Universiteti i Prishtinës

Programimi në Internet

Strings and Regular Expressions in PHP

Dr. Ing. Lule Ahmedi

Strings and their Workaround

Questions that often arise:

- How to format and manipulate text?
- How to search words, phrases, or other patterns within a string?
- How to strore data into and retrieve them from a database
- How to build search engines

In this lecture:

- Formatting strings
- Joining and splitting strings
- Comparing strings
- Matching and replacing substrings
- Using regular expressions

Formatting Strings

`trim()`, `ltrim()`, `chop()`:

- Strip whitespaces (`\n`, `\r`, `\t`, `\v`, `\0`, spaces) from the start and/or end of a string
- Characters other then whitespaces can be stripped out if specified as a second argument

HTML formatting with `nl2br()`:

- Replaces all the newlines with `<br/>` tag
- Otherwise a single line displayed to the browser

Change the case of a string:

- `strtoupper()`, `strtolower()`, `ucfirst()`, `ucwords()`

Escape special characters like quotes, backslashes, etc.:

- `AddSlashes(string` $input$`)`, `StripSlashes(`$input$`)`

Format a String for Printing

- `echo()`, `print()` (the latter returns a boolean value)
- `printf()` - format a string before printing it to the browser
- `sprintf()` - format a string and return it
- `printf ("Total amount of order is %s (with shipping %.2f) ", $total, $total_shipping);`
  - %s and %.2f are *type specifiers*

Joining and Splitting Strings

- Want to look at words in a sentence (say for spellchecking)
- Want to split a domain name, a complete URL, or email address into its components
- `array explode(string` *separator*`, string` *input* `[, int` *limit*`]);`
  - Splits a string *input* into (*number* of) pieces on a specified *separator* string

```
$email_array = explode('@', $email);

if ($email_array[1]=='bigcustomer.com')
$toaddress =
  'bob@example.com'; else $toaddress = 'feedback@example.com';
```

Joining and Splitting Strings (cont.)

- implode() or join() - the reverse effects of explode()
- `$new_email = implode('@', $email_array);`
- `strtok()` similar to `explode()`, but gets pieces (tokens) from a string one at a time
- `string substr(string` *string*`, int` *start* `[, int` *length*`] );`
- `substr('Your customer service is excellent', 0, 4)` would result into `'Your'`

Comparing Strings

- `int strcmp(string` $str1$`, string` $str2$`);`
- `strcasecmp()` - is identical, but not case sensitive
- `strnatcmp()` - compares strings according to a "natural order", not according to a "lexicographical order"
- `strlen(string` $length$`)` - gets the length of a string

Matching and Replacing Substrings

- `string strstr(string` *haystack*`, string` *needle*`);`
- `int strpos(string` *haystack*`, string` *needle*`, int [`*offset*`] );`
- `mixed str_replace(mixed` *needle*`, mixed` *new_needle*`, mixed` *haystack*`);`
  - Will replace all the instances of *needle* on *haystack* with *new_needle* and return the new version of *haystack*
  - `$feedback = str_replace($offcolor, '%!@', $feedback);`
- `string substr_replace(string` *string*`, string` *replacement*`, int` *start*`, int [`*length*`]);`
  - Will replace part (defined with the last two parameters) of *string* with *replacement*
  - `$test = substr_replace($test, 'X', -1);`

Regular Expressions

Two styles of syntax supported in PHP:

- POSIX style: compiled by default
  - Easier to learn and execute faster, but binary-unsafe
- Perl-compatible regular expressions style : compiles in the PCRE library

Pattern matching

- Using string functions: *limited* to exact (substring) match
- Regular expressions necessary for more complex ones

What are Regular Expressions?

A way of describing a pattern in a piece of text

- **Exact match**, e.g., searching for terms like "shop", "delivery"

Matching regexprs is similar to `strstr()`, i.g., match a string somewhere within another string

- **Substring match**, e.g., "shop" matches the regexpr "shop", but also "h", "ho", ..

More advanced regexprs:

- Use **meta-characters** to indicate, say,
  - For a pattern to occur at the start or end of a string
  - For a part of the pattern to be repeated
  - For characters in a pattern to be of particular type

Character Sets and Classes

Character sets - to match any character of a particular type (a kind of wildcard)

- `'.'` - a wildcard for any single character (except \n)
  - `.at` matches `'cat'`, `'#at'`, `'sat'`, ..
  - Often used for filename matching in operating systems
- `[a-z]`, `[a-zA-Z]`, or `[aeiou]` - a character class, i.e., a set of characters (can be a range, like is `[a-z]`, or a set of ranges, like is `[a-zA-Z]`) to which a (*single*) matched character must belong to; the latter is
- `[^a-z]` - matches any character that is not between a and z
- A number of **predefined character classes** exists: `[[:alnum:]]`, `[[:alpha:]]`, `[[:space:]]`, etc.

Repetition and Subexpressions

A repetition:

- There might be multiple occurences of a particular string or class of characters
  - `'*'` zero or more occurences of a pattern
  - `'+'` one or more occurences of a pattern
- Example: `[[:alnum:]]+` means "at least one alphanumeric character"

Subexpressions:

- Expressions within paranthesis to address patterns on a more fine granular basis
  - For each subexpression one pattern

Counted Subexpressions

- `{n}` - exact nunmber (n) of repetitions
- `{m, n}` - a range from m to n of repetitions
- `{m, }` - opened ended range of repetitions, at least m of them
  - Example: `(very ){1, 3}` - matches 'very ', 'very very ', and 'very very very '

Anchoring to the Begining or End of a String

- The caret symbol ’^’ - a particular subexpression should appear at the start of a searched string
- ’$’ - analogously, at the end

Examples:

- `^bob` - matches `bob` at the start of a string
- `com$` - matches `com` at the end of a string
- `^[a-z]$` - matches any single character from a to z, in the string of its own

Branching

- A vertical pipe ’|’ - represents a choice in a regexpr
  - `(com)|(edu)|(net)` - matches either of the strings within parentheses

Matching Special Characters

- Special characters, such as '.', '{', or '$'
  - To match them, a slash '\' should be put in front of it, say \., or \{, or, \$

RegExpr Example #1

Detect particular terms in the customer feedback:

- To match on `'shop'`, `'customer feedback'`, or `'retail'`
  - Using string functions: three different searches, one for each term
  - Using regular expressions: in one pass
    - `shop|customer service|retail`

RegExpr Example #2: E-mail Validator

- Encode the standardized format of an email address

```
^[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-\.]+$
```

- `^[a-zA-Z0-9_\-\.]+` - start the string with at least one letter, number, underscore, hyphen, or dot, or some combination of those (username)
- `@` - matches a literal '@'
- `[a-zA-Z0-9\-]+` - matches the first part of the host name including aphanumeric characters and hyphens
- `\.` - matches a literal '.' (slashed out since '.' is a special character, like _ and - as well
- `[a-zA-Z0-9\-\.]+$` - matches the rest of a domain name, including letters, numbers, hyphens, and more dots if required, up until the end of the string

Finding Substrings

- `int ereg(string` $pattern$`, string` $search$`, array [`$matches$`]);`
- It searches for matches of the $pattern$ in the $search$ string
- If searches are found for subexpressions of $pattern$, they will be stored in the array $matches$, one subexpression per array element
- `eregi()` is identical, but case insensitive

Finding Substrings Example

```
<?php
if (!eregi
 ('^[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-\.]+$',
                                                    $email))
{
 echo 'That's not a valid email address. Please return'
       .' to the previous page and try again.';
 exit;
}

$toaddress = 'feedback@example.com';       // default
if (eregi('shop|customer service|retail', $feedback))
  $toaddress = 'retail@example.com';
else if (eregi('deliver.*|fulfil.*', $feedback))
  $toaddress = 'fulfilment@example.com';
else if (eregi('bill|account', $feedback))
  $toaddress = 'account@example.com';

if (eregi('bigcustomer\.com', $email))
  $toaddress = 'bob@example.com';
?>
```

Replacing Substrings

Similar to the `str_replace()` string function

- `string ereg_replace(string` $pattern$`, string` $replacement$`, string` $search$`);`
- It searches for the $pattern$ in the $search$ string and replaces it with the string $replacement$
- `eregi_replace()` is identical, but not case sensitive

Splitting Strings

- `array split(string` $pattern$`, string` $search$`, int [max]);`
- It splits the string $search$ into substrings on the $pattern$ and returns the (max number of) substrings in an array

To split the host name into its components:

```
$domain = 'cst.see-university.edu.mk';
$arr = split ('\.', $domain);

while (list($key, $value) = each ($arr))
   echo '<br/>'.$value;
```

String Functions vs. RegExprs

- Regular expressions run less efficiently than the string functions with similar functionality
  - $\Rightarrow$ For simple applications, use string expressions where possible