

PHP Advanced Data Validation using Regular Expression Lecture-5

By: Ashiqullah Alizai
Herat University
Computer Science Faculty

Alizai.csf@hotmail.com

What is Regular Expression?

- A regular expression is a pattern that specifies a list of characters.
- Regular Expressions are a language of string patterns built into most modern programming languages, Perl, PHP, .NET and including Java 1.4 onward.
- A regular expression defines a search pattern for strings.
- Regular expressions can be used to search, edit and manipulate text.
- The abbreviation for Regular Expression is Regex.

Why Regular Expression

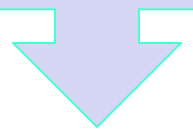
// a function that may use for email validation

```
function checkEmail($email){  
    $email = trim($email);  
    if (!checkLength($email,6)) {  
        return false;    }  
    elseif (!strpos($email,'@')) {  
        return false;    }  
    elseif (!strpos($email,'.')) {  
        return false;  
    }  
    elseif (strrpos($email,'.') < strpos($email,'@')) {  
        return false;  
    }  
    return true; }  

```

Why Regular Expression

We can use a regular expression to make this function both simpler and more powerful:



```
function checkEmail($email)
$emailPattern = '/^(\w+\.)*\w+ @ (\w+\.)+[A-Za-z]+$/' ;
    return preg_match($emailPattern,$email);
}
```

What is Regular Expression?

- PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.
- POSIX Regular Expressions
- PERL Style Regular Expressions

POSIX Regular Expressions:

- The structure of a POSIX regular expression is similar to that of a typical arithmetic expression:
- various elements (operators) are combined to form more complex expressions.
- Concepts being used in POSIX regular expression.
 - **Brackets:** Brackets ([]) are used to find a range of characters.
 - you could use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

Expression	Description
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

Quantifiers:

- The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and \$ flags all follow a character sequence.

Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing zero or more p's. This is just an alternative way to use p*.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p\$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Examples:

Expression	Description
<code>[^a-zA-Z]</code>	It matches any string not containing any of the characters ranging from a through z and A through Z.
<code>p.p</code>	It matches any string containing p, followed by any character, in turn followed by another p.
<code>^. {2}\$</code>	It matches any string containing exactly two characters.
<code>(.*)</code>	It matches any string enclosed within and .
<code>p(hp)*</code>	It matches any string containing a p followed by zero or more instances of the sequence hp.

Predefined Character Ranges

- For your programming convenience several predefined character ranges, also known as character classes, are available.
- Character classes specify an entire range of characters, for example, the alphabet or an integer set:

Expression	Description
<code>[:alpha:]</code>	It matches any string containing alphabetic characters aA through zZ.
<code>[:digit:]</code>	It matches any string containing numerical digits 0 through 9.
<code>[:alnum:]</code>	It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
<code>[:space:]</code>	It matches any string containing a space.

PHP's Regexp POSIX Functions

Function	Description
<u>ereg()</u>	The <code>ereg()</code> function searches a string specified by <code>string</code> for a string specified by <code>pattern</code> , returning <code>true</code> if the pattern is found, and <code>false</code> otherwise.
<u>ereg_replace()</u>	The <code>ereg_replace()</code> function searches for string specified by <code>pattern</code> and replaces <code>pattern</code> with <code>replacement</code> if found.
<u>eregi()</u>	The <code>eregi()</code> function searches throughout a string specified by <code>pattern</code> for a string specified by <code>string</code> . The search is not case sensitive.
<u>eregi_replace()</u>	The <code>eregi_replace()</code> function operates exactly like <code>ereg_replace()</code> , except that the search for <code>pattern</code> in <code>string</code> is not case sensitive.
<u>split()</u>	The <code>split()</code> function will divide a string into various elements, the boundaries of each element based on the occurrence of <code>pattern</code> in <code>string</code> .
<u>spliti()</u>	The <code>spliti()</code> function operates exactly in the same manner as its sibling <code>split()</code> , except that it is not case sensitive.
<u>sql_regcase()</u>	The <code>sql_regcase()</code> function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

PHP Function ereg()

- **Syntax:** `int ereg(string pattern, string originalstring, [array regs]);`
- **Definition and Usage:** The `ereg()` function searches a string compare with pattern
- The search is case sensitive in regard to alphabetical characters.
- The optional input parameter `regs` contains an array of all matched expressions that were grouped by parentheses in the regular expression.
- **Return Value:** It returns `true` if the pattern is found, and `false` otherwise.

PHP Function ereg(): Example

```
<?php

$email_id = "admin@tutorialspoint.com";
$retval = ereg("(\\.) (com$)", $email_id);
if( $retval == true )
{
    echo "Found a .com<br>";
}
else
{
    echo "Could not found a .com<br>";
}
$retval = ereg(("(\\.) (com$)"), $email_id, $regs);
if( $retval == true )
{
    echo "Found a .com and reg = ". $regs[0];
}
else
{
    echo "Could not found a .com";
}
?>
```

This will produce following result

```
Found a .com
Found a .com and reg = .com
```

PHP Function `ereg_replace()`

- **Syntax:** `string ereg_replace (string pattern, string replacement, string originalstring);`
- **Definition and Usage:** The `ereg_replace()` function searches for string specified by pattern and replaces pattern with replacement if found.
- Like `ereg()`, `ereg_replace()` is case sensitive.
- **Return Value:** After the replacement has occurred, the modified string will be returned. If no matches are found, the string will remain unchanged.

```
<?php  
  
$copy_date = "Copyright 1999";  
$copy_date = ereg_replace("([0-9]+)", "2000", $copy_date);  
print $copy_date;  
  
?>
```

This will produce following result

```
Copyright 2000
```

PHP Function eregi()

- **Syntax:** `eregi(string pattern, string string, [array regs]);`
- **Definition and Usage:** The `eregi()` function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive. `Eregi()` can be particularly useful when checking the validity of strings, such as passwords.
- **Return Value:** It returns true if the pattern is validated, and false otherwise.

```
<?php

$password = "abc";
if (! eregi ("[:alnum:]{8,10}", $password))
{
    print "Invalid password! Passwords must be from 8 - 10 chars";
}
else
{
    print "Valid password";
}
?>
```

This will produce following result

```
Invalid password! Passwords must be from 8 - 10 chars
```


PHP Function eregi_replace()

- **Syntax:** string eregi_replace (string pattern, string replacement, string originalstring);
- **Definition and Usage:** The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
- **Return Value:** After the replacement has occurred, the modified string will be returned.
 - If no matches are found, the string will remain unchanged.

```
<?php  
  
$copy_date = "Copyright 2000";  
$copy_date = eregi_replace("([a-z]+)", "&Copy;", $copy_date);  
print $copy_date;  
  
?>
```

This will produce following result

```
© 2000
```

PHP Function split()

- **Syntax:** array split (string pattern, string string [, int limit])
- **Definition and Usage:** The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
- The optional input parameter limit is used to signify the number of elements into which the string should be divided, starting from the left end of the string and working rightward.
- In cases where the pattern is an alphabetical character, split() is case sensitive.
- **Return Value:** Returns an array of strings after splitting up a string.
-

PHP Function split(): Example

```
<?php

$ip = "123.456.789.000"; // some IP address
$iparr = split ("\.", $ip);
print "$iparr[0] <br />";
print "$iparr[1] <br />" ;
print "$iparr[2] <br />" ;
print "$iparr[3] <br />" ;

?>
```

This will produce following result

```
123
456
789
000
```

PHP Function sql_regcase()

- **Syntax:** string sql_regcase (string string)
- **Definition and Usage:** The sql_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.
- If the alphabetical character has both an uppercase and a lowercase format, the bracket will contain both forms; otherwise the original character will be repeated twice.
- **Return Value:** Returns a string of bracketed expression along with converted character.

```
<?php  
  
$version = "php 4.0";  
print sql_regcase($version);  
  
?>
```

This will produce following result:

```
[Pp] [Hh] [Pp] [ ] [44] [..] [00]
```

PERL Style Regular Expressions:

- Perl-style regular expressions are similar to their POSIX counterparts.
- The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions.
- In fact, you can use any of the quantifiers introduced in the previous POSIX section.
- Lets give explanation for few concepts being used in PERL regular expressions.

Metacharacters

- A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.
- For instance, you can search for large money sums using the '\d' metacharacter: `/([\d]+)000/`, Here \d will search for any string of numerical character.

Character	Description
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo bar baz)	matches any of the alternatives specified

Meta Characters

Meta Characters	Explanation
\	Escape the next meta-character (it becomes a normal / literal character)
^	Match the beginning of the line
.	Match any character (except newline)
	Alternation for ('or' statement)
()	Grouping
[]	Custom character class
\$	Match the end of the line (or before newline at the end)

Example

- The pattern `^foo` can be found in "food", but not in "barfood".
- The pattern `foo$` can be found in "curfoo", but not in "food"
- The pattern `f[aeiou]d` can be found in "fad" and "fed", but not in "food", "faed" or "fd".
- The pattern `f[aeiou]{2}d` can be found in "faed" and "feod", but not in "fod", "fed" or "fd"
- The pattern `foo$|^bar` can be found in "foo" and "bar", but not "foobar".
- The pattern `fo\\.d` can be found in "fo.d", but not in "food" or "fo4d"

Example

- The pattern `fo.d` can be found in `"food"`, `"foad"`, `"fo9d"`, and `"fo*d"`.
- The pattern `fo\dd` can be found in `"fo1d"`, `"fo4d"` and `"fo0d"`, but not in `"food"` or `"fodd"`.
- The pattern `fo\Dd` can be found in `"food"` and `"foad"`, but not in `"fo4d"`.
- The pattern `fo\wd` can be found in `"food"`, `"fo_d"` and `"fo4d"`, but not in `"fo*d"`.
- The pattern `fo\Wd` can be found in `"fo*d"`, `"fo@d"` and `"fo.d"`, but not in `"food"`.
- The pattern `fo\sd` can be found in `"fo d"`, but not in `"food"`.
- The pattern `fo\Sd` can be found in `"fo*d"`, `"food"` and `"fo4d"`, but not in `"fo d"`.

Modifiers

- Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

Modifier	Description
i	Makes the match case insensitive
m	Specifies that if the string has newline or carriage return characters, the ^ and \$ operators will now match against a newline boundary, instead of a string boundary
o	Evaluates the expression only once
s	Allows use of . to match a newline character
x	Allows you to use white space in the expression for clarity
g	Globally finds all matches
cg	Allows a search to continue even after a global match fails

Quantifiers

Quantifiers	Explanation and alternatives
*	Match zero or more times, is an alternative for {0,}
+	Match one or more times, is an alternative for {1,}
?	Match no or one times, ? is an alternative for {0,1}
{n}	Match exactly n number of times
{n, }	Match at least n times,
{n,m}	Match at least n but not more than m times

Example

- The pattern `foo?` can be found in "food" and "fod", but not "faod".
- The pattern `fo+` can be found in "fod", "food" and "foood", but not "fd".
- The pattern `fo*d` can be found in "fd", "fod" and "food".
- The pattern `fo{3}d` can be found in "foood", but not "food" or "foood".
- The pattern `fo{2,4}d` can be found in "food", "foood" and "foood", but not "fod" or "fooooo".
- The pattern `fo{2,}d` can be found in "food" and "fooooo", but not "fod".



PHP's Regexp PERL Compatible Functions

PHP offers following functions for searching strings using Perl-compatible regular expressions:

Function	Description
<u>preg_match()</u>	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
<u>preg_match_all()</u>	The preg_match_all() function matches all occurrences of pattern in string.
<u>preg_replace()</u>	The preg_replace() function operates just like ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
<u>preg_split()</u>	The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
<u>preg_grep()</u>	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
<u>preg_quote()</u>	Quote regular expression characters

PHP Function preg_match()

- **Syntax:** `int preg_match (string pattern, string string [, array pattern_array], [, int $flags [, int $offset]])`;
- **Definition and Usage:** The `preg_match()` function searches string for pattern, returning true if pattern exists, and false otherwise.
- **Return Value:** Returns true if pattern exists, and false otherwise.

```
<?php

$line = "Vi is the greatest word processor ever created!";
// perform a case-Insensitive search for the word "Vi"
if (preg_match("/\bVi\b/i", $line, $match)) :
    print "Match found!";
endif;

?>
```

This will produce following result.

```
Match found!
```

PHP Function preg_match_all()

- **Syntax:** int preg_match_all (string pattern, string string, array pattern_array [, int order]);
- **Definition and Usage:** The preg_match_all() function matches all occurrences of pattern in string.
- **Return Value:** Returns the number of matching and Returns true if pattern exists, and false otherwise.

```
<?php

$userinfo = "Name: <b>John Poul</b> <br> Title: <b>PHP Guru</b>";
preg_match_all ("/<b>(.*?)</b>/U", $userinfo, $pat_array);
print $pat_array[0][0]." <br> ".$pat_array[0][1]."\n";

?>
```

This will produce following result.

```
John Poul
PHP Guru
```

PHP Function preg_replace()

- **Syntax:** mixed preg_replace (mixed pattern, mixed replacement, mixed string [, int limit [, int &\$count]]);
- **Definition and Usage:** The preg_replace() function operates just like POSIX function ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
- **Return Value:** After the replacement has occurred, the modified string will be returned.

```
<?php
$copy_date = "Copyright 1999";
$copy_date = preg_replace("([0-9]+)", "2000", $copy_date);
print $copy_date;
?>
```

This will produce following result.

```
Copyright 2000
```

PHP Function preg_split()

- **Syntax:** array preg_split (string pattern, string string [, int limit [, int flags]]);
- **Definition and Usage:** The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
- **Return Value:** Returns an array of strings after splitting up a string.

```
<?php
```

```
$ip = "123.456.789.000"; // some IP address  
$iparr = split ("/\\./", $ip);  
print "$iparr[0] <br />";  
print "$iparr[1] <br />" ;  
print "$iparr[2] <br />" ;  
print "$iparr[3] <br />" ;  
  
?>
```

This will produce following result.

```
123  
456  
789  
000
```

PHP Function preg_grep()

- **Syntax:** array preg_grep (string \$pattern, array \$input [, int \$flags]);
- **Definition and Usage:** Returns the array consisting of the elements of the input array that match the given pattern.
- **Return Value:** Returns an array indexed using the keys from the input array.

```
<?php

$foods = array("pasta", "steak", "fish", "potatoes");
// find elements beginning with "p", followed by one or more letters.
$p_foods = preg_grep("/p(\w+)/", $foods);
print "Found food is " . $p_foods[0];
print "Found food is " . $p_foods[1];

?>
```

This will produce following result.

```
Found food is pasta
Found food is potatoes
```

Note: If flag is set to **PREG_GREP_INVERT**, this function returns the elements of the input array that do not match the given pattern.

PHP Function preg_quote()

- **Syntax:** string preg_quote (string \$str [, string \$delimiter]);
- **Definition and Usage:** preg_quote() takes str and puts a backslash in front of every character that is part of the regular expression syntax.
- **Return Value:** Returns the quoted string.

```
<?php  
  
$keywords = '$40 for a g3/400';  
$keywords = preg_quote($keywords, '/');  
echo $keywords;  
  
?>
```

This will produce following result.

```
\$40 for a g3\/400
```

This is the end for this lecture



WEB_3

