Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Web Programming
7) PHP, Reg Exp

Emmanuel Benoist

Fall Term 2013-14

Computer Science Division

PHP Regular Expressions

- Motivation
- RegExp syntax
  - Searching for a pattern
  - Special chars
  - Sets
  - Repetitions
  - Accessing the matched content
  - Matching subexpressions
- PHP functions using RegExp
  - `preg_match_all()`
  - `preg_replace()`
  - `preg_split()`
  - `preg_quote()`
  - `preg_grep()`
- Examples

Motivation

Regular Expressions

What to do?

- Search an expression
- Replace
- Parse a file (rudimentely)
  - File parsing: log file

::1 − − [02/Dec/2010:16:31:22 +0100] "GET␣/coursAWT/↘
→examples/webSecurity/xss/helloBoss.phps␣HTTP/1.1" 200 937

RegExp syntax

Searching for a pattern

The search function `preg_match()`

- What we need is a pattern:

```
$pattern ='/toto/';
$s1 = 'hello toto';
preg_match($pattern, $s1); // returns true
$s2 = 'hello toto and titi';
preg_match($pattern,$s2); // returns true also
```

Examples

```
// the "i" after the pattern delimiter indicates
// a case-insensitive search
if (preg_match ("/php/i",
                    "PHP est le meilleur langage pour le web.")) ↘
                    →{
    print "A match was found.";
} else {
    print "A match was not found.";
}
```

Examples (Cont.)

```
// the \b in the pattern indicates a word boundary,
// so only the distinct word "web" is matched,
// and not a word partial like "webbing" or "cobweb"
if (preg_match ("/\bweb\b/i",
                    "PHP a the web scripting language")) {
    print "A match was found.";
} else {
    print "A match was not found.";
}
if (preg_match ("/\bweb\b/i",
                    "PHP is a website scripting language")) {
    print "A match was found.";
} else {
    print "A match was not found.";
}
```

RegExp Syntax: Subexp / Or

- Subexpression = ()

```
$p = '/cat(erpilar|aract)/';
preg_match($p,'cataract'); // true
preg_match($p,'caterpilar'); // true;
preg_match($p,'cat'); // false
$p2 = '/cat(erpilar|aract|)/';
preg_match($p2,'cat'); // true
```

Special chars

Special chars

- Some chars have a special meaning:

. = any character	$\backslash b$ = word boundary
$\backslash w$ = any letter of digit	$\backslash W$ = anything not in $\backslash w$
$\backslash d$ = any letter of digit	$\backslash D$ = anything not in $\backslash d$

```
$p='/c\wt/';
preg_match($p,'hello cat');// returns true
preg_match($p,'cot');// returns true
preg_match($p,'caterpilar');// returns true
preg_match($p,'Le duc travaille');// returns false
preg_match($p,'or donc,tu viens');// returns false
```

Spaces

' ' = one space char	\t = tabular
\r = carriage return	\n = line feed
\s = [\t\r\n] any space	
^ = Start of a string	$ = End of a string

```
$p='/^$/'; // matches only the empty string
$p2='/^cat$/';
preg_match($p2,'caterpilar'); // returns false
preg_match($p2,'cat'); // returns true
$p3='/c\st/';
preg_match($p3,'Le duc travaille');// returns true
```

Sets

Sets

- `[]` Represents character sets
- `[12]` 1 or 2
- `[1234567]` = `[1-7]` 1 to 7
- `\d` = `[0-9]`
- `\w [0-9a-zA-Z]`
- `/c[au]t/` matches both 'cat' and 'cut'

Negative sets

- `[^5]` any char not 5
- `[^\w]` = `\W`

Repetitions

Repetitions

How to match a year? 2010 is a repetition of four digits

- First attempt: `\d\d\d\d` (four times `\d`)
- Repetition: `\d{4}` (four times `\d`)
- `(19|20)(\d{2})` is a good regexp for testing a year of birth

Other repetitions

- `\d{2,4}` A positive integer with 2, 3 or 4 digits
- `\d{2,}` A number composed of a least two digits
- `\d{,3}` At most 3 digits
- `\d{1,}` At least one digit
- can be written: `\d+`

Repetitions (Cont.)

- `\d{0,}` Any repetition, at least 0 time
- `\d*` idem (i.e. can contain 0 to any number of digits)
- `\d?` = `\d{0,1}` one or zero time the digit

```
$p='/-?\d+/';
echo preg_match($p,'100'); // writes 1
echo preg_match($p,'100'); // writes 1
echo preg_match($p,'100.10'); // writes 1
echo preg_match($p,'-100'); // writes 1
echo preg_match($p,'toto'); //writes 0
echo preg_match($p,'toto 50'); //writes 1
```

Accessing the matched content

What has been found?

```
$p='/-?\d+/';
echo preg_match($p,'toto 50'); //writes 1
```

- We want to find the number not 1.
- `preg_match` takes another argument: the array of matches
- Each subexpression () is written in this array
- Index 0 corresponds to the whole match (the regexp itself)
- Index *i* corresponds to subexpression number *i*

```
$p='/-?\d+/';
preg_match($p,'toto 50',$a);
echo $a[0]; // writes 50
preg_match($p,'toto -3050',$b);
echo $b[0]; // writes -3050
```

Matching subexpressions

Matching of subexpressions

- The result array contains all the subexpressions (ranked by starting open parenthesis)

```
$patternProf = '/([a-zA-Z]{3}\d{1,2})@bfh.ch/';
$address = 'bie1@bfh.ch';
preg_match($patternProf,$address,$res);
echo "email=" .$res[0];
echo "kuerzel=" .$res[1];
```

Subexpressions (Cont.)

```
$log= '192.168.1.10 - - [02/Dec/2010:16:31:22 +0100] "↘
→GET /coursAWT/hello.php HTTP/1.1" 200 937 ';
$pattern='/([\d\.]+)-(.*)- \[(.*)\] "(GET|POST)/'
preg_match($pattern,$log,$res);
foreach($res as $item){
 echo $item."<br>\n"
}
```

Read a File

Example

```
<?
// get a web page into an array
//$fcontents = file ('http://www.libe.com');

// Read a File from the local disk into an array
$fcontents = file ('testFile.php');

// Print out the array
foreach ($fcontents as $line_num => $line) {
    echo "<b>Line $line_num:</b> " .
        htmlspecialchars ($line) . "<br>\n";
}
?>
```

PHP functions using RegExp

PHP functions using RegExp

- **preg_match()**
- **preg_match_all()** Correspondance Operator
- **preg_replace()** Substitution Operator
- **preg_split()** Split Operator
- **preg_quote()** Quote regular expression characters

preg_match_all()

preg_match_all()

Perform a global regular expression match

int preg_match_all (string pattern, string subject,
array matches [, int order])

Searches subject for all matches to the regular expression given in pattern and puts them in matches in the order specified by order. After the first match is found, the subsequent searches are continued on from end of the last match.

preg_match_all() (Cont.)

Order can be one of two things

- **PREG_PATTERN_ORDER**
  Orders results so that `$matches[0]` is an array of full pattern matches, `$matches[1]` is an array of strings matched by the first parenthesized subpattern, and so on.

```
preg_match_all ("|<[^>]+>(.*)</[^>]+>|U",
  "<b>example: </b><div align=left>this is a test</div>",
  $out, PREG_PATTERN_ORDER);
print $out[0][0]." , ".$out[0][1]."\n";
print $out[1][0]." , ".$out[1][1]."\n";

# Output
# <b>example: </b>, <div align=left>this is a test</div>
# example: , this is a test
```

preg_match_all() (Cont.)

- **PREG_SET_ORDER**
  Orders results so that `$matches[0]` is an array of first set of matches, `$matches[1]` is an array of second set of matches, and so on.

```
preg_match_all ("|<[^>]+>(.*)</[^>]+>|U",
    "<b>example: </b><div align=left>this is a test</↘
    →div>",
     $out, PREG_SET_ORDER);
print $out[0][0]." , ".$out[0][1]."\n";
print $out[1][0]." , ".$out[1][1]."\n";

# Output
# <b>example: </b>, example:
# <div align=left>this is a test</div>, this is a test
```

preg_replace()

preg_replace()

Perform a regular expression search and replace

mixed preg_replace (mixed pattern, mixed replacement,
mixed subject [, int limit])

Searches subject for matches to pattern and replaces them with replacement . If limit is specified, then only limit matches will be replaced; if limit is omitted or is -1, then all matches are replaced.

Example 1

```
$patterns = array ("/(19|20)(\d{2})-(\d{1,2})-(\d{1,2})/",
                   "/^\s*{(\w+)}\s*=/");
// Use a backward reference on the regExp \1 = first ()
$replace = array ("\\3/\\4/\\1\\2", "$\\1 =");
print preg_replace ($patterns, $replace,
                    "{startDate} = 1999-5-27");

# Output
# $startDate = 5/27/1999
```

preg_replace() (Cont.)

Example 2 (Using /e modifier)
This would capitalize all HTML tags in the input text.

```
preg_replace ("/(<\/?)(\w+)([^>]*>)/e",
                "'\\1'.strtoupper('\\2').'\\3'",
              $html_body);
```

preg_split()

preg_split()

Split string by a regular expression

Returns an array containing substrings of subject split along boundaries matched by pattern.

Examples

```
// Get the parts of a search string.
// split the phrase by any number of commas or space ↘
→characters,
$keywords = preg_split ("/[\s,]+/",
                                "hypertext language, programming");
// Splitting a string into component characters.
$str = 'string';
$chars = preg_split('//', $str, 0, PREG_SPLIT_NO_EMPTY);
print_r($chars);
```

preg_quote()

preg_quote()

Quote regular expression characters

string preg_quote (string str [, string delimiter])

preg_quote() takes str and puts a backslash in front of every character that is part of the regular expression syntax.

```
// In this example, preg_quote($word) is used to keep the
// asterisks from having special meaning to the regular
// expression.
$textbody = "This book is *very* difficult to find.";
$word = "*very*";
$textbody = preg_replace ("/".preg_quote($word)."/",
                          "<i>".$word."</i>",
                          $textbody);
```

preg_grep()

preg_grep()

Return array entries that match the pattern

array preg_grep (string pattern, array input)

preg_grep() returns the array consisting of the elements of the input array that match the given pattern.

```
// return all array elements
// containing floating point numbers
$fl_array = preg_grep ("/^(\d+)?\.\d+$/", $array);
```

Examples

Examples

Usefull regular Expressions

```
# match any empty line
/^$/
# match any email address
/\w+\.?\w*\@(\w+|\.){1,3}\w{2,3}/
# match any line with at least 80 characters
/.{80,}/
```

Pattern repetition

```
foreach($fcontents as $line_num => $line) {
    if(preg_match(
          "/\"(GET|POST) ([^\?]*)\??(.*)? (HTTP\/\d\.\d)\"/i",
          $line, $matches)){
        print "URL = $matches[2]; Param = $matches[3]; ".
              "Protocol = $matches[4]<br>\n";

        if(preg_match("/[\w_-~\/\.]*(html|jsp|htm|php)/i",
                      $matches[2],$urlContent)){
            print "Suffix = $urlContent[1]<br>\n";
            $suffixes[$urlContent[1]]++;
        }
        // We want to split the URL and get its suffix
        $url_content = preg_split("/[\.\/]/",$matches[2]);
        $suffix = $url_content[count($url_content)-1];
        print "Suffix = $suffix<br>\n";
        $suffixesBis[$suffix]++;
    }
}
```

Example

```
$ok_html = "I <b>love</b> shrimps dumplings.";
$bad_html = "I <b>love</i> shrimps dumplings.";
if (preg_match('@<([bi])>.*?</\1>@', $ok_html)) {
    print ("Good for you! (OK; Backreferences)\n");
}
if (preg_match('@<([bi])>.*?</\1>@', $bad_html)) {
    print ("Good for you! (BAD; Backreferences)\n");
}
if (preg_match('@<[bi]>.*?</[bi]@', $ok_html)) {
    print ("Good for you! (OK; No Backreferences)\n");
}
if (preg_match('@<[bi]>.*?</[bi]@', $bad_html)) {
    print ("Good for you! (BAD; No Backreferences)\n");
}
```

Example

This example prints:

Good for you! (OK; Backreferences)
Good for you! (OK; No Backreferences)
Good for you! (BAD; No Backreferences)

Example

```
$members=<<<TEXT

Name E-Mail Address<br>
-------------------------------------------------------------------<br>
Inky T. Ghost inky@pacman.example.com<br>
Donkey K. Gorilla kong@banana.example.com<br>
Mario A. Plumber mario@franchise.example.org<br>
TEXT;

print preg_replace('/([^@\s]+)@(([-a-z0-9]+\.)+[a-z]{2,})/',
                   '\\1 at \\2', $members);
```

This examples prints

```
 Name E-Mail Address
-------------------------------------------------------------------
 Inky T. Ghost inky at pacman.example.com
 Donkey K. Gorilla kong at banana.example.com
 Mario A. Plumber mario at franchise.example.org
```

Split

Send the content of a String into an Array

```
$line = "142.34.123.12";
@address= split /\./ , $line;
foreach $l(@address){
   print $l." ";
}
print "\n";
# Output:
# 142 34 123 12

$line = 'emmanuel.benoist@bfh.ch';
@address= split /[\.|\@]/ , $line;
foreach $l(@address){
   print $l." ";
}
print "\n";
# Output:
# emmanuel benoist bfh ch
```

Conclusion

- **Regular Expressions are much more than what I explained**
  - Theory of Regular Languages, together with Automata
  - Part of a course about Theoretical Computer Science
- **Reg Exp can be useful even without theory**
  - One can use it,
  - Guess and try if it works
- **RegExp are widely used in scripting languages**
  - For manipulating data, content of files, ...
- **Reg Exp can hardly be seen as part of the Web Technology**
  - But there is nowhere else for this stuff
  - It is part of the fundamental knowledge of CS students.