ScopeTreasure Courses

PHP COURSES · LESSON PH.5 · CODE SCOPE SERIES

Loops: for, while and foreach

Loops repeat work without rewriting it — learn for loops, while loops, foreach over arrays, break and continue, and how to avoid infinite loops.

Introduction

Imagine printing the numbers 1 to 100 by writing 100 echo lines. Or building a list of 50 students by copying one block 50 times. That is not programming — that is typing. A **loop** repeats a block of code for you, as many times as you say.

PHP gives you three loops. `for` is for when you know how many rounds you need — count from 1 to 10. `while` is for when you do not know — keep going until something becomes true. `foreach` is for walking through an array, one item at a time — the loop you will use most in real PHP, because real PHP is mostly lists of things.

Every loop needs a way to stop. A loop that never stops is an **infinite loop**: the page hangs, the server works forever, and you have to kill it. The rule is simple — every loop must move towards its own ending.

(Roman Urdu: Loop ka matlab hai "dohrana" — aik hi kaam baar baar, likhe baghair. `for` gin-ti ke liye, `while` jab tak ke liye, `foreach` fehrist ke liye.)

Key points

- **A `for` loop has three parts in its head:** start (`$i = 1`), condition (`$i <= 5`), update (`$i++`). All three sit in one line.
- **A `while` loop checks its condition before every round.** If the condition starts false, the block never runs.
- **A `do...while` loop runs once first, then checks.** Useful when the first round must happen no matter what.
- **`foreach` walks an array one item at a time:** `foreach ($fruits as $fruit)` gives you each value in `$fruit`.
- **`foreach ($arr as $key => $value)`** gives you the key as well — vital for associative arrays.
- **`break` leaves the loop completely.** `continue` skips only this round and jumps to the next.
- **`$i++` adds 1; `$i--` subtracts 1.** Forgetting the update is how infinite loops are born.
- **A loop counter usually starts at 0 for arrays** (because array indexes start at 0) and at 1 for human counting.
- **Loops can sit inside loops** (nested loops) — for example, a table with rows and columns — but keep them shallow.

- **Never trust a `while` condition you cannot see changing.** Before you run it, point at the line that changes the condition and say it out loud.

Examples

Example 1 — for: count 1 to 5

```
<?php
for ($i = 1; $i <= 5; $i++) {
    echo $i . "<br>";
}
?>
```

What you will get:

```
1
2
3
4
5
```

`$i` starts at 1, the block runs while `$i <= 5`, and `$i++` adds 1 after each round. When `$i` becomes 6, the condition is false and the loop stops.

Example 2 — while: fill cups until the jug is empty

```
<?php
$cups = 0;
while ($cups < 3) {
    $cups++;
    echo "Cup $cups filled<br>";
}
echo "Done";
?>
```

What you will get:

```
Cup 1 filled
Cup 2 filled
Cup 3 filled
Done
```

The condition is checked before every round. If `$cups` had started at 3, nothing would print at all.

Example 3 — foreach: read a list

```
<?php
$fruits = ["apple", "banana", "mango"];
foreach ($fruits as $fruit) {
    echo $fruit . "<br>";
}
?>
```

What you will get:

```
apple
banana
mango
```

No counting, no indexes, no chance of an off-by-one error. `foreach` handles the walking; you handle what happens to each item.

Example 4 — foreach with keys

```
<?php
$marks = ["Ali" => 88, "Sara" => 92, "Bilal" => 76];
foreach ($marks as $name => $score) {
    echo "$name: $score<br>";
}
?>
```

What you will get:

```
Ali: 88
Sara: 92
Bilal: 76
```

With `$key => $value` you get the label and the value together. This is the shape of almost every list page you will ever build: a loop over rows, printing one line per row.

Example 5 — break and continue

```
<?php
for ($i = 1; $i <= 10; $i++) {
    if ($i == 3) {
        continue; // skip 3
    }
    if ($i == 7) {
        break; // stop at 7
    }
    echo $i . " ";
}
?>
```

What you will get:

```
1 2 4 5 6
```

`continue` skips round 3 only — 3 never prints. `break` leaves the loop at 7, so 7, 8, 9 and 10 never print.

Example 6 — A loop that builds an HTML list

```
<?php
$students = ["Ali", "Sara", "Bilal", "Hira"];
echo "<ul>";
foreach ($students as $student) {
    echo "<li>$student</li>";
}
echo "</ul>";
?>
```

What you will get:

```
• Ali
• Sara
• Bilal
• Hira
```

This is the real pattern: PHP loops over data and prints HTML around it. The browser receives a finished list; it never knows a loop was involved.

Quick recap

- `for` counts a known number of rounds: start, condition, update in one head.

- `while` repeats until its condition turns false — the update must live inside.
- `foreach` walks an array item by item; `$key => $value` gives the key too.
- `break` leaves the loop; `continue` skips one round.
- Every loop must move towards its ending, or it never stops.

Common mistakes

1. **Forgetting the update in a while loop** — `while ($i < 5)` with no `$i++` inside runs forever and hangs the page. Fix: put the update inside the loop, and test with a small limit first.
2. **Writing `for ($i = 1; $i <= 5; $i--)`** — the counter shrinks, the condition stays true, the loop never ends. Fix: the update must move towards the condition's end.
3. **Off-by-one errors** — `for ($i = 0; $i <= 3; $i++)` on a 3-item array tries index 3, which does not exist. Fix: for arrays use `$i < count($arr)`, not `<=`.
4. **Using `$i` after the loop and expecting the last value** — after the loop `$i` holds the value that failed the condition (6 after counting to 5). Fix: save the value you need inside the loop.
5. **Modifying an array while foreach walks it** — adding or removing items mid-loop gives surprising results. Fix: build a new array, or loop over a copy.
6. **Nesting loops and reusing `$i` for both** — the inner loop destroys the outer counter. Fix: use different counters, like `$i` outside and `$j` inside.
7. **Echoing inside a loop without `<br>` or a separator** — all values run together on one line. Fix: print a separator, a `<br>`, or wrap each item in an HTML tag.

Practice

1. Write a `for` loop that prints the numbers 1 to 10, one per line.
2. Write a `while` loop that prints the even numbers from 2 to 20.
3. Make an array of 5 fruits and print each one with `foreach`, numbered 1 to 5.
4. Make an associative array of 3 students and their marks, and print "Name: marks" for each with `foreach ($marks as $name => $score)`.
5. Print the numbers 1 to 20 but skip the multiples of 3 with `continue`.
6. Search an array of names for "Sara" with a loop and `break` as soon as you find her — print "Found at position X".
7. Deliberately write a `while` loop with no update, run it, and stop the page load — then fix it and write down, in one line, why the update matters. (Keep the limit small: `while ($i < 3)`.)

Practice quiz

1. When do you use a for loop?

- When the number of rounds is unknown

- When the number of rounds is known, with start, condition and update in one head
- Only for arrays, never for counting
- Only inside functions

Answer: When the number of rounds is known, with start, condition and update in one head

2. What happens if a while loop never updates its condition?

- It stops after 10 rounds by itself
- It never stops, an infinite loop that hangs the page
- It skips to the next line
- It prints an error and continues

Answer: It never stops, an infinite loop that hangs the page

3. What is the difference between break and continue in a loop?

- They are the same
- break leaves the loop completely, continue skips only this round and goes to the next
- break skips one round, continue leaves the loop
- break works only in for loops

Answer: break leaves the loop completely, continue skips only this round and goes to the next

---

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-loops · More free tools and notes at https://scopetreasure.com