Regular expressions in PHP

Outline

- Presentation
- PCRE functions
- Searching for a string
- The beginning and the end of a string
- The quantifiers
- Character classes
- `preg_match` , `preg_match_all`, `preg_replace` and `preg_quote` functions

Outline

- **Presentation**
- PCRE functions
- Searching for a string
- The beginning and the end of a string
- The quantifiers
- Character classes
- `preg_match` , `preg_match_all`, `preg_replace` and `preg_quote` functions

Presentation

- regular expression or REGEX
  - a string that describes a string pattern
  - model described using: operators, values, variables
  - options available to simplify writing
- REGEX utility
  - MATCH: check that a string matches a given pattern: REGEX
    - Does my string start / end with ...?
    - Does my string contain / do not contain ...?
    - Does my string have the syntax of an e-mail / web address?
  - EXTRACT: extract part of a string (= a group)
    - I want the parameters param1 and param3 of the URL ...
    - Parsing a web page
  - SUBSTITUTE: replace part of a character string with another
    - Massively change the format of a CSV, SQL file ...

Presentation

- Which tools use REGEX?
  - Most programming languages
    - Java, PHP, Perl, Javascript, ...
    - also MySQL, HTML5
  - Most text editors
    - Eclipse, Notepad ++, vi, ...

Outline

- Presentation
- **PCRE functions**
- Searching for a string
- The beginning and the end of a string
- The quantifiers
- Character classes
- `preg_match` , `preg_match_all`, `preg_replace` and `preg_quote` functions

PCRE functions

- In PHP, there are 2 "languages" for regex
  - POSIX
  - PCRE
    - *Perl Compatible Regular Expressions*
    - → more powerful (power, speed of treatment)
- functions using PCRE start with `preg_`
  - `preg_match_all`
  - `preg_match`
  - `preg_replace`
  - `preg_split`
  - `etc`

PCRE functions

- Replacement or comparison functions:
  - `preg_X ('mask', 'replacement string', 'string of characters in which the mask will be applied');`
- The chain "mask"
  - regular expression
- regex
  - is always delimited by two identical characters called delimiters
    - ``
    - `##`
  - delimiters → allow to add options (insensitivity to upper / lower case)
    - "`blabla`i"

Example

- here is the general example of using a regex:
  - `mask`options
- which can give for example:
  - **`[a-z]*`i**
  - a string consisting of only uppercase or lowercase letters
  - the **i** in the option means that the search is insensitive to lowercase or uppercase

Flags

- are the options of the regular expression
  - `c`: in case an error occurs, do not reset the search position
  - g: search globally, that is find all occurrences
  - i: ignore the case
  - m: multiple lines, the string can have multiple lines
  - o: ounce, apply the search pattern only once
  - s: so that the "dot" also works for line breaks (so that the text can be bold in several lines)
  - x: extended, use extended syntax
- Warning! : all these flags are not available in all languages, the most used are: `g`, `i`, `m`

Outline

- Presentation
- PCRE functions
- **Searching for a string**
- The beginning and the end of a string
- The quantifiers
- Character classes
- `preg_match` , `preg_match_all`, `preg_replace` and `preg_quote` functions

Searching for a string

- `strpos()` function of PHP
  - search for a simple string of characters
  - much faster than a regex
  - very limited features because it can only have a fixed string parameter
- regex
  - ability to impose effective conditions on the search query
- `preg_match()` function
  - search for a string according to specific conditions

strpos example

```
<?php
$mystring = 'An example of a simple string';
$findme   = 'example';
$pos = strpos($mystring, $findme);
if ($pos === false) {
    echo "$findme not found in $mystring";
} else {
    echo "$findme' found in $mystring";
    echo " at position $pos";
}
?>
```

preg_match() example

```
<?php
  $mystring =  'An example of a simple string';
  $findme='`example`';
     if(preg_match($findme, $mystring))
          echo 'True, the string matches the mask';
     else
          echo 'False, the string does not match the mask';
?>
```

- the string matches the mask because
  - we are looking for the "example" string that fits well in the sentence "An example of a simple string".
- `preg_match()` functions are not limited to that, otherwise use `strpos()` ...

Other example

```
<?php
  $mystring =  'An example of a simple string';
  $findme='`Example`';
     if(preg_match($findme, $mystring))
          echo 'True, the string matches the mask';
     else
          echo 'False, the string does not match the mask';
?>
```

- false result because the example E is uppercase
- To fix this problem,
  - we use the option "i" which means that the mask does not take into account the small and capital letters.
  - the options are placed after the second delimiter (here the character `)
    - `$findme='`Exemple`i';`

preg_match() function

String	Regex	Result
'An example of a simple string'	`example`	True (the word "example" is in the string)
'An example of a simple string'	`complicated`	False (the word "complicated" is not in the string)
'An example of a simple string'	`Example`	False because the E is capitalized in the regex and not in the string
'An example of a simple string'	`Example`i	True because the `i` makes case insensitive (upper / lower case) the regex

Searching for a string: "or"

- The symbol "or" is designated in PCRE by a single vertical bar |
  - unlike PHP which asks for 2 | |
- example
  - Let's say that we were looking in the string for the word "site" or the word "web".
  - `site|web`

Searching for a string: "or"

String	Regex	Result
'Welcome to my website'	`site|web`	True (we have either site or web in the chain)
'WELCOME TO MY WEBSITE'	`site|web`	False (neither word is lowercase)
'WELCOME TO MY WEBSITE'	`site|web`i	True, we made the regular expression case-insensitive
'Welcome to my web space'	`site|web`	True because the word "web" is in the chain to analyze
'Welcome to my space'	`site|web`	False because neither the word "site" nor the word "web" appear in the chain

Outline

- Presentation
- PCRE functions
- Searching for a string
- **The beginning and the end of a string**
- The quantifiers
- Character classes
- `preg_match` , `preg_match_all`, `preg_replace` and `preg_quote` functions

Limit characters

- example
  - to say "does this string begin with the capital letter A and end with a lowercase b?
- symbols
  - ^ which denotes the beginning of a string
  - $ which denotes the end of a string
  - \b characterizes a word break (characters at the beginning or end of a word)
  - \B characterizes the characters inside a word (neither at the beginning nor at the end)

The beginning and the end of a string

String	Regex	Result
'Ali'	`^A`	True (the string starts with an A)
'Ali'	`i$`	True (the string ends with an i)
'Date me'	`^Dt$`	False because the string is different from "Dt"
'Welcome to my web space'	`^Welcome to my web space$`	True because the string is identical on both sides
‘Welcome to my web space'	`^Welcome space$`	False because there are words between the word "Welcome" and the word “space"
eurelis	`\beur`	True (Word starting with eur)
agitateur	`\beur`	False
agitateur	`eur\b`	True (word ending in eur)
eurelis	`eur\b`	False
git OR ci	`\bi`	False (word that does not start with i)
git	`i\B`	True (word that does not end with i)
ci	`i\B`	False

Outline

- Presentation
- PCRE functions
- Searching for a string
- The beginning and the end of a string
- **The quantifiers**
- Character classes
- `preg_match` , `preg_match_all`, `preg_replace` and `preg_quote` functions

Quantifiers

- example
  - say "I want to know if this string starts with 7 times the letter A", or
  - "I need an A, three o and two z".
- First we must remember three symbols:
  - ? which denotes "0 or 1 occurrence".
  - + which designates "1 or more occurrences".
  - * which denotes "0, 1 or more occurrences".
- These symbols apply only to the letter (or word if you place parentheses) that is located to the left of the symbol.

Quantifiers

- ? which denotes "0 or 1 occurrence".
  - mask `v?` → string 'v' will be valid, 'vv' will not be valid and ‘ ‘ will be valid
- + which denotes "1 or more occurrences".
  - mask `v+` → the string 'v' will be valid, 'vv' will be, 'vvvvvv' too, but ‘ ‘ will not be valid.
- * which denotes "0, 1 or more occurrences".
  - mask `v*` → the string 'v' will be valid, 'vv' will be, 'vvvvvv' as well as ‘ ‘

Quantifiers

String	Regex	Result
'Anthony'	`An?thony`	True (There are 0 or 1 "n" after the A)
'Annnnnnthony'	`An?thony`	False (there is more than 1 "n" after the A)
'Annnnnnthony'	`An+thony`	True (there is at least 1 "n" after A)
'Cat'	`Cats?$`	True because the word "cat" can also be here plural (0 or 1 "s"), then we mark the end of the chain
'Cat'	`Cats*$`	True because * means "0", "1" or "many" here there are no s, the condition is still true.
'Cat'	`Cats+$`	False

Quantifiers

- another way to quantify → accurate
- to quantify a string or a letter, or a class of characters
- use braces {}
- {X}: The string must be repeated X times.
- {X, Y}: The string can be repeated from X times to Y times.
- {X,}: The string must be repeated at least X times but there is no number limit.

Quantifiers

- `A{2}` will work for AA but not for AAA or A.
- `A{2,4}` will work for AA, for AAA and for AAAA but not for A or for AAAAA.
- `A{2,}` will work for AA, for AAA, for AAAA … but not for A.

String	Regex	Result
'Anthony'	`An{0,1}thony`	True (There are 0 or 1 "n" after the A)
'Annnnnnthony'	`An{2}thony`	False (there are not two "n" after the A)
'Annnnnnthony'	`An{2,}thony`	True (there is at least 2 "n" after the A)

Outline

- Presentation
- PCRE functions
- Searching for a string
- The beginning and the end of a string
- The quantifiers
- **Character classes**
- `preg_match` , `preg_match_all`, `preg_replace` and `preg_quote` functions

Character classes

- perform accurate searches while limiting the number of regex characters
- always enclosed in single [] or double [[]] (special character classes) brackets.
- Here's an example of a regular expression that includes a character class: `[nv]ous`
- the [nv] means "n" OR "v".
- The words “nous" and “vous" will therefore be accepted, so it's the equivalent of `(n|v)ous`

String	Regex	Result
'Nous'	`[nv]ous`i	True (the i makes the regex insensitive to the case)
'Nous'	`[vs]$`	True (the string ends well with a v or a s)

Special Classes

- dash between multiple values → character intervals
- examples
  - `[a-z]` : indicates a lowercase letter from a to z
  - `[0-3]` : indicates a number from 0 to 3
  - `[a-z0-9]` : indicates lowercase letters OR numbers from 0 to 9

String	Regex	Result
'Anthony'	`[a-z]*`	False
'Annnnnnthony'	`[a-z]*`i	True (all characters are lowercase or uppercase letters)
'Annnnnnthony1'	`[0-9]$`	True (the string ends well with a number from 0 to 9)

Character classes

- prohibit one or a range of characters
- symbol ^ placed just after the first opening [
- It no longer means "beginning of the chain" but a ban.

String	Regex	Result
'ANTHONY'	`^[^a-z]*$`	True (the string does not contain lowercase letters)
'Anthony'	`^[^a-z]*$`	False (the string contains lowercase letters)

Special Classes

- double brackets → predefined to simplify the syntax of your regex

`[[:alpha:]]`	any letter
`[[:lower:]]`	any letter in lowercase
`[[:upper:]]`	any letter in uppercase
`[[:alnum:]]`	any letter or number
`[[:digit:]]`	any number
`[[:punct:]]`	any punctuation
`[[:space:]]`	any white space
`[[:blank:]]`	space or tabulation
`[[:graph:]]`	printable characters
`[[:cntrl:]]`	escape characters
`[[:xdigit:]]`	hexadecimal characters
`[[:print:]]`	printable characters except those of control

Metacharacters

- special characters performing specific actions on strings
- list of these character:
  - [ ] ! ( ) {} ^ $? . + * \ #
- problem → You must precede them with a backslash \

String	Regex	Result
'ça va?'	`ça va?$`	False (the string does not end with 0 or 1 "a")
'ça va?'	`ça va\?$`	True (we escaped the character correctly)

- Attention
  - this rule does not apply inside character classes
  - the class [ab+?] means that you can have the letter "a" or "b" or "+" or "?"

Short classes

- reduce the length of the regex in disregard of their direct understanding
- backslash followed by a character → more complex character class

. → any character.
\w → words [_a-zA-Z0-9]
\W → which does not correspond to a word [^_a-zA-Z0-9]
\d → one digit [0-9]
\D → no numbers [^0-9]
\s → space (corresponds to \t \n \r)
\S → not a space (\t \n \r)
\t → tabulation
\n → line break
\r → carriage return

Outline

- Presentation
- PCRE functions
- Searching for a string
- The beginning and the end of a string
- The quantifiers
- Character classes
- **`preg_match , preg_match_all, preg_replace and preg_quote functions`**

`preg_match` function

- The syntax :
  - `preg_match (string pattern, string subject , [array result ]);`
- preg_match returns true if the subject string contains the given pattern
- If the *result array* option is provided, preg_match fills an array with matches.
  - $*result*, will contain the results of the search:
  - $*result*[0] will contain the part that satisfied the complete mask
  - $*result*[1] will contain the part that satisfied the first capturing parenthesis and so on ...

Example 1

- Test on a login field *letters digits* only

```
$login="Toto1508";
if(!preg_match('`^[[:alnum:]]{4,8}$`',$login))  {
    echo "Wrong login"; }
else {  echo "Correct Login";  }
```

- The login can only contain letters and numbers and must be from 4 characters to 8 characters maximum.
- `[[:alnum:]]` class determining the exclusive use of letters and numbers.
- `{4,8}` is the recognition interval with a minimum of 4 characters and a maximum of 8 characters.

Example 2

- Same as the previous + underscore using the predefined class \w

```
$login="Toto_1508";
if(!preg_match('`^\w{4,8}$`',$login)) {
    echo "Wrong login"; }
else {  echo "Correct Login";  }
```

- The login can only contain letters and numbers, as well as the inderscore and must be from 4 characters to 8 characters maximum.
- \w predefined class used with PCREs and determining the exclusive use of letters and numbers, as well as underscores.
- {4,8} is the recognition interval with a minimum of 4 characters and a maximum of 8 characters.

Test on an email

```
//--- example with the predefined classes
$r='`^[[:alnum:]]([-_.]?[[:alnum:]])*@[[:alnum:]]([-_.]?[[:alnum:]])*\.([a-z]{2,4})$`';
preg_match($r,$string);

//--- example with the predefined classes
$r='`^\w([-_.]?\w)*@\w([-_.]?\w)*\.([a-z]{2,4})$`';
preg_match($r,$string)
```

Test on an email

```
$r='`^[[:alnum:]]([-_.]?[[:alnum:]])*@[[:alnum:]]([-_.]?[[:alnum:]])*\.([a-z]{2,4})$`';
```

- email must begin with a letter or number [[:alnum:]] then
- can be followed by a single - or a single _ or even a single point.
- followed by a series of letters or numbers ([-_.]?[[:alnum:]])*
- Then comes @ followed only by one or more letters or numbers [[:alnum:]]
- then can be followed by a single - or a single _ or even a single point. ,
- followed by a series of letters or numbers ([-_.]? [[:alnum:]])*
- Finally comes the point. followed by 2 to 4 letters for domain \.([a-z]{2,4})
- You will notice the \ in front of the point which makes it possible to escape it so that it is no longer regarded as a metacharacter but as the sign. (point)

preg_match_all function

- The syntax :
  - preg_match_all (string pattern, string subject, array result, optional parameter);
- preg_match_all returns in a table all occurrences of the pattern found in the subject string.
- An order can be sent by the optional parameter
- After finding a first result, the search continues to the end of the chain.
- order may take PREG_PATTERN_ORDER
  - The order is such that $matches[0] is a table that contains results that satisfy the full mask
  - $matches[1] is a table that contains the results that satisfy the first capturing parenthesis, etc.

Example 1

- Extract phone numbers from a string

```
$string="Hello, please call 01-99-99-55.55 or 0033 1 75-75-75-
75 or more 0495/999.999 also 0032 99.99.99.99 as soon as
possible";

$pattern='`(\d+\s?[./-]?)+`';

preg_match_all($pattern,$string,$out);

$nb=count($out[0]);

for($i=0;$i<$nb;$i++) {  echo $out[0][$i].' ';  }
```

Example 1

- example:
  - extract a series of phones numbers placed in a text string.
- preg_match_all finds the first occurrence and starts again till the end of the chain.
- The pattern takes into account separators such as the point. the dash - and the slash /
- we take into account the numeric characters with \d
- \s? means that the digits can be followed by a space. (optional)
- [./-]? this allows a separator between the digits. (optional)

Example 2

- Extract images from personal markup (bbcode)

```
<?php
$ch='text [image]image1[/image] text [image]image2[/image] text
[image]image3[/image] text';

preg_match_all('`\[image]([^[]+)\[/image]`',$ch,$out);

echo '<pre>';
print_r($out);
echo '</pre>';
?>
```

Example 2

```
Array
  (
        [0] => Array
        (
                [0] => [image]image1[/image]
                [1] => [image]image2[/image]
                [2] => [image]image3[/image]
        )

        [1] => Array
        (
                [0] => image1
                [1] => image2
                [2] => image3
        )
  )
```

`preg_replace` function

- The syntax :
  - `preg_replace (string pattern, new pattern, string  subject, int limit);`
- preg_replace replaces the string pattern with the new pattern and returns the modified string.
- If int limit is set, the number of replacements will be limited
- replacement can contain references of the form \\ n or, since PHP 4.0.4 $n.
- These references will be replaced by the text captured by the n'-th capturing parenthesis of the mask.
  - n can take values from 0 to 99, and
  - \\0 or $0, correspond to the text of which satisfies the complete mask.
- Opening parentheses are counted from left to right (starting at 1) to determine the capturing parenthesis number.

Example 1

- Remove non useful

```
<?php
$str="hello, this is a text with break line<br /><br /><br /><br /><br /><br
/> and with some unnecessary break line<br/><br /><br /><br /><br />";
//--- echo to see
echo trim($str).'<br/>This is not good !';

$str=preg_replace('`[<br\s?/>]{2,}`','<br/><br/> ',$str);
echo '<hr>'.trim($str).'This is better, no ?';
?>
```

- Very simple … we will focus on removing the series of more than two line breaks so as to make the layout more correct.

Example 2

- Make clickable a link between < and >

```
<?php
$strA="Test an url placed between < and > <https://www.europeancards.com>
opening and closing tags ";

//--- 1st version ---//
$r='`<([^>]+)>`';
$str=preg_replace($r, '<a href="$1" target="_blank">$1</a>', $strA);
echo $str;

//--- 2nd version ---//
echo '<br/>The same with https:// in the mask<br/>';
$r='`<(https?://[^>]+)>`';
$str1=preg_replace($r, '<a href="$1" target="_blank">$1</a>', $strA);
?>
```

Example 2

- Very simple and widespread on most forums, etc ...
- In the first we match everything between <and>.
- In the second version, we take into account the part http:// or https://
- Without it, the text will not be recognized as a link.

`preg_quote` function (from php3)

- The syntax :
  - `preg_quote (string subject, delimiter element);`
- preg_quote adds a backslash to any special characters in the subject string.
- This function is very useful if the string generated during the execution must be used in a mask.
- If the optional delimiter argument is provided, it will also be escaped.
- This is convenient for escaping the delimiter required by the PCRE functions.
- The slash / is the most common delimiter.
- Special characters that will be escaped: . \\ + * ? [^] $ ( ) { } = ! < > | :

References

- http://www.php.net/manual/en/book.pcre.php