MEDICAPS UNIVERSITY

Regular Expressions in Php

Laxmikant Soni

Learning Objectives

- By the end of this lecture, students will:
  - Understand the concept and importance of regular expressions in PHP.
  - Learn the difference between POSIX and PCRE regular expressions.
  - Use PHP functions such as `preg_match()`, `preg_match_all()`, `preg_replace()`, and `preg_split()` effectively.
  - Apply regular expressions to validate user input (e.g., emails, phone numbers, passwords).
  - Write and test patterns for searching, replacing, and extracting text from strings.
  - Handle common pitfalls like greedy vs. non-greedy matching and escaping special characters.

What is a Regular Expression?

A **regular expression** is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Syntax

In PHP, regular expressions are strings composed of **delimiters**, a **pattern**, and optional **modifiers**.

```
$exp = "/medicaps university/i";
```

Syntax

In the example above:

- `/` is the **delimiter**

- `medicaps university` is the **pattern** being searched for

- `i` is a **modifier** that makes the search case-insensitive

The delimiter can be any character that is not a letter, number, backslash, or space.
The most common delimiter is the forward slash (`/`), but when your pattern contains forward slashes it is convenient to choose other delimiters such as `#` or `~`.

Common Regular Expression Modifiers in PHP

Modifier	Description
`i`	Makes the pattern case-insensitive.
`m`	Treats the string as multi-line. `^` and `$` match the start and end of each line.
`s`	Enables “dotall” mode, where `.` matches newline characters as well.
`u`	Enables Unicode support (UTF-8).
`x`	Allows whitespace and comments inside the pattern for readability.
`A`	Forces the match to occur only at the beginning of the string.
`D`	Forces `$` to match only at the end of the string.
`U`	Inverts the “greedy” behavior of quantifiers, making them lazy by default.

Regular Expression Functions

Function	Description
`preg_match()`	Returns **1** if the pattern was found in the string and **0** if not.
`preg_match_all()`	Returns the **number of times** the pattern was found in the string (may be 0).
`preg_replace()`	Returns a **new string** where matched patterns have been replaced with another string.
`preg_split()`	Splits a string into an array using a regular expression as the delimiter.
`preg_grep()`	Returns an array of elements from the input array that match a given pattern.
`preg_filter()`	Returns an array with matched elements replaced, similar to `preg_replace()` but only includes matches.

Example: `preg_match()`

The `preg_match()` function searches a string for a pattern and returns **1 if a match is found**, and **0 if not**.

```php
<?php
$pattern = "/php/i";    // 'i' makes the search
case-insensitive
$text = "I love learning PHP!";

if (preg_match($pattern, $text)) {
    echo "Match found!";
} else {
    echo "No match found!";
}
?>
```

Example: `preg_match_all()`

The `preg_match_all()` function searches a string for a pattern and returns the **number of matches found**.
It can also return the matches themselves if you pass an array variable.

```php
<?php
$pattern = "/\d+/";    // pattern to match one or more
digits
$text = "There are 2 apples, 5 bananas, and 12
oranges.";

if (preg_match_all($pattern, $text, $matches)) {
    echo "Total matches found: " . count($matches[0]) .
"\n";
    print_r($matches[0]);  // print all matched numbers
} else {
    echo "No matches found!";
}
?>
```


Explanation:

- The pattern // matches one or more digits.
- preg_match_all() finds all matches in the string, not just the first one.
- The $matches[0] array contains all the matched values.

Example: `preg_replace()`

The `preg_replace()` function searches a string for a pattern and **replaces** all matches with a new string.

```php
<?php
$pattern = "/apple/i";   // 'i' makes it case-insensitive
$text = "I like Apple, apple juice, and APPLE pie.";

// Replace all occurrences of "apple" with "mango"
$result = preg_replace($pattern, "mango", $text);

echo $result;
?>
```

- The pattern /apple/i matches “Apple”, “apple”, and “APPLE”.
- preg_replace() replaces all matches with “mango”.

Task 1: Replace digits with

Write a PHP program that replaces **all digits** in a string with #.

*Input:**
My phone number is 9876543210.

Expected Output:
My phone number is ##########.

Task 2: Censor bad words

Write a PHP program that replaces the words **bad**, **ugly**, and **stupid** with ***.

Input:
That was a bad idea, really stupid and ugly!

Expected Output:
That was a *** idea, really *** and ***!

Task 3: Format dates

Write a PHP program that converts dates from the format **DD/MM/YYYY** to **YYYY-MM-DD**.

Input:
Today’s date is 07/09/2025.

Expected Output:
Today’s date is 2025-09-07.