ScopeTreasure Courses

PHP COURSES · LESSON PH.6 · CODE SCOPE SERIES

Functions

A function is a named block of code you can run again and again — learn function, parameters, return values, and why a variable inside stays inside.

Introduction

By now your pages repeat the same jobs: work out a total, print a greeting, check a grade. Copying those lines everywhere means fixing a bug in five places. A **function** solves it: you write the job once, give it a name, and call that name whenever you need it.

A function takes **parameters** (the inputs it needs) and can **return** a value (the answer it gives back). `function add($a, $b)` takes two numbers and returns their sum. The caller writes `$total = add(250, 3);` and gets 750 back — without ever seeing the maths inside.

PHP also ships with hundreds of built-in functions you already met: `echo`, `date()`, `count()`, `strlen()`. Writing your own works exactly the same way.

(Roman Urdu: Function matlab "aik kaam, aik naam" — aik dafa likho, jab chaho bula lo. `return` jawab wapas deta hai.)

One rule surprises every beginner: a variable made inside a function stays inside it. This is called **scope**. It is not a limitation — it is protection. Your function's `$total` can never accidentally overwrite the page's `$total`.

Key points

- **Define with `function name() { ... }`, call with `name()`.** The definition does nothing until something calls it.
- **Parameters are the inputs, listed inside the brackets.** `function greet($name)` takes one value; the caller passes the real value.
- **Arguments are the values you pass when you call.** In `greet("Ali")`, `"Ali"` is the argument for the parameter `$name`.
- **`return` ends the function immediately and gives a value back.** Any line after `return` never runs.
- **A function without `return` gives back `null`.** Use it for jobs that only do something, like printing.
- **A default value makes a parameter optional:** `function greet($name = "Guest")` works with or without an argument.
- **Variables inside a function are local.** They are born when the function runs and die when it returns.
- **A variable outside is invisible inside** — unless you pass it in as a parameter, or (rarely) declare it `global`.
- **Name functions after what they do:** `calculateTotal()`, `isPass()`, `formatPrice()` — a good name is half the documentation.

- **One function, one job.** A function that calculates AND prints AND saves is three functions wearing one name.

Examples

Example 1 — Your first function

```
<?php
function salam($name) {
    echo "Hello, $name!<br>";
}
salam("Ali");
salam("Sara");
salam("Bilal");
?>
```

What you will get:

```
Hello, Ali!
Hello, Sara!
Hello, Bilal!
```

One definition, three calls. Change the greeting in one place and all three calls change.

Example 2 — return gives the answer back

```
<?php
function add($a, $b) {
    return $a + $b;
}
$total = add(250, 3);
echo "Total: $total";
?>
```

What you will get:

```
Total: 750
```

`return` hands the value back to the caller, who stores it in `$total`. Without `return`, the function could print the answer but the caller could never use it in further maths.

Example 3 — A default parameter

```
<?php
function greet($name = "Guest") {
    return "Hello, $name!";
}
echo greet("Ali");
echo "<br>";
echo greet();
?>
```

What you will get:

```
Hello, Ali!
Hello, Guest!
```

Callers who pass a name get it used; callers who pass nothing get the default. Defaults must come after the required parameters.

Example 4 — Scope: inside stays inside

```
<?php
$total = 100;
function addTax($price) {
    $total = $price + 10; // this $total lives ONLY here
    return $total;
}
echo addTax(100);
echo "<br>";
echo $total; // still 100 — the outside $total never changed
?>
```

What you will get:

```
110
100
```

The `$total` inside the function and the `$total` outside are two different boxes that happen to share a name. The function cannot touch the outside one.

Example 5 — A real helper: pass or fail

```
<?php
function grade($marks) {
    if ($marks >= 90) {
        return "A";
    } elseif ($marks >= 75) {
        return "B";
    } elseif ($marks >= 50) {
        return "C";
    }
    return "F";
}
echo grade(82);
echo "<br>";
echo grade(45);
?>
```

What you will get:

```
B
F
```

`return` ends the function at once, so the first matching grade is the answer and the rest never runs. Now any page can grade marks with one line.

Example 6 — Built-in functions are functions too

```
<?php
$name = "ali ahmed";
echo strtoupper($name);
echo "<br>";
echo strlen($name);
echo "<br>";
echo date("Y");
?>
```

What you will get:

```
ALI AHMED
9
2026
```

`strtoupper` uppercases, `strlen` counts characters, `date` gives the year. Your own functions and PHP's built-ins are called in exactly the same way.

Quick recap

- `function name($param) { ... }` defines; `name($value)` calls.
- Parameters are inputs; `return` gives the answer back and ends the function.
- Default values make parameters optional — they go last.
- Variables inside are local; the outside world cannot see them.
- Name functions after their job, and give each one a single job.

Common mistakes

1. **Calling a function before it is defined in the same file flow** — in plain PHP files, define functions at the top or in an included file before calling them. Fix: put definitions first, calls after.
2. **Forgetting `return` and wondering why the value is empty** — a function that only echoes gives back `null`. Fix: `return` the value when the caller needs it.
3. **Echoing inside a function and also echoing its result** — `echo greet("Ali");` where `greet` already echoes prints the greeting and then a blank. Fix: either echo inside OR return — pick one.
4. **Expecting an outside variable to work inside** — `$total` outside is invisible inside the function. Fix: pass it as a parameter.
5. **Putting a default parameter before a required one** — `function f($a = 1, $b)` is an error in older PHP and confusing everywhere. Fix: required parameters first, defaults last.
6. **Writing one giant function for the whole page** — it becomes untestable and unreusable. Fix: split it into small functions with names that say the job.
7. **Naming a function the same as a built-in** — `function count()` collides with PHP's own `count()`. Fix: use clear, specific names like `countStudents()`.

Practice

1. Write a function `double($n)` that returns twice its input, and print `double(7)`.
2. Write `greet($name)` that returns "Hello, NAME!" — call it with your name and with no argument (add a default).
3. Write `isPass($marks)` that returns `true` when marks are 50 or more, `false` otherwise — test it with 45 and 75.
4. Write `billTotal($price, $qty)` that returns price times quantity, and use it to print the total for 3 cups at 250.
5. Write a function that takes a name and returns it uppercased with `strtoupper` — then print the result.
6. Deliberately try to echo a variable made inside a function from outside it — read the warning, and write down in one line what scope means.

7. Split this job into two functions: one that works out a 10% discount amount, and one that prints "You pay: X" — then use both for a bill of 6000.

Practice quiz

1. What does return do inside a function?

- It prints the value and keeps running
- It ends the function immediately and gives the value back
- It restarts the function
- It deletes the function

Answer: It ends the function immediately and gives the value back

2. What does a default value do in function greet($name = "Guest")?

- It forces every caller to pass a name
- It lets callers skip the argument: greet() uses "Guest"
- It deletes the parameter
- It makes the function run twice

Answer: It lets callers skip the argument: greet() uses "Guest"

3. A variable made inside a function is used outside it. What happens?

- It works, variables are shared
- It does not exist outside: variables inside a function are local to it
- It becomes a global automatically
- The function runs twice

Answer: It does not exist outside: variables inside a function are local to it

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/ph-functions · More free tools and notes at https://scopetreasure.com