ScopeTreasure Courses

PHP COURSES · LESSON PH.8 · CODE SCOPE SERIES

Strings

A string is text inside quotes — learn single vs double quotes, variables inside strings, length, searching, replacing, and splitting text.

Introduction

Almost everything a web app handles is text: names, emails, messages, passwords, page titles. In PHP, text is a **string** — characters inside quotes. And PHP gives you a whole toolbox for working with strings: measuring them, searching inside them, changing their case, cutting them up and joining them back together.

Two facts from PH.2 return here with more weight. First, double quotes replace variables inside them (`"Hello $name"`), single quotes do not. Second, the dot `.` joins strings — the operator you will use more than any other in this lesson.

(Roman Urdu: String matlab "likha hua text" — quotes ke andar. Double quotes variable ki value daal deti hain, single quotes nahi.)

Strings also hide the most common validation jobs in web development: is this email long enough? Does this password contain a number? Is this name empty? The functions in this lesson answer all three.

Key points

- **A string is text in quotes.** `"Ali"` and `'Ali'` are both strings — but they behave differently with variables inside.
- **Double quotes interpolate:** `"Hello $name"` becomes Hello Ali. **Single quotes do not:** `'Hello $name'` stays as written.
- `strlen($str)` **counts characters**, including spaces. An empty string has length 0.
- `strtoupper($str)` **/** `strtolower($str)` change the case of the whole string.
- `strpos($haystack, $needle)` finds where a piece of text starts — or returns `false` when it is not there.
- `str_replace($find, $with, $str)` swaps one piece of text for another, everywhere it appears.
- `substr($str, $start, $length)` cuts out part of a string.
- `trim($str)` removes spaces from both ends — essential for form input.
- `explode($sep, $str)` splits a string into an array; `implode($glue, $arr)` joins an array back into a string.
- `htmlspecialchars($str)` turns `<`, `>` and `&` into safe text — never print user input on a page without it.

Examples

Example 1 — Length and case

```
<?php
$name = "Ali Ahmed";
echo strlen($name);
echo "<br>";
echo strtoupper($name);
echo "<br>";
echo strtolower($name);
?>
```

What you will get:

```
9
ALI AHMED
ali ahmed
```

`strlen` counts 9 characters — the space counts too. Case functions return a new string; the original `$name` is unchanged.

Example 2 — Searching: strpos and its false trap

```
<?php
$email = "ali@example.com";
$pos = strpos($email, "@");
echo $pos;
echo "<br>";
if (strpos($email, "@") !== false) {
    echo "Looks like an email";
}
?>
```

What you will get:

```
3
Looks like an email
```

`@` sits at position 3 (counting from 0). The trap: when the needle is at position 0, `strpos` returns `0`, and `0 == false` is true in PHP — so a careless `== false` check lies to you. Always compare with `!== false`.

Example 3 — Replacing and cutting

```
<?php
$msg = "Hello, Ali! Welcome, Ali!";
echo str_replace("Ali", "Sara", $msg);
echo "<br>";
echo substr("Hello, World!", 0, 5);
echo "<br>";
echo substr("Hello, World!", 7);
?>
```

What you will get:

```
Hello, Sara! Welcome, Sara!
Hello
World!
```

`str_replace` swaps every occurrence. `substr` with a start and a length cuts a piece; with only a start it takes everything from there to the end.

Example 4 — trim: cleaning form input

```
<?php
$typed = "   Ali   ";
echo strlen($typed);
echo "<br>";
$clean = trim($typed);
echo strlen($clean);
echo "<br>";
echo "[" . $clean . "]";
?>
```

What you will get:

```
9
3
[Ali]
```

Users type accidental spaces before and after their input. `trim` removes them from both ends. Always trim form values before checking or saving them.

Example 5 — explode and implode: strings become arrays and back

```
<?php
$csv = "apple,banana,mango";
$fruits = explode(",", $csv);
echo $fruits[1];
echo "<br>";
echo implode(" - ", $fruits);
?>
```

What you will get:

```
banana
apple - banana - mango
```

`explode` splits on the comma into an array; `implode` joins the array with a glue string. Together they are the simplest way to handle comma-separated data.

Example 6 — Printing user text safely

```
<?php
$comment = "<script>alert('hack');</script> Nice post!";
echo htmlspecialchars($comment);
?>
```

What you will get:

```
<script>alert('hack');</script> Nice post! — shown as harmless text, not run as code.
```

`htmlspecialchars` turns `<` into `<`, so the browser shows the text instead of running it. Without this, a visitor's comment could run JavaScript on every other visitor's page. Never skip it.

Quick recap

- Double quotes replace variables; single quotes do not; `.` joins strings.
- `strlen` counts; `strtoupper` / `strtolower` change case.
- `strpos` finds text — compare with `!== false`, never `== false`.
- `str_replace` swaps; `substr` cuts; `trim` cleans spaces.
- `explode` splits to an array; `implode` joins back.
- `htmlspecialchars` makes user text safe to print — always use it.

Common mistakes

1. **Checking strpos with `== false`** — when the text is found at position 0, `0 == false` is true and your code thinks it was not found. Fix: always use `!== false`.
2. **Forgetting that strlen counts bytes, not letters, for Urdu/Arabic text** — multibyte characters count as more than one. Fix: use `mb_strlen($str)` for non-English text.
3. **Printing user input without htmlspecialchars** — a `<script>` in a comment runs on every visitor's browser. Fix: escape every user value you print.
4. **Using single quotes and expecting the variable** — `'Hello $name'` prints the dollar sign. Fix: double quotes, or join with `.`.
5. **Comparing strings with `==` when case matters** — `"Ali" == "ali"` is false. Fix: compare `strtolower($a) == strtolower($b)` when case should not matter.
6. **Forgetting trim on form values** — `" Ali "` fails an exact-match check against `"Ali"`. Fix: trim every value the moment it arrives.
7. **Treating `substr` positions as starting at 1** — strings start at 0, so `substr($s, 1, 3)` skips the first character. Fix: count from 0.

Practice

1. Store your full name and print its length, its uppercase version, and its lowercase version.
2. Use `strpos` to check whether "example.com" contains "@" — write the check with `!== false` and test it with an address that has no "@".
3. Replace every "chai" with "coffee" in the sentence "I like chai. Chai is great." — how many changed, and why?
4. Take `" Sara "` with spaces, trim it, and print the length before and after.
5. Split `"red,green,blue"` with `explode`, print the second colour, then join the array with `" + "` using `implode`.
6. Cut the first 5 characters from `"Hello, World!"` with `substr` — then cut everything from position 7 to the end.
7. Print the string `<b>Hi</b>` twice: once raw (watch it turn bold) and once with `htmlspecialchars` (watch it show as text) — write down in one line why the second one is the safe way.

Practice quiz

1. What does strlen("Ali Ahmed") give?

- 8
- 9, because the space counts too
- 2
- 0

Answer: 9, because the space counts too

2. How must you test the result of strpos()?

- With == false
- With !== false, because position 0 is a real find and 0 == false is true
- With > 0 only
- You cannot test it

Answer: With !== false, because position 0 is a real find and 0 == false is true

3. Why print user text with htmlspecialchars()?

- To make it bold
- So <script> in a comment shows as harmless text instead of running as code
- To make it shorter
- It is only a style habit

Answer: So <script> in a comment shows as harmless text instead of running as code

---

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