CM0133 Internet Computing

7 More PHP
String Manipulation & Regular Expressions

Objectives

- Variables and Memory
- String manipulation
- What are regular expressions
- What can regular expressions be used for
- Why regular expressions can seem daunting

Automatic Type Conversion

```
$mystring = "12";
$myinteger = 20;
print $mystring + $myinteger;
```

Will result in 32, despite the fact that one variable is a string and one is an integer.

```
$bool = true;
print "Bool is set to $bool\n";
$bool = false;
print "Bool is set to $bool\n";
```

Output:

```
Bool is set to 1

Bool is set to
```

True is converted to 1 while false is converted to an empty string.
To change that behaviour we need an explicit cast:

```
print "Bool is set to ";
print (int)$bool;
```

Variable Variables

- Allow you to access the contents of a variable without knowing its name directly – it is like indirectly referring to the variable:

```
$bar = 10;
$foo = "bar";
```

- From that point, there are two ways to output the value of $bar:

```
print $bar;
print $$foo;
```

- PHP will lookup $foo, convert it to a string and look up the variable of the same name and return its value. This indirect access is possible at an arbitrary level of complexity, e.g. $$$bar and $$$$$foo.

References

- When you use the = (assignment) operator, PHP performs a "copy assignment". It takes the value from operand two and copies it into operand one. This does not work when we want to be able to change operand two later. It is then when we use references.
- References are variable aliases. The & operator creates the alias:

```
$big_long_variable_name = "PHP"
$short = & $big_long_variable_name;
$big_long_variable_name .= " rocks!";
print "\$short is $short\n";
print "Long is $big_long_variable_name\n";
```

results in:

```
$short is PHP rocks !
Long is PHP rocks !
```

References II

- If we continue with the previous example and change the values after the assignment:

```
$short = "Programming $short";
print "\$short is $short\n";
print "Long is $big_long_variable_name\n";
```

results in:

```
$short is Programming PHP rocks !
Long is Programming PHP rocks !
```

- This is very useful for functions, when we want to avoid copying large strings or arrays:

Note the & in the code:

```
function &ret_ref() {
        $var = "PHP";
        return $var;
}

$v = & ret_ref();
```

Memory Management - Symbol Tables

- PHP uses reference counting and copy-on-write to manage memory.
- To do that PHP utilizes symbol tables. There are two parts to a variable. It's name e.g. $name and its value e.g. “Fred”. A symbol table is an array that maps variable names to positions of their values in memory.
- When you copy a value from one variable to another, PHP doesn't get more memory for a copy of the value. Instead it updates the symbol table to say both variables are names for the same “chunk of memory”.

Copy-on-write

- So the following code does not actually create a new array.

```
$worker = array(“Fred”,35,”Flintstone”);
$other = $worker;      // array isn't copied
```

- If you then modify either copy, PHP allocates the memory and makes the copy:

```
$worker[1] = 36;
```

- By delaying the allocation and copying, PHP saves time and memory in lot of situations. This is copy-on-write.

Reference Count

- Each value pointed to by a symbol table has a reference count. The reference count represents the number of ways there are to access that piece of memory.
- After the initial assignment of the array $worker and $worker to $other, the array pointed to by the symbol table entries $worker and $other has reference count of 2.
- When a variable goes out of scope (e.g. end of the function), the reference count is decreased by one.
- When the reference count of a value reaches 0, its memory is freed. This is reference counting.
- With isset(*var*) and unset(*var*) you can gain control of the memory management.

String Access

```
strlen($str);
```
Length of string

```
$str[$i];
```
Access ith character

```
trim($str,[, charlist]);
ltrim($str,[, charlist]);
rtrim($str,[, charlist]);
```
trims whitespace, optional characters can be trimmed with charlist, ltrim – leftside, rtrim – rightside only.

```
strtolower($str);
strtoupper($str);
```
Changing the case to lower / uppercase.

```
ucfirst($str);
```
Operates only on first character of string

```
ucwords($str);
```
Operates only on first character of each word

String Manipulation

```
$name = “Fred Flintstone”;
$fluff = substr($name,6,4);
$sound = substr($name,11);
```
Result:

“lint”
“tone”

```
substr_replace(string,
               replacement,
               start,length);
```
replaces a part of a string with another string.

```
strcmp(string1,string2);
```
Compares two strings

- 0 - if the two strings are equal
- <0 - if string1 is less than string2
- >0 - if string1 is greater than string2

String - Replace

```
str_replace  ($search,$replace,
              $subject,  [, int &$count  ] )
```

Many more functions can be found at http://php.net

But what if you need something more fancy?

Regular Expressions

- A regular expression (regex or regexp for short) is a special text string for describing a search pattern. You can think of regular expressions as wildcards with a range of additional capabilities.
- You are probably familiar with wildcard notations such as *.txt to find all text files in a file manager.

  The regex equivalent is .*\.txt$.

Regular Expressions

- Offer you more power over your strings
  - Replace text
  - Test for a pattern within a string
  - Extract a substring from within a String
- Have a complex syntax
- String functions are usually faster and easier to read. Use regular expressions with care and if you have a particular need.

PHP Regular Expressions

- PHP supports two types of regular expressions
  - POSIX-extended
  - Perl-Compatible Regular Expressions (PCRE)
- PCRE are more powerful than the POSIX ones and also faster.
- Basic PCRE functions are:

```
preg_match(pattern,text);
preg_match_all(pattern,text);
```

Preg_match applies the pattern to the text and returns 1 if it found a match and 0 if it doesn't. For speed reasons only the first match is returned (preg_match_all does not exit after first match – more later).

Examples I

preg_match(“/php/”, “php”)	True
preg_match(“php/”, “php”)	Error; you need a slash at the start
preg_match(“/php/”, “PHP”)	False, regexps are case-sensitive
preg_match(“/php/i”, “PHP”)	True; /i means case-insensitive
preg_match(“/Foo/i”, “FOO”)	True

Regex Syntax

- Regular expressions are formed
  - Starting with a delimiter (forward slashes (/), hash signs (#) and tildes (~))
  - Followed by a sequence of special symbols and words to match
  - Then another delimiter (see above)
  - Optionally a modifier, i.e. string of letters that affect the expression (e.g. i modifier in the previous example)
- If the delimiter needs to be matched inside the pattern it must be escaped using a backslash. If the delimiter appears often inside the pattern, it is a good idea to choose another delimiter in order to increase readability.

```
/http:\/\//
#http://#
```

The preg_quote() function may be used to escape a string for injection into a pattern and its optional second parameter may be used to specify the delimiter to be escaped.

Writing a Regex Checker

```
<?php  extract($_POST); ?>

<html>
<h1>Regular Expression Checker</h1>

<form method="POST" action="<?php $_SERVER['PHP_SELF']; ?>">

       Pattern: <textarea name="pattern" cols="140" rows="3">
        <?php print ((isset($pattern))? $pattern: ''); ?>
</textarea> <br /><br />
       String: <input type="text" name="str"
     value="<?php print ((isset($str)) ? $str : ''); ?>" />

       <input type="submit" value= "Test Expression" />

</form> <!-- continued on next slide -->
```

Writing a Regex Checker cont.

```
<?php
if (isset($pattern) && isset($str)) {
    $test = preg_match($pattern,$str);
    print "<br>" ;

    if ($test) {
        print $pattern . "<br /><br /> matches <br /><br /> " . $str ;
     } else {
        print $pattern . "<br /><br /> does not match <br /><br />" . $str ;
     }
} else {
    print "<p style='font-color:red;'> Enter a pattern and a expression.</p>";
}
?>
<html>
```

Regex Character Classes

- Regex allow you to form character classes of words using brackets [ ]
  - [Ff] will match “F” or “f”
  - [A-Z] will match the range of all upper-case letters
  - [A-Z][a-z] matches all letters, whether upper or lower case
  - [a-z0-9] matches lower-case letters and numbers only
- At the beginning of a Character class the caret ^ means “not”
  - [^A-Z] matches everything but upper-case letters
  - [^A-Za-z0-9] will accept symbols only – no upper-case letters, no lower-case letters, no numbers

Regex Character Classes - Examples

preg_match(“/[Ff]oo/”, “Foo”)	True
preg_match(“/[^Ff]oo/”, “Foo”)	False, regex says anything not F or f followed by oo, would match too, boo, zoo
preg_match(“/[A-Z][0-9]/”, “K9”)	True
preg_match(“/[A-S]esting/”, “Testing”)	False, acceptable range ends at S
preg_match(“/[A-T]esting/”, “Testing”)	True, range is inclusive
preg_match(“/[a-z]esting[0-9][0-9]/”, “Testing AA”)	False

Regex Character Classes - Examples

preg_match("/[a-z]esting[0-9][0-9]/", "testing99)	True
preg_match("/[a-z]esting[0-9][0-9]/", "Testing99)	False, case sensitivity
preg_match("/[a-z]esting[0-9][0-9]/i", "Testing99)	True, case problems fixed with modifier i
preg_match("/[^a-z]esting/", "Testing99)	True, first character can be anything that is not a lowercase letter
preg_match("/[^a-z]esting/i", "testing99)	False ! The range excludes lowercase characters only. But i makes it insensitive, which turns [^a-z] into [^a-zA-Z] !!!

Regex Special Characters

- The metacharacters +, *, ? and { } affect the number of times a pattern should be matched and ( ) allows you to create subpatterns, and $ and ^ affect the position
  - + … match 1 or more of the previous expression
  - * … match 0 or more of the previous expression
  - ? … match 0 or one of the previous expression

```
preg_match("/[A-Za-z]*/", $string);
// matches "", "a", "aaaa", "The sun has got his
hat on", etc.

preg_match("/-?[0-9]+/", $string);
// matches 1, 100, 98798798, and also -1,
-23402. etc. → -?
```

Required and Optional Matches

- This example shows two character classes where the first is required and the second optional:

```
preg_match("/\$[A-Za-z_][A-Z][a-z_0-9]*/",$string);
```

Because $ is a regex symbol (start of line) we need a backslash \

required

optional
* matches 0 or more times

This regex matches PHP variable names.

Examples

preg_match(“/[A-Z]+/”,”123”);	False
preg_match(“/[A-Z][A-Z0-9]+/i”,”A123”);	True
preg_match(“/[0-9]?[A-Z]+/”, ”10 Green Bottles”);	True, matches “0G”
preg_match(“/[0-9]?[A-Z0-9]*/i”, ”10 Green Bottles”);	True
preg_match(“/[A-Z]?[A-Z]?[A-Z]*/”,””);	True; 0 or 1 match, then 0 or 1 match, then 0 or more means “” matches.

Regex - Braces

Braces { } can be used to define specific repeat counts in three different ways:

1) {n} … matches n instances of the previous expression
2) {n,} … matches a minimum of n instances of the previous expression
3) {m,n} … match a minimum of m and a maximum on n instances of the previous expression.

Examples

preg_match(“/[A-Z]{3}/”,”FuZ”);	False, the regex will match exactly 3 uppercase letters
preg_match(“/[A-Z]{3}/i”,”FuZ”);	True, same as above but modifier i set.
preg_match(“/[0-9]{3}-[0-9]{4}/”,”555-1234”);	True, precisely 3 numbers then dash then 4 numbers matches U.S. Telephone numbers
preg_match(“/[a-z]+[0-9]?[a-z]{1}/”,”aaa1”);	True, must end with one lowercase letter
preg_match(“/[A-Z]{1,}99/”,”99”);	False, must start with at least one uppercase letter
preg_match(“/[A-Z]{1,5}99/”,”FINGERS99”);	True; “S99”, “RS99”, “ERS99”, “GERS99”, and “NGERS99” all fit criteria
preg_match(“/[A-Z]{1,5}[0-9]{2}/”,”adams42”)	True

Parenthesis

- Parenthesis inside regular expressions allow you to define subpatterns that should be matched individually. The most common use for these is to specify groups of alternatives for matches, allowing you to match very specific criteria.
- E.g. (cat|car) sat on the (mat|drive) matches
  1) The cat sat on the mat
  2) The car sat on the mat
  3) The cat sat on the drive
  4) The car sat on the drive

Examples

preg_match("/(Linux|Mac OS X)/", "Linux")	True
preg_match("/(Linux|Mac OS X){2}/", "Mac OS Xlinux")	True
preg_match("/(Linux|Mac OS X){2}/", "Mac OS X Linux")	False, There is a space there, which is not part of the regex.
preg_match("/contra(diction|vention)/", "contravention")	True
preg_match("/Windows ([0-9][0-9]+|Me|XP)/", "Windows 2000")	True, matches 95,98,2000,2003, Me, and XP
preg_match("/Windows (([0-9][0-9]+|Me|XP)|Codename (Whistler|Longhorn))/","Windows Codename Whistler")	True, uses nested sub-patterns to match all version of Windows but also Codenames.

Start and End of Line

- The dollar $ and the caret ^ symbols, stand for "end of line" and "start of line", respectively.

```
$multitest="This is \na long test \nto
see whether \nthe dollar \nSymbol \nand
the \ncaret symbol \nwork as planned";
```

\n … NewLine character

```
This is
A long test
To see whether
The dollar
Symbol
And the
caret symbol
Work as planned.
```

Examples

```
preg_match("/is$/m", $multiline);
preg_match("/the$/m", $multiline);
preg_match("/^the/m", $multiline);
preg_match("/^Symbol/m", $multiline);
```

$$\Big\}\ \text{TRUE}\ \downarrow$$

This is
A long test
To see whether
The dollar
Symbol
And the
caret symbol
Work as planned.

To parse multiline Strings we need the **m** Modifier. Without it the line would be interpreted as one line. The modifier m will consider start (^) and end of lines ($) whereever a newline character is.

Words and Whitespace Regex

- The previously presented patterns are very common but there are many more we will not cover. 5 patterns that are very often in use are the following word and whitespace patterns:

1) .	… match a single character except newline
2) \s	… match any whitespace
3) \S	… match any non whitespace
4) \b	… on a word boundary
5) \B	… not on a word boundary

Example

$string = "Foolish child!";

preg_match("[\S]{7}[\s]{1}[\S]{6}/",$string) True

$string = "Foolish child!";

preg_match("/oo\b/i",$string) False (but e.g. foo, zoo)

preg_match("/oo\B/i",$string) True (also e.g. wool, pool)

Storing Matched Strings

- The preg_match() and the preg_match_all() function have a third parameter that allows you to pass in an array for it to store a list of matched strings.

```
$a = "Foo moo boo tool foo";
preg_match("/[A-Za-z]oo\b/i",$a, $matches);
```

```
preg_match_all("/[A-Za-z]oo\b/i",$a, $matches);
```

```
var_dump($matches);
```

var_dump(*var*) outputs the contents of the variables passed to it for closer inspection and is particularly useful with arrays and objects.

Replacements with Regex

- The function preg_replace() allows to accomplish string replacement. It works the same way as preg_match().

```
preg_replace(pattern,replacement, subject);
```

Example:
```
$a = "Foo moo boo tool foo";
$b = "preg_replace("/[A-Za-z]oo\b/", "Got word: $0\n", $a);
```

That script would output the following:

```
Got word: Foo
Got word: Foo
Got word: Foo
tool Got word: foo
```

The 2. parameter is plain text but can contain **$n** to insert the text matched by subpattern n of your regex rule. If you have no subpatterns, you should use **$0** to use the matched text.

Regex Replacement - Subpatterns

- If you are using subpatterns then $1, $2 and so on are set to the individual matches for each subpattern:

```
$match = "/the (car|cat) sat on the (drive|mat)/";
$input = "the cat sat on the mat";
print_reg_replace($match, "Matched $0, $1, and $2\n", $input);
```

$0 … the cat sat on the mat
$1 … cat
$2 … mat

Screen Scraping with Regex

- Regular expressions can be used to extract information of a web page.
- Think of sports results or images or other interesting items that you might want to extract from a page and use on your page.
- These technique is called screen scraping
- Before you extract materials from another website you need to check the legal situation.
  - What does the terms of use of the target website say about extracting information by automated means ?
  - Can you republish contents extracted, what are the conditions in terms of ownership and copyrights ?

Image Extractor

```
<?php
// extract the variables specified in the form on this site
extract($_POST);
?>

<form method="POST" action="<?php $_SERVER['PHP_SELF']; ?>">
String: <input type="text" name="url" value="<?php print
((isset($url)) ? $url : ''); ?>" />

<input type="submit" value= "Extract Images" />

</form>
```

Note: Illustration only head, body, etc. are omitted in this example

Image Extractor

For this example to work you need Curl support enabled by your PHP installation.

```
// create curl resource
$ch = curl_init();

// set url - this is the on entered into the form
curl_setopt($ch, CURLOPT_URL, $url);

//return the transfer as a string
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

// $output contains the output string
$output = curl_exec($ch);

 // Here comes the code to extract tags from $output
      ...

// close curl resource to free up system resources
curl_close($ch);
```

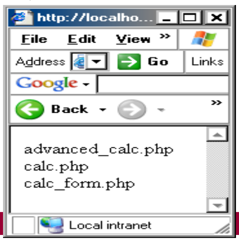
```
$pat = "/<img[^>]+>/i" ;
preg_match_all($pattern,$output,$matches);

foreach( $matches[0] as $key => $img_tag)
{
  $pat = "%src=[\s]*\"%";

  if (preg_match("%src=[\s]*\"/%",$img_tag)) {
    // image referenced with a relative link without /
    print (preg_replace($pat,("src=\"http://" . $url),
                           $img_tag)."\n") ;
  }
  else if( … )preg_match("%src=[\s]*\"(http)%",$img_tag)) {
     // image referenced with an absolute link
     print ($img_tag . "\n") ;  // no changes required just print
  }
  else {
    // image referenced with an relative link but no backslash
    print (preg_replace($pat, ("src=\"http://" . $url . "/"),
                        $img_tag) . "\n" ) ;
      }

}
```

Regex Summary

- Match Email - http://www.regular-expressions.info/email.html
- Form example enter email – do again form processing
- Password File
- Trim whitespaces leading and following
- String Manipulation

- Lets modify the initial example using a regular expression
- We only list files with the extension **.php**

```
<?php
    $handle = opendir('Stuff');
    if($handle) {
      while(false !== ($file = readdir($handle))){
            if(preg_match("/\w+(.php)/",$file)){
            print "$file <br>";
            }
      }
      closedir($handle);
    }
?>
```



```
advanced_calc.php
calc.php
calc_form.php
```

Regex for a British Postcode ?

^([A-PR-UWYZ0-9][A-HK-Y0-9][AEHMNPRTVXY0-9]?[ABEHMNPRVWXY0-9]? {1,2}[0-9][ABD-HJLN-UW-Z]{2}|GIR 0AA)$

Look at the discussion:
http://www.regxlib.com/REDetails.aspx?regexp_id=260

Literature

- Hudson Paul (2005): PHP in a Nutshell, O'Reilly, QA 76.73.P224.H3 (most examples taken from there)
- Lerdorf Rasmus and Tatroe Kevin (2002): Programming PHP, O'Reilly, QA 76.73.P224.L3
- Watt Andrew, Beginning Regular Expressions, Wrox 2005, QA76.9.T48.W2
- http://php.net
- http://www.regular-expressions.info/
- http://nadeausoftware.com/articles/2007/06/php_tip_how_get_web_page_using_curl
- http://www.catswhocode.com/blog/15-php-regular-expressions-for-web-developers