];

foreach ($listing as $key => $value) {
    echo $key . ': ' . $value . '<br>';
}
?>
```

The result will be:

```
title: Sunny apartment in Rome
city: Rome
price: 250000
category: Apartment
```

At each turn of the loop, the key of the current element goes into `$key` and its value goes into `$value`. The loop continues until it has read all the elements, then it stops.

In practice, though, the form you'll use most doesn't need `$key`. When you retrieve a list of listings from the database and want to display them one by one, you simply write:

```
foreach ($listings as $listing) {
    echo $listing['title'] . '<br>';
    echo $listing['city'] . '<br>';
    echo $listing['price'] . '<br>';
}
```

This is exactly the pattern you'll see repeated from Lesson 5 onwards every time we read data from the database. Keep it in mind: it'll be like meeting an old friend.

> ***WordPress does it for you… but do you know how?*** *The WordPress Loop, the one you write in themes as* `while (have_posts())` *followed by* `the_post()`*, is conceptually a disguised* `foreach`*. WordPress takes the result of a MySQL query, transforms it into an array, and lets you iterate through it. The* `the_title()` *function inside the loop does nothing more than print the title of the current element of the array. Now that you know* `foreach` *and associative arrays, the WordPress Loop mechanism holds no more secrets.*

In Drupal `foreach` is everywhere.

In Drupal templates written with the Twig language the syntax is `{% for item in items %}`, but the concept is identical: a loop that goes through an array of data returned from the database. In PHP modules you write `foreach` exactly as we've seen here. Understanding this tool well means understanding 50% of the PHP code you'll encounter working with Drupal.

Flow Control: if, else and Operators

Variables and arrays give us data. Loops let us iterate through it. But a useful program must also be able to **make decisions**: do one thing if a condition is true, do another if it's false. This is called flow control and is handled with the `if` and `else` instructions.

```php
<?php
$price = 150000;

if ($price < 200000) {
    echo "Affordable listing.";
} else {
    echo "Premium listing.";
```

```
}
?>
```

In other words: if (`if`) the price is less than 200,000, display “Affordable listing”. Otherwise (`else`) display “Premium listing”. Since 150,000 is less than 200,000, the result will be “Affordable listing.”

Important note: in PHP the comparison operator is the double equals `==` and not the single `=`. The single `=` assigns a value to a variable, the double `==` compares two values. It’s the most classic mistake for beginners, keep it in mind.

You can also combine `if` and `else` with a third intermediate branch using `elseif`:

```php
<?php
$price = 350000;

if ($price < 200000) {
    echo "Affordable listing.";
} elseif ($price < 500000) {
    echo "Mid-range listing.";
} else {
    echo "Luxury listing.";
}
?>
```

PHP evaluates the conditions from top to bottom and stops at the first one that matches.

Comparison Operators

Besides the double equals `==` there are other comparison operators you'll use often:

Operator	Meaning
`==`	equal to
`!=`	not equal to
`>`	greater than
`<`	less than
`>=`	greater than or equal to
`<=`	less than or equal to
`&&`	and (both conditions must be true)
`||`	or (at least one of the conditions must be true)

Putting It All Together: `if` Inside a `foreach`

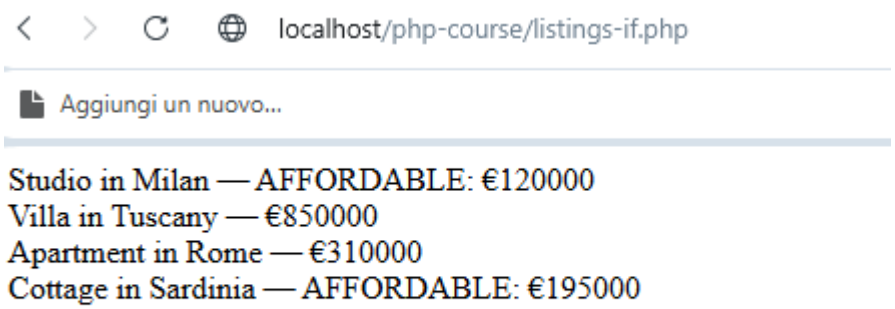
Now let's combine what we've learned. A `foreach` loop that iterates through an array of listings and uses `if` to highlight those below a certain price threshold:

```php
<?php
$listings = [
    ['title' => 'Studio in Milan',     'price' => 120000],
    ['title' => 'Villa in Tuscany',    'price' => 850000],
    ['title' => 'Apartment in Rome',   'price' => 310000],
    ['title' => 'Cottage in Sardinia', 'price' => 195000],
];

foreach ($listings as $listing) {
    if ($listing['price'] < 200000) {
        echo $listing['title']
            . ' — AFFORDABLE: €' . $listing['price'] . '<br>';
    } else {
        echo $listing['title'] . ' — €' . $listing['price'] . '<br>';
```

```
    }
}
?>
```

Save the file as `listings-if.php` and open it in the browser. You'll see the complete list with the affordable listings highlighted.



The foreach loop goes through the listings array and the if adds the AFFORDABLE label to prices below the threshold.

This is exactly the type of logic you'll use in the following lessons when you work with real data from the database: a `foreach` that iterates through the results of a MySQL query, and an `if` that decides how to display each row.

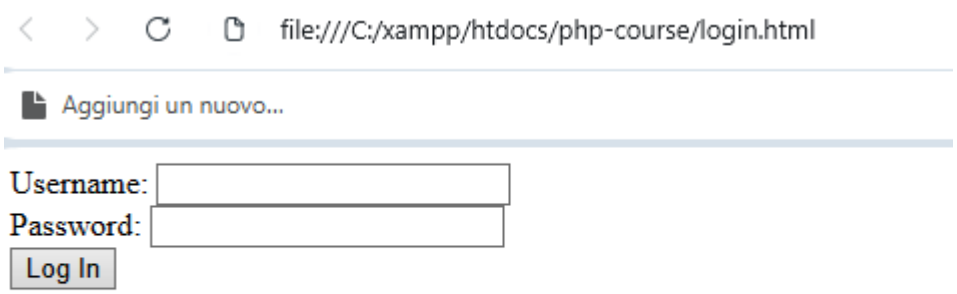
Exercise: A Simple Login System

Let's close the lesson with a practical example that brings together variables, arrays and flow control in a real context: a login system that checks the credentials entered in an HTML form.

First let's create the form. Create a file `login.html`:

```
<html>
<head>
    <title>Login</title>
</head>
<body>
    <form action="login.php" method="post">
        Username: <input name="username" type="text"><br>
        Password: <input name="password" type="password"><br>
        <input type="submit" value="Log In">
    </form>
</body>
</html>
```

Open it from the browser at **http://localhost/php-course/login.html**.



The HTML login form with username and password fields and the submit button.

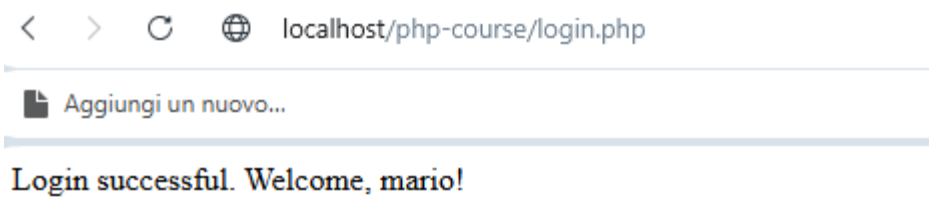
Now create the file `login.php` that receives the data from the form and checks it:

```
<?php
$username = $_POST["username"];
$password = $_POST["password"];

$valid_users = [
    'mario' => 'rossi',
    'carlo' => 'bianchi',
];
```

```php
if (isset($valid_users[$username])
    && $valid_users[$username] === $password) {
    echo "Login successful. Welcome, " . $username . "!";
} else {
    echo "Incorrect username or password. Please try again.";
}
?>
```

Try the form: first enter a wrong combination, then try `mario` / `rossi` and then `carlo` / `bianchi`.



The login.php page displays the welcome message after entering valid credentials.

In this file there are three new concepts worth analyzing.

`$_POST["username"]` is a special associative array that PHP fills automatically with the data sent from the form. The keys correspond to the `name` attributes of the HTML fields. It's the same type of associative array we saw earlier, only filled by the browser instead of by us.

`$valid_users` is an associative array where the key is the username and the value is the corresponding password. It's a list of valid users. In the previous version we would have written a separate `if` for each user; with the array we can manage as many as we want.

`isset($valid_users[$username])` checks whether a key equal to the entered username exists in the array. If it exists, it compares the value (the stored password) with the one entered in the form. It's a combination of array and `if` in a single line.

> ***WordPress does it for you… but do you know how?*** *The WordPress login system works with exactly this principle. When you enter your credentials, WordPress retrieves from the database the user corresponding to the email, an associative array with all their data, and then compares the entered password with the stored one. If they match, it creates a session. The logical structure is identical to ours: array,* `if`*, comparison. Just bigger and more robust.*

In Drupal the Form API handles all of this for you.

Drupal also uses `$_POST` and associative arrays internally, but when you write code for Drupal you never touch them directly. You use the Form API, which takes care of collecting the data, validating it and passing it to you already clean in an array. The comparison logic remains the same, but the framework manages it in a standardized and secure way. You'll better understand this difference when you see how Drupal handles login in Lesson 9.

A Note on Security

The login system we just built is useful for understanding the logic, but it's not suitable for a real site. The credentials are written directly in the code and the passwords are not protected. In the following lessons we'll learn to

manage users through the MySQL database and to encrypt passwords correctly. For now the important thing is to have the tools clear: variables, arrays, loops and flow control.

Exercise

Open the `listings-if.php` file we created in this lesson and add a third branch with `elseif` to create three price ranges: below 200,000 “Affordable”, between 200,000 and 500,000 “Mid-range”, above 500,000 “Luxury”. Then add a fifth listing to the array with a price of your choice and verify that it falls in the right range.

As a second exercise, modify `login.php` by adding a third valid user to the `$valid_users` array and try logging in with the new credentials.

In the next lesson we enter the world of databases. We’ll see what MySQL is, how a table is structured and how it’s managed with phpMyAdmin, the graphical tool that XAMPP has already installed for you.

Daniele Venditti

Continue the Journey

If this ebook has been useful to you, you can continue to explore the topics covered through the other resources of the project.

Website

If you want to continue exploring web development and discover new ebooks, articles and updated resources, visit the official website:

☞ https://www.progettimultimediali.com/ebook

Newsletter

Receive updates on new ebooks and content dedicated to Drupal and web development:

☞ Subscribe to the newsletter

YouTube Channel

Tutorials, in-depth content and practical explanations in video format:

☞ Subscribe to the YouTube channel

Thank you for reading this ebook.

If you found it useful, leaving a review in the store where you purchased it helps other developers discover this content.