How to Use PHP Regular Expressions for Pattern Matching and Data Validation

Olivia J · Follow
8 min read · 4 days ago

Listen Share More

Regular expressions are powerful tools for finding and manipulating text based on certain patterns. They can be used for various tasks such as validating user input, extracting data from web pages, replacing text in files, and more.

In this post, we will learn how to use regular expressions in PHP, a popular server-side scripting language. We will cover the basics of regular expression syntax, how to create and use regular expression objects, and some common examples of regular expression usage in PHP.

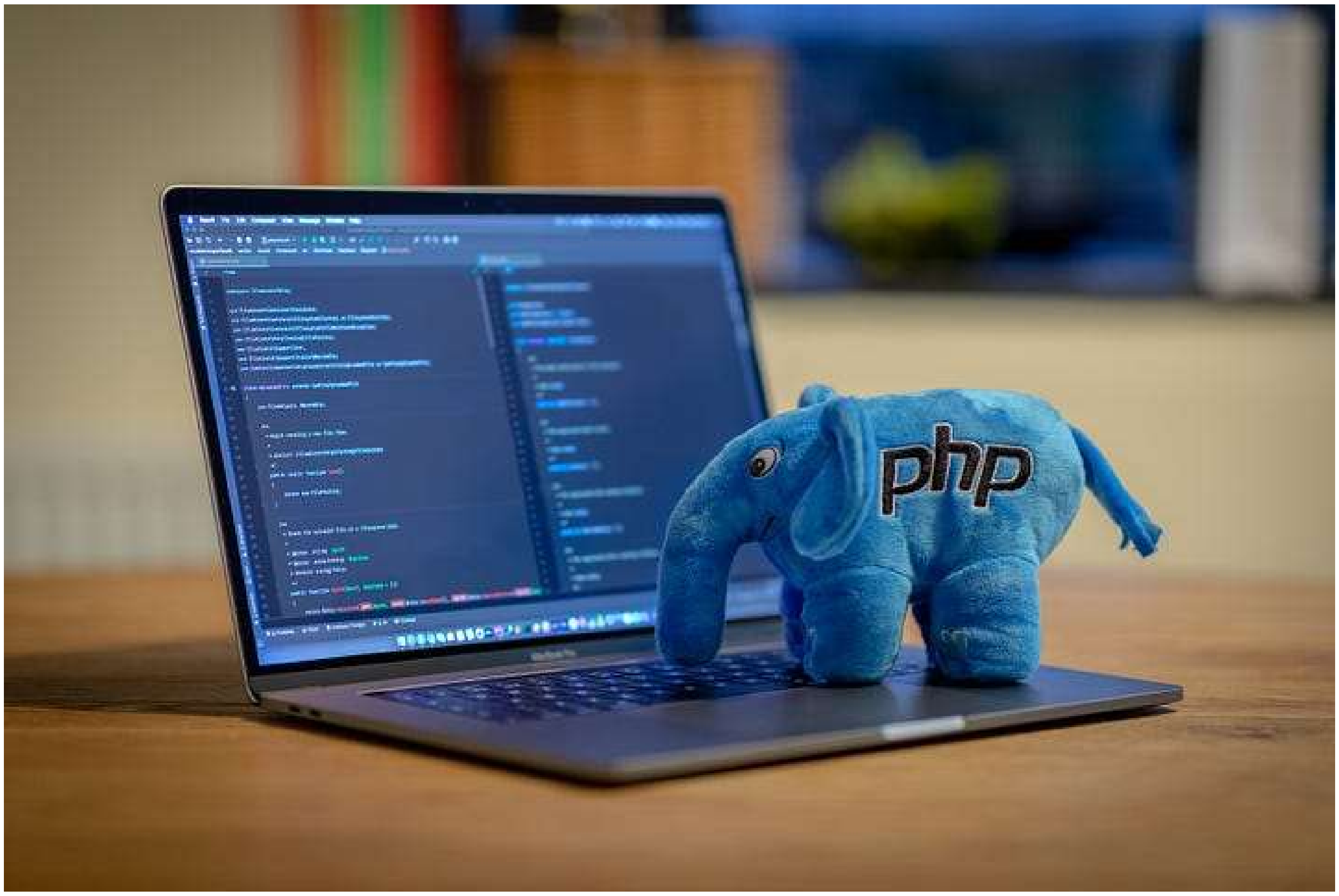

Photo by Ben Griffiths on Unsplash

What are Regular Expressions?

A regular expression (or regex) is a sequence of characters that defines a search pattern. For example, the regex `/[a-z]+/` matches one or more lowercase letters, while the regex `/[0-9]{3}-[0-9]{3}-[0-9]{4}/` matches a phone number in the format `xxx-xxx-xxxx`.

A regex can consist of literal characters (such as `a`, `b`, `c`, etc.), special characters (such as `^`, `$`, `.`, `*`, etc.), and character classes (such as `[a-z]`, `[0-9]`, `\w`, `\d`, etc.). Each character or group of characters has a specific meaning and function in the regex.

How to Create and Use Regular Expression Objects in PHP

In PHP, there are two ways to create and use regular expression objects: using the `preg_` functions or using the `PCRE` class.

The `preg_` functions are a set of built-in functions that allow you to perform various operations with regular expressions, such as matching, replacing, splitting, and filtering. The most commonly used `preg_` functions are:

- `preg_match($pattern, $subject, $matches)` - This function tries to match a regex `$pattern` against a string `$subject`. If a match is found, it returns `true` and stores the matched subpatterns in an array `$matches`. Otherwise, it returns `false`.
- `preg_replace($pattern, $replacement, $subject)` - This function replaces all occurrences of a regex `$pattern` in a string `$subject` with a string `$replacement`. It returns the modified string or `null` if an error occurs.
- `preg_split($pattern, $subject)` - This function splits a string `$subject` into an array of substrings using a regex `$pattern` as the delimiter. It returns the array of substrings or `false` if an error occurs.
- `preg_grep($pattern, $array)` - This function filters an array `$array` of strings by returning only those that match a regex `$pattern`. It returns the filtered array or `false` if an error occurs.

To use the `preg_` functions, you need to enclose your regex in delimiters (such as `/`, `#`, or `~`) and optionally add modifiers (such as `i`, `m`, or `g`) after the closing delimiter. For example:

```
// Match any word that starts with "cat"
$pattern = "/\bcat\w*/i"; // The "i" modifier makes the match case-insensitive
$subject = "I like cats and caterpillars but not catacombs.";
if (preg_match($pattern, $subject, $matches)) {
  echo "Match found: " . $matches[0] . "\n"; // Output: Match found: cats
} else {
  echo "No match found.\n";
}

// Replace all occurrences of "dog" with "puppy"
$pattern = "/dog/";
$replacement = "puppy";
$subject = "The dog chased the cat and the dog barked at the mailman.";
$new_subject = preg_replace($pattern, $replacement, $subject);
echo $new_subject . "\n"; // Output: The puppy chased the cat and the puppy barke

// Split a string by commas or spaces
```

```
$pattern = "/[,\s]+/";
$subject = "red, green blue , yellow";
$array = preg_split($pattern, $subject);
print_r($array); // Output: Array ( [0] => red [1] => green [2] => blue [3] => ye

// Filter an array of email addresses by domain name
$pattern = "/@gmail\.com$/";
$array = ["alice@gmail.com", "bob@yahoo.com", "charlie@hotmail.com", "david@gmai
$new_array = preg_grep($pattern, $array);
print_r($new_array); // Output: Array ( [0] => alice@gmail.com [3] => david@gmai
```

The `PCRE` class is an object-oriented wrapper for the `preg_` functions. It allows you to create and manipulate regular expression objects using methods and properties. The most commonly used methods and properties are:

- `__construct($pattern)` - This method creates a new regular expression object with a given regex `$pattern`.
- `match($subject)` - This method tries to match the regex against a string `$subject`. If a match is found, it returns an array of matched subpatterns. Otherwise, it returns an empty array.
- `replace($replacement, $subject)` - This method replaces all occurrences of the regex in a string `$subject` with a string `$replacement`. It returns the modified string.
- `split($subject)` - This method splits a string `$subject` into an array of substrings using the regex as the delimiter. It returns the array of substrings.
- `grep($array)` - This method filters an array `$array` of strings by returning only those that match the regex. It returns the filtered array.
- `$delimiter` - This property holds the delimiter used for the regex. It can be changed by calling the `setDelimiter($delimiter)` method.
- `$modifiers` - This property holds the modifiers used for the regex. It can be changed by calling the `setModifiers($modifiers)` method.

To use the PCRE class, you need to include it in your script using the `require_once()` function. For example:

```
// Include the PCRE class
require_once("PCRE.php");

// Create a new regular expression object
$regex = new PCRE("/\bcat\w*/i");

// Match any word that starts with "cat"
$subject = "I like cats and caterpillars but not catacombs.";
$matches = $regex->match($subject);
if (!empty($matches)) {
  echo "Match found: " . $matches[0] . "\n"; // Output: Match found: cats
} else {
  echo "No match found.\n";
}

// Replace all occurrences of "dog" with "puppy"
$regex->setPattern("/dog/");
$replacement = "puppy";
$subject = "The dog chased the cat and the dog barked at the mailman.";
$new_subject = $regex->replace($replacement, $subject);
echo $new_subject . "\n"; // Output: The puppy chased the cat and the puppy bark

// Split a string by commas or spaces
$regex->setPattern("/[,\s]+/");
$subject = "red, green blue , yellow";
$array = $regex->split($subject);
print_r($array); // Output: Array ( [0] => red [1] => green [2] => blue [3] => ye

// Filter an array of email addresses by domain name
$regex->setPattern("/@gmail\.com$/");
$array = ["alice@gmail.com", "bob@yahoo.com", "charlie@hotmail.com", "david@gmai
$new_array = $regex->grep($array);
print_r($new_array); // Output: Array ( [0] => alice@gmail.com [3] => david@gmai
```

Examples of Regular Expression Usage in PHP

Here are some examples of how you can use regular expressions in PHP for various purposes:

Validating User Input

One common use case for regular expressions is to validate user input before processing it. For example, you can use regular expressions to check if an email address is valid, if a password meets certain criteria, if a phone number has a correct format, etc.

For example:

```
// Validate an email address
function validate_email($email) {
  // Define a regex for email format
  $regex = "/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/";
  // Return true if email matches regex, false otherwise
  return preg_match($regex, $email);
}

// Validate a password
function validate_password($password) {
  // Define a regex for password criteria
  // At least 8 characters long
  // At least one uppercase letter
  // At least one lowercase letter
  // At least one digit
  // At least one special character
  $regex = "/^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[!@#$%^&*]).{8,}$/";
  // Return true if password matches regex, false otherwise
  return preg_match($regex, $password);
}

// Validate a phone number
function validate_phone($phone) {
  // Define a regex for phone format
  // Optional country code (+ or 00 followed by 1 to 3 digits)
  // Optional space or dash separator
  // Area code (3 digits)
  // Optional space or dash separator
  // Local number (7 digits)
  $regex = "/^(\+|00\d{1,3})?[- ]?\d{3}[- ]?\d{7}$/";
  // Return true if phone matches regex, false otherwise
```

```
    return preg_match($regex, $phone);
  }
```

Extracting Data from Web Pages

Another common use case for regular expressions is to extract data from web pages or other sources of text. For example, you can use regular expressions to scrape information from HTML tags, parse JSON data, find URLs or email addresses in text, etc.

For example:

```
// Extract the title of a web page
function extract_title($html) {
  // Define a regex for the title tag
  $regex = "/<title>(.*?)<\/title>/";
  // Try to match the regex against the HTML string
  if (preg_match($regex, $html, $matches)) {
    // Return the first subpattern (the title text)
    return $matches[1];
  } else {
    // Return an empty string if no match found
    return "";
  }
}

// Extract the values of a JSON object
function extract_json_values($json) {
  // Define a regex for the key-value pairs in JSON format
  $regex = "/\"(\w+)\":\s*(\".*?\"|\d+|true|false|null)/";
  // Try to match the regex against the JSON string
  if (preg_match_all($regex, $json, $matches)) {
    // Return an associative array of keys and values
    return array_combine($matches[1], $matches[2]);
  } else {
    // Return an empty array if no match found
    return [];
  }
}

// Extract all URLs from a text
function extract_urls($text) {
  // Define a regex for the URL format
  $regex = "/https?:\/\/[\w\-\.]+(:\d+)?(\/[\w\-\.\/\?=&%#]*)?/";
  // Try to match the regex against the text string
```

```
    if (preg_match_all($regex, $text, $matches)) {
      // Return an array of URLs
      return $matches[0];
    } else {
      // Return an empty array if no match found
      return [];
    }
  }
```

Replacing Text in Files

A third common use case for regular expressions is to replace text in files or strings. For example, you can use regular expressions to perform search and replace operations, format text according to certain rules, generate new text from templates, etc.

For example:

```
  // Replace all occurrences of "foo" with "bar" in a file
  function replace_foo_with_bar($filename) {
    // Read the file contents into a string variable
    $content = file_get_contents($filename);
    // Define a regex for the word "foo"
    $regex = "/\bfoo\b/";
    // Replace all matches with "bar"
    $new_content = preg_replace($regex, "bar", $content);
    // Write the modified string back to the file
    file_put_contents($filename, $new_content);
  }

  // Format a date string in YYYY-MM-DD format
  function format_date($date) {
    // Define a regex for the date format
    // MM/DD/YYYY or M/D/YYYY
    $regex = "/(\d{1,2})\/(\d{1,2})\/(\d{4})/";
    // Replace the matches with YYYY-MM-DD format
    $new_date = preg_replace($regex, "$3-$1-$2", $date);
    // Return the formatted date
    return $new_date;
  }

  // Generate a greeting message from a template
  function generate_greeting($name) {
    // Define a regex for the placeholder {name}
    $regex = "/\{name\}/";
```

```php
    // Define a template for the greeting message
    $template = "Hello {name}, welcome to our site!";
    // Replace the placeholder with the name parameter
    $message = preg_replace($regex, $name, $template);
    // Return the generated message
    return $message;
  }
```

In this post, we have learned how to use regular expressions in PHP for pattern matching and data validation. We have covered the basics of regular expression syntax, how to create and use regular expression objects, and some common examples of regular expression usage in PHP.

Regular expressions are very useful and versatile tools for working with text data. They can help you perform various tasks such as validating user input, extracting data from web pages, replacing text in files, and more.

However, regular expressions can also be complex and tricky to write and debug. Therefore, it is important to test your regular expressions before using them in your code. You can use online tools such as Regex101 or RegExr to test and visualize your regular expressions.

I hope you have enjoyed this post and learned something new. If you have any questions or feedback, please leave a comment below. Thank you for reading!