CHAPTER 1

Working with Strings

IN THIS CHAPTER:

If you're like most novice PHP developers, you probably have only a passing acquaintance with PHP's string functions. Sure, you know how to print output to a Web page, and you can probably split strings apart and glue them back together again. But there's a lot more to PHP's string toolkit than this: PHP has more than 175 string manipulation functions, and new ones are added on a regular basis. Ever wondered what they were all for?

If you have, you're going to be thrilled with the listings in this chapter. In addition to offering you a broad overview of PHP's string manipulation capabilities, this chapter discusses many other tasks commonly associated with strings in PHP—removing unnecessary whitespace, finding and replacing string patterns, counting and extracting string segments, identifying duplicate words, encrypting text and generating string passwords. Along the way, you'll find out a little more about those mysterious string functions, and also learn a few tricks to help you write more efficient code.

1.1 Controlling String Case

Problem

You want to force a string value to upper- or lowercase.

Solution

Use the `strtoupper()` or `strtolower()` functions:

```
<?php
// define string
$rhyme = "And all the king's men couldn't put him together again";

// uppercase entire string
// result: "AND ALL THE KING'S MEN COULDN'T PUT HIM TOGETHER AGAIN"
$ucstr = strtoupper($rhyme);
echo $ucstr;

// lowercase entire string
// result: "and all the king's men couldn't put him together again"
$lcstr = strtolower($rhyme);
echo $lcstr;
?>
```

Comments

When it comes to altering the case of a string, PHP makes it easy with four built-in functions. Two of them are illustrated previously: the `strtoupper()` function uppercases all the characters in a string, while the `strtolower()` function lowercases all the characters in a string.

For more precise control, consider the `ucfirst()` function, which capitalizes the first character of a string (good for sentences), and the `ucwords()` function, which capitalizes the first character of every word in the string (good for titles). Here's an example:

```
<?php
// define string
$rhyme = "and all the king's men couldn't put him together again";

// uppercase first character of string
// result: "And all the king's men couldn't put him together again"
$ucfstr = ucfirst($rhyme);
echo $ucfstr;

// uppercase first character of every word of string
// result: "And All The King's Men Couldn't Put Him Together Again"
$ucwstr = ucwords($rhyme);
echo $ucwstr;
?>
```

1.2 Checking for Empty String Values

Problem

You want to check if a string value contains valid characters.

Solution

Use a combination of PHP's `isset()` and `trim()` functions:

```
<?php
// define string
$str = "  ";
```

```
// check if string is empty
// result: "Empty"
echo (!isset($str) || trim($str) == "") ? "Empty" : "Not empty";
?>
```

Comments

You'll use this often when working with form data, to see if a required form field contains valid data or not. The basic technique is simple: use `isset()` to verify that the string variable exists, then use the `trim()` function to trim whitespace from the edges and equate it to an empty string. If the test returns true, it's confirmation that the string contains no value.

NOTE

It's instructive to note that many developers use PHP's `empty()` function for this purpose. This isn't usually a good idea, because `empty()` will return true even if the string passed to it contains the number `0` (PHP treats `0` as Boolean `false`). So, in the following illustration, the script will produce the result `"Empty"` even though the string variable actually contains data.

```
<?php
// define string
$str = "0";

// check if string is empty
// result: "Empty"
echo (empty($str)) ? "Empty" : "Not empty";
?>
```

1.3 Removing Characters from the Ends of a String

Problem

You want to remove the first/last *n* characters from a string.

Solution

Use the `substr()` function to slice off the required number of characters from the beginning or end of the string:

```
<?php
// define string
$str = "serendipity";
```

```
// remove first 6 characters
// result: "ipity"
$newStr = substr($str, 6);
echo $newStr;

// remove last 6 characters
// result: "seren"
$newStr = substr($str, 0, -6);
echo $newStr;
?>
```

Comments

The `substr()` function enables you to slice and dice strings into smaller strings. It typically accepts three arguments, of which the last is optional: the string to act on, the position to begin slicing at, and the number of characters to return from its start position. A negative value for the third argument tells PHP to remove characters from the end of the string.

1.4 Removing Whitespace from Strings

Problem

You want to remove all or some whitespace from a string, or compress multiple spaces in a string.

Solution

Use a regular expression to find and replace multiple whitespace characters with a single one:

```
<?php
// define string
$str = "   this is a string with  lots of    emb e dd   ↵
ed whitespace    ";

// trim the whitespace at the ends of the string
// compress the whitespace in the middle of the string
// result: "this is a string with lots of emb e dd ed whitespace"
$newStr = ereg_replace('[[:space:]]+', ' ', trim($str));
echo $newStr;
?>
```

Comments

There are two steps involved in performing this task. First, use the `trim()` function to delete the unnecessary whitespace from the ends of the string. Next, use the `ereg_replace()` function to find multiple whitespace characters in the string and replace them with a single space. The end result is a string with all extra whitespace removed.

Alternatively, remove all the whitespace from a string, by altering the replacement string used by `ereg_replace()`. The following variant illustrates this:

```
<?php
// define string
$str = "   this is a string with  lots of    emb e dd   ↵
ed whitespace    ";

// remove all whitespace from the string
// result: "thisisastringwithlotsofembeddedwhitespace"
$newStr = ereg_replace('[[:space:]]+', '', trim($str));
echo $newStr;
?>
```

1.5 Reversing Strings

Problem

You want to `reverse` a string.

Solution

Use the strrev() function:

```
<?php
// define string
$cards = "Visa, MasterCard and American Express accepted";

// reverse string
// result: "detpecca sserpxE naciremA dna draCretsaM ,asiV"
$sdrac = strrev($cards);
echo $sdrac;
?>
```

Comments

It's extremely simple, this "give it a string, and `strrev()` gives it back to you in reverse" task. But despite the fact that it's nothing to write home about, `strrev()` is often used to perform some advanced tasks. See the listing in "1.20: Extracting Sentences from a Paragraph" for an example.

1.6 Repeating Strings

Problem

You want to repeat a string *n* times.

Solution

Use the `str_repeat()` function:

```
<?php
// define string
$laugh = "ha ";

// repeat string
// result: "ha ha ha ha ha ha ha ha ha ha "
$rlaugh = str_repeat($laugh, 10);
echo $rlaugh;
?>
```

Comments

PHP's `str_repeat()` function is equivalent to Perl's `x` operator: it repeats a string a fixed number of times. The first argument to `str_repeat()` is the string to be replicated; the second is the number of times to replicate it.

The `str_repeat()` function can come in quite handy if you need to print a boundary line of special characters across your output page—for example, an unbroken line of dashes or spaces. To see this in action, view the output of the following code snippet in your browser—it displays a line of `Ø` characters across the page by continuously printing the HTML character code `&Oslash`:

```
<?php
// define string
$special = "&Oslash;";
```

```
// repeat string
$rspecial = str_repeat($special, 62);
echo $rspecial;
?>
```

1.7 Truncating Strings

Problem

You want to truncate a long string to a particular length, and replace the truncated characters with a custom placeholder—for example, with ellipses.

Solution

Use the `substr()` function to truncate the string to a specified length, and append the custom placeholder to the truncated string:

```
<?php
function truncateString($str, $maxChars=40, $holder="...") {
    // check string length
    // truncate if necessary
    if (strlen($str) > $maxChars) {
        return trim(substr($str, 0, $maxChars)) . $holder;
    } else  {
        return $str;
    }
}

// define long string
$str = "Just as there are different flavors of client-side scripting,↵
there are different languages that can be used on
the server as well.";

// truncate and print string
// result: "Just as there are different flavours of..."
echo truncateString($str);

// truncate and print string
// result: "Just as there are di >>>"
echo truncateString($str, 20, " >>>");
?>
```

Comments

The user-defined function `truncateString()` accepts three arguments: the string to truncate, the length at which to truncate it (default 40 characters), and the custom character sequence to use at the point of termination (default ...). Within the function, the `strlen()` function first checks if the string is over or under the permissible limit. If it's over the limit, the `substr()` function slices off the bottom end of the string, and the placeholder is appended to the top end.

1.8 Converting Between ASCII Characters and Codes

Problem

You want to retrieve the American Standard Code for Information Interchange (ASCII) code corresponding to a particular character, or vice versa.

Solution

Use the `ord()` function to get the ASCII code for a character:

```
<?php
// define character
$char = "\r";

// retrieve ASCII code
// result: 13
$asc = ord($char);
echo $asc;
?>
```

Use the `chr()` function to get the character corresponding to an ASCII code:

```
<?php
// define ASCII code
$asc = 65;

// retrieve character
// result: "A"
$char = chr($asc);
echo $char;
?>
```

Comments

PHP's `ord()` function retrieves the ASCII code corresponding to a particular character (or the first character, if the argument to `ord()` contains more than one character). The `chr()` function does the reverse, returning the character corresponding to a specific ASCII code.

You can use `chr()` to generate the entire alphabet, if you like:

```
<?php
// result: "abcd...xyz"
for ($a=97; $a<(97+26); $a++) {
    echo chr($a);
}
?>
```

NOTE

You can find a list of ASCII characters and codes at `http://www.lookuptables.com/`, and a Unicode table at `http://www.unicode.org/Public/UNIDATA/NamesList.txt`.

1.9 Splitting Strings into Smaller Chunks

Problem

You want to break up a long string into smaller segments, each of a fixed size.

Solution

Use the `str_split()` function to break the string into fixed-length "chunks":

```
<?php
// define string
$str = "The mice jumped over the cat, giggling madly ↵
as the moon exploded into green and purple confetti";

// define chunk size
$chunkSize = 11;

// split string into chunks
// result: [0] = The mice ju [1] = mped over t [2] = he cat, gig
// [3] = gling madly ...
```

```
$chunkedArr = str_split($str, $chunkSize);
print_r($chunkedArr);
?>
```

Comments

The `str_split()` function splits a string into fixed-length blocks and returns them as elements of an array. By default, each "chunk" is one character long, but you can alter this by passing the `str_split()` function a second argument defining the chunk size (as in the previous snippet).

1.10 Comparing Strings for Similarity

Problem

You want to compare two strings to see if they sound similar.

Solution

Use the `metaphone()` function to test if the strings sound alike:

```
<?php
// compare strings
// result: "Strings are similar"
echo (metaphone("rest") == metaphone("reset")) ? ↵
"Strings are similar" : "Strings are not similar";

// result: "Strings are similar"
echo (metaphone("deep") == metaphone("dip")) ? ↵
"Strings are similar" : "Strings are not similar";

// result: "Strings are not similar"
echo (metaphone("fire") == metaphone("higher")) ? ↵
"Strings are similar" : "Strings are not similar";
?>
```

Comments

PHP's `metaphone()` function—a more accurate version of its `soundex()` function—is one of the more unique ones in the PHP string toolkit. Essentially, this function produces a signature for the way a string sounds; similar-sounding strings

produce the same signature. You can use this property to test two strings to see if they're similar—simply calculate the `metaphone()` keys of each string and see if they're the same.

TIP

The `metaphone()` function comes in handy in search queries, to find words similar to the search string the user provides. Also consider the `levenshtein()` and `similar_text()` functions to compare strings by character instead of pronunciation.

1.11 Parsing Comma-Separated Lists

Problem

You want to extract the individual elements of a comma-separated list.

Solution

Decompose the string into an array using the comma as the delimiter:

```
<?php
// define comma-separated list
$ingredientsStr = "butter, milk, sugar, salt, flour, caramel";

// decompose string into array
// using comma as delimiter
$ingredientsArr = explode(", ", $ingredientsStr);

// iterate over array
// print individual elements
foreach ($ingredientsArr as $i) {
    print $i . "\r\n";
}
?>
```

Comments

PHP's `explode()` function makes it a single-step process to split a comma-separated string list into an array of individual list elements. The previous listing clearly illustrates this: the `explode()` function scans the string for the delimiter and cuts out the pieces around it, placing them in an array. Once the list items have been extracted, a `foreach()` loop is a good way to process the resulting array.

TIP

You can combine the elements of an array into a comma-separated string list—the reverse of the listing above—with PHP's `implode()` function.

1.12 Parsing URLs

Problem

You want to extract the protocol, domain name, path, or other significant component of a URL.

Solution

Use the `parse_url()` function to automatically split the URL into its constituent parts:

```
<?php
// define URL
$url = "http://www.melonfire.com:80/community/columns/trog/ ↵
article.php?id=79 &page=2";

// parse URL into associative array
$data = parse_url($url);

// print URL components
foreach ($data as $k=>$v) {
    echo "$k: $v \n";
}
?>
```

Comments

The `parse_url()` function is one of PHP's more useful URL manipulation functions. Pass it a Uniform Resource Locator (URL), and `parse_url()` will go to work splitting it into its individual components. The resulting associative array contains separate keys for the protocol, host name, port number, remote path, and `GET` arguments. You can then easily access and use these keys for further processing—for example, the variable `$data['host']` will return the value `www.melonfire.com`.

Consider the output of the previous script, which illustrates this:

```
scheme: http
host: www.melonfire.com
port: 80
path: /community/columns/trog/article.php
query: id=79&page=2
```

1.13 Counting Words in a String

Problem

You want to count the number of words in a sentence or paragraph.

Solution

Use a pattern to identify the individual words in the string, and then count how many times that pattern recurs:

```
<?php
// define string
$text = "Fans of the 1980 group will have little trouble recognizing ↵
the group's distinctive synthesized sounds and hypnotic dance beats,↵
since these two elements are present in almost every song on the ↵
album; however, the lack of diversity and range is troubling, and I'm ↵
hoping we see some new influences in the next album. More
intelligent lyrics might also help.";

// decompose the string into an array of "words"
$words  = preg_split('/[^0-9A-Za-z\']+/', $text, -1, ↵
PREG_SPLIT_NO_EMPTY);

// count number of words (elements) in array
// result: "59 words"
echo count($words) . " words";
?>
```

Comments

The `preg_split()` function is probably one of PHP's most underappreciated functions. This function accepts a Perl-compliant regular expression and a subject

string, and returns an array containing substrings matching the pattern. It's a great way of finding the matches in a string and placing them in a separate array for further processing. Read more about the function and its arguments at `http://www.php.net/preg_split`.

In this listing, the regular expression `[^0-9A-Za-z\']+` is a generic pattern that will match any word. All the words thus matched are fed into the `$words` array. Counting the number of words in the string is then simply a matter of obtaining the size of the `$words` array.

An alternative is to use the new `str_word_count()` function to perform this task. Here's an example:

```
<?php
// define string
$text = "Fans of the 1980 group will have little trouble recognizing ↵
the group's distinctive synthesized sounds and hypnotic dance beats,↵
since these two elements are present in almost every song on the ↵
album; however, the lack of diversity and range is troubling, and I'm
↵
hoping we see some new influences in the next album. More intelligent
lyrics might also help.";

// count number of words
// result: "58 words"
$numWords = str_word_count($text);
echo $numWords . " words";
?>
```

NOTE

Wondering about the discrepancy in the results above? The `str_word_count()` function ignores numeric strings when calculating the number of words.

1.14 Spell-Checking Words in a String

Problem

You want to check if one or more words are spelled correctly.

Solution

Use PHP's `ext/pspell` extension to check words against an internal dictionary:

```
<?php
// define string to be spell-checked
$str = "someun pleez helpp me i canot spel";

// check spelling
// open dictionary link
$dict = pspell_new("en", "british");

// decompose string into individual words
// check spelling of each word
$str = preg_replace('/[0-9]+/', '', $str);
$words = preg_split('/[^0-9A-Za-z\']+/', $str, -1, ↵
PREG_SPLIT_NO_EMPTY);
foreach ($words as $w) {
    if (!pspell_check($dict, $w)) {
        $errors[] = $w;
    }
}

// if errors exist
// print error list
if (sizeof($errors) > 0) {
    echo "The following words were wrongly spelt: " . ↵
implode(" ", $errors);
}
?>
```

> ***NOTE***
>
> *In order for this listing to work, PHP must be compiled with support for the `pspell` extension. (You can obtain instructions from the PHP manual at `http://www.php.net/pspell`.)*

Comments

The first task here is to identify the individual words in the sentence or paragraph. You accomplish this using the `preg_split()` function and regular expression previously discussed in the listing in the "1.13: Counting Words in a String" section. The `pspell_new()` function is used to open a link to the appropriate language dictionary, and the `pspell_check()` function iterates over the word list, checking each word

against the dictionary. For words that are incorrectly spelled, `pspell_check()` returns false; these words are flagged, placed in an array and displayed in a list once the process is complete.

With a little modification, you can have the previous listing check a file (rather than a variable) for misspelled words, and even offer suggestions when it encounters errors. Consider this variant, which illustrates the process and incorporates a call to `pspell_suggest()` to recommend alternatives for each wrongly-spelled word:

```
<?php
// define file to be spell-checked
$file = "badspelling.txt";

// check spelling
// open dictionary link
$dict = pspell_new("en", "british", "", "", PSPELL_FAST);

// open file
$fp = fopen ($file, 'r') or die ("Cannot open file $file");

// read file line by line
$lineCount = 1;
while ($line = fgets($fp, 2048)) {
    // clean up trailing whitespace
    $line = trim($line);

    // decompose line into individual words
    // check spelling of each word
    $line = preg_replace('/[0-9]+/', '', $line);
    $words = preg_split('/[^0-9A-Za-z\']+/', $line, -1, ↵
PREG_SPLIT_NO_EMPTY);

    foreach ($words as $w) {
        if (!pspell_check($dict, $w)) {
            if (!is_array($errors[$lineCount])) {
                $errors[$lineCount] = array();
            }
            array_push($errors[$lineCount], $w);
        }
    }
    $lineCount++;
}
```

```
// close file
fclose($fp);

// if errors exist
if (sizeof($errors) > 0) {
    // print error list, with suggested alternatives
    echo "The following words were wrongly spelt: \n";
    foreach ($errors as $k => $v) {
        echo "Line $k: \n";
        foreach ($v as $word) {
            $opts = pspell_suggest($dict, $word);
            echo "\t$word (" . implode(', ', $opts) . ")\n";
        }
    }
}
?>
```

NOTE

It's important to remember that `pspell_check()` returns false on numeric strings. This can result in numerous false positives if your string contains numbers by themselves. The previous listing works around this problem by removing all the number sequences from the string/file before passing it to `pspell_check()`.

1.15 Identifying Duplicate Words in a String

Problem

You want to identify words that appear more than once in a string.

Solution

Decompose the string into individual words, and then count the occurrences of each word:

```
<?php
// define string
$str = "baa baa black sheep";

// trim the whitespace at the ends of the string
$str = trim($str);
```

```
// compress the whitespace in the middle of the string
$str = ereg_replace('[[:space:]]+', ' ', $str);

// decompose the string into an array of "words"
$words  = explode(' ', $str);

// iterate over the array
// count occurrences of each word
// save stats to another array
foreach ($words as $w) {
    $wordStats[strtolower($w)]++;
}

// print all duplicate words
// result: "baa"
foreach ($wordStats as $k=>$v) {
    if ($v >= 2) { print "$k \r\n"; }
}
?>
```

Comments

The first task here is to identify the individual words in the sentence or paragraph. You accomplish this by compressing multiple spaces in the string, and then decomposing the sentence into words with `explode()`, using a single space as [the] delimiter. Next, a new associative array, `$wordStats`, is initialized and a key is created within it for every word in the original string. If a word occurs more than once, the value corresponding to that word's key in the `$wordStats` array is incremented by 1.

Once all the words in the string have been processed, the `$wordStats` array will contain a list of unique words from the original string, together with a number indicating each word's frequency. It is now a simple matter to isolate those keys with values greater than 1, and print the corresponding words as a list of duplicates.

1.16 Searching Strings

Problem

You want to search a string for a particular pattern or substring.

Solution

Use a regular expression with PHP's `ereg()` function:

```
<?php
// define string
$html = "I'm <b>tired</b> and so I <b>must</b> go
<a href='http://domain'>home</a> now";

// check for match
// result: "Match"
echo ereg("<b>(.*)+</b>", $html) ? "Match" : "No match";
?>
```

Use a regular expression with PHP's `preg_match()` function:

```
<?php
// define string
$html = "I'm <b>tired</b> and so I <b>must</b> go
<a href='http://domain'>home</a> now";

// check for match
// result: "Match"
echo preg_match("/<b>(.*?)<\/b>/i", $html) ? "Match" : "No match";
?>
```

Comments

When it comes to searching for matches within a string, PHP offers the `ereg()` and `preg_match()` functions, which are equivalent: both functions accept a regular expression and a string, and return true if the string contains one or more matches to the regular expression. Readers familiar with Perl will usually prefer the `preg_match()` function, as it enables them to use Perl-compliant regular expressions and, in some cases, is faster than the `ereg()` function.

TIP

For case-insensitive matching, use the `eregi()` function instead of the `ereg()` function.

TIP

Read more about regular expressions at `http://www.melonfire.com/community/columns/trog/article.php?id=2`.

1.17 Counting Matches in a String

Problem

You want to find out how many times a particular pattern occurs in a string.

Solution

Use PHP's `preg_match_all()` function:

```
<?php
// define string
$html = "I'm <b>tired</b> and so I <b>must</b> go
<a href='http://domain'>home</a> now";

// count occurrences of bold text in string
// result: "2 occurrence(s)"
preg_match_all("/<b>(.*?)<\/b>/i", $html, &$matches);
echo sizeof($matches[0]) . " occurrence(s)";
?>
```

Comments

The `preg_match_all()` function tests a string for matches to a particular pattern, and returns an array containing all the matches. If you need the total number of matches, simply check the size of the array with the `sizeof()` function.

For simpler applications, also consider the `substr_count()` function, which counts the total number of occurrences of a substring within a larger string. Here's a brief example:

```
<?php
// define string
$text = "ha ha ho hee hee ha ho hee hee ho ho ho ha hee";

// count occurrences of "hee " in string
// result: "5 occurrence(s)"
echo substr_count($text, "hee") . " occurrence(s)";
?>
```

1.18 Replacing Patterns in a String

Problem

You want to replace all/some occurrences of a pattern or substring within a string with something else.

Solution

Use a regular expression in combination with PHP's `str_replace()` function (for simple patters):

```
 <?php
// define string
$str = "Michael says hello to Frank";

// replace all instances of "Frank" with "Crazy Dan"
// result: "Michael says hello to Crazy Dan"
$newStr = str_replace("Frank", "Crazy Dan", $str);
echo $newStr;
?>
```

For more complex patters, use a regular expression in combination with PHP's `preg_replace()` function:

```
<?php
// define string
$html = "I'm <b>tired</b> and so I <b>must</b> go ↵
<a href='http://domain'>home</a> now";

// replace all bold text with italics
// result: "I'm <i>tired</i> and so I <i>must</i> go
<a href='http://domain'>home</a> now"
$newStr = preg_replace("/<b>(.*?)<\/b>/i", "<i>\\1</i>", $html);
echo $newStr;
?>
```

Comments

For simple applications that don't need complex pattern matching or regular expressions, consider PHP's `str_replace()` function. You can't use regular

expressions with this function—all it enables you to do is replace one (or more) substrings with one (or more) replacement strings. Although it's limited, it can be faster than either `ereg_replace()` or `preg_replace()` in situations which don't call for advanced expression processing.

PHP's `preg_replace()` function takes the `preg_match()` function a step forward—in addition to searching for regular expression matches in the target string, it can also replace each match with something else. The `preg_replace()` function accepts a Perl-compliant regular expression, and its return value is the original string after all substitutions have been made. If no matches could be found, the original string is returned. Note also the use of a *back-reference* (`\\1`) in the `preg_replace()` version of the listing; this back-reference serves as a placeholder for text enclosed within the pattern to be matched.

By default, both functions replace all occurrences of the search string with the replacement string. With `preg_replace()`, however, you can control the number of matches that are replaced by passing the function an optional fourth parameter. Consider the following snippet, which limits the number of replacements to 1 (even though there are two valid matches):

```
<?php
// define string
$html = "I'm <b>tired</b> and so I <b>must</b> go
       <a href='http://domain'>home</a> now";

// replace all bold text with italics
// result: "I'm <i>tired</i> and so I <b>must</b> go
       <a href='http://domain'>home</a> now"
$newStr = preg_replace("/<b>(.*?)<\/b>/i", "<i>\\1</i>", $html, 1);
echo $newStr;
?>
```

As an interesting aside, you can find out the number of substrings replaced by `str_replace()` by passing the function an optional fourth parameter, which counts the number of replacements. Here's an illustration:

```
<?php
// define string
$str = "Michael says hello to Frank. Frank growls at Michael. Michael ↵
feeds Frank a bone.";

// replace all instances of "Frank" with "Crazy Dan"
$newStr = str_replace("Frank", "Crazy Dan", $str, &$counter);
```

```
// print number of replacements
// result: "3 replacement(s)"
echo "$counter replacement(s)";
?>
```

TIP

You can perform multiple search-replace operations at once with `str_replace()`, by using arrays for both the search and replacement strings.

1.19 Extracting Substrings

Problem

You want to extract the substring preceding or following a particular match.

Solution

Use the `preg_split()` function to split the original string into an array delimited by the match term, and then extract the appropriate array element(s):

```
<?php
// define string
$html = "Just when you begin to think the wagon of ↵
<a name='#war'>Vietnam</a>-grounded movies is grinding to a slow halt, ↵
you're hit squarely in the <a name='#photo'>face</a> with another ↵
one. However, while other movies depict the gory and glory of war ↵
and its effects, this centers on the ↵
<a name='#subject'>psychology</a> of troopers before ↵
they're led to battle.";

// split on <a> element
$matches = preg_split("/<a(.*?)>(.*?)<\/a>/i", $html);

// extract substring preceding first match
// result: "Just when...of"
echo $matches[0];

// extract substring following last match
// result: "of troopers...battle."
echo $matches[sizeof($matches)-1];
?>
```

Comments

The `preg_split()` function accepts a regular expression and a search string, and uses the regular expression as a delimiter to split the string into segments. Each of these segments is placed in an array. Extracting the appropriate segment is then simply a matter of retrieving the corresponding array element.

This is clearly illustrated in the previous listing. To extract the segment preceding the first match, retrieve the first array element (index `0`); to extract the segment following the last match, retrieve the last array element.

If your match term is one or more regular words, rather than a regular expression, you can accomplish the same task more easily by `explode()`-ing the string into an array against the match term and extracting the appropriate array elements. The next listing illustrates this:

```
<?php
// define string
$str = "apples and bananas and oranges and pineapples and lemons";

// define search pattern
$search = " and ";

// split string into array
$matches = explode($search, $str);

// count number of segments
$numMatches = sizeof($matches);

// extract substring preceding first match
// result: "apples"
echo $matches[0];

// extract substring between first and fourth matches
// result: "bananas and oranges and pineapples"
echo implode($search, array_slice($matches, 1, 3));

// extract substring following last match
// result: "lemons"
echo $matches[$numMatches-1];
?>
```

1.20 Extracting Sentences from a Paragraph

Problem

You want to extract the first or last sentence from a paragraph.

Solution

Use the `strtok()` function to break the paragraph into sentences, and then extract the appropriate sentence:

```
<?php
// define string
$text = "This e-mail message was sent from a notification-only address! ↵
It cannot accept incoming e-mail. Please do not reply to this message. ↵
Do you understand?";

// extract first sentence
// result: "This e-mail message was sent from a notification-only ↵
address"
$firstSentence = strtok($text, ".?!");
echo $firstSentence;

// extract last sentence
// result: "Do you understand"
$lastSentence = strrev(strtok(strrev(trim($text)), ".?!"));
echo $lastSentence;
?>
```

Comments

To extract the first or last sentence of a paragraph, it is necessary to first break the string into individual sentences, using the common sentence terminators—a period, a question mark, and an exclamation mark—as delimiters. PHP's `strtok()` function is ideal for this: it splits a string into smaller segments, or tokens, based on a list of user-supplied delimiters. The first token obtained in this manner will be the first sentence of the paragraph.

Extracting the last sentence is a little more involved, and there are quite a few ways to do it. The previous listing uses one of the simplest: it reverses the paragraph

and extracts the last sentence as though it were the first, again using `strtok()`. The extracted segment is then re-reversed using the `strrev()` function.

1.21 Generating String Checksums

Problem

You want to obtain a hash signature for a string

Solution

Use PHP's `md5()` or `sha1()` functions:

```
 <?php
// define string
$str = "two meters north, five meters west";

// obtain MD5 hash of string
// result: "7c00dcc2a1e4e89133b849a003448788"
$md5 = md5($str);
echo $md5;

// obtain SHA1 hash of string
// result: "d5db0063b0e2d4d7d33514e2da3743ce8daa44bf"
$sha1 = sha1($str);
echo $sha1;
?>
```

Comments

A hash signature is a lot like a fingerprint—it uniquely identifies the source that was used to compute it. Typically, a hash signature is used to verify if two copies of a string or file are identical in all respects; if both produce the same hash signature, they can be assumed to be identical. A hash function, like PHP's `md5()` or `sha1()` function, accepts string input and produces a fixed-length signature (sometimes called a checksum) that can be used for comparison or encryption. The `md5()` function produces a 128-bit hash, while the `sha1()` function produces a 160-bit hash. Read more at `http://www.faqs.org/rfcs/rfc1321.html`.

1.22 Encrypting Strings (One-Way Encryption)

Problem

You want to encrypt a string using one-way encryption.

Solution

Use PHP's `crypt()` function:

```
<?php
// define cleartext string
$password = "guessme";

// define salt
$salt = "rosebud";

// encrypt string
// result: "rouuR6YmPKTOE"
$cipher = crypt($password, $salt);
echo $cipher;
?>
```

Comments

PHP's `crypt()` function accepts two parameters: the string to encrypt and a key (or salt) to use for encryption. It then encrypts the string using the provided salt and returns the encrypted string (or ciphertext). A particular combination of cleartext and salt is unique—the ciphertext generated by `crypt()`-ing a particular string with a particular salt remains the same over multiple `crypt()` invocations.

Because the `crypt()` function uses one-way encryption, there is no way to recover the original string from the ciphertext. You're probably wondering what use this is—after all, what's the point of encrypting something so that it can never be decrypted? Well, one-way encryption does have its uses, most notably for password verification: it's possible to validate a previously-encrypted password against a user's input by re-encrypting the input with the same salt and checking to see if the two pieces of ciphertext match. The next example illustrates this process:

```
<?php
// define cleartext string
$password = "guessme";
```

```
// define salt
$salt = "rosebud";

// encrypt string
$cipher = crypt($password, $salt);

// assume the user inputs this
$input = "randomguess";

// encrypt the input
// test it against the encrypted password
// result: "Passwords don't match"
echo ($cipher == crypt($input, $salt)) ? ↵
"Passwords match" : "Passwords don't match";

// now assume the user inputs this
$input = "guessme";

// encrypt the input
// test it against the encrypted password
// result: "Passwords match"
echo ($cipher == crypt($input, $salt)) ? ↵
"Passwords match" : "Passwords don't match";
?>
```

Here, the cleartext password is encrypted with PHP's `crypt()` function and the defined salt, with the result checked against the (encrypted) original password. If the two match, it indicates that the supplied password was correct; if they don't, it indicates that the password was wrong.

1.23 Encrypting Strings (Two-Way Encryption)

Problem

You want to encrypt a string using two-way encryption.

Solution

Use PHP's `ext/mcrypt` extension to perform two-way encryption or decryption:

```
<?php
// function to encrypt data
```

```
function encryptString($plaintext, $key) {
    // seed random number generator
    srand((double) microtime() * 1000000);

    // encrypt string
    $iv = mcrypt_create_iv( ↵
mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_CFB), ↵
MCRYPT_RAND);
    $cipher = mcrypt_encrypt(MCRYPT_BLOWFISH, $key, ↵
$plaintext, MCRYPT_MODE_CFB, $iv);

    // add IV to ciphertext
    return $iv . $cipher;
}

// function to decrypt data
function decryptString($ciphertext, $key) {
    // extract IV
    $iv = substr($ciphertext, 0,↵
mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_CFB));
    $cipher = substr($ciphertext, ↵
mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_CFB));

    // decrypt string
    return mcrypt_decrypt(MCRYPT_BLOWFISH, $key, $cipher,↵
MCRYPT_MODE_CFB, $iv);
}

// define cleartext string
$input = "three paces west, up the hill, turn nor-nor-west ↵
and fire through the left eye socket";

// define key
$key = "rosebud";

// returns encrypted string
$ciphertext = encryptString($input, $key);
echo $ciphertext;

// returns decrypted string
$cleartext = decryptString($ciphertext, $key);
echo $cleartext;
?>
```

NOTE

In order for this listing to work, PHP must be compiled with support for the `mcrypt` *extension (you can obtain instructions from the PHP manual at* `http://www.php.net/mcrypt`*).*

Comments

The previous listing uses two user-defined functions: `encryptString()` and `decryptString()`. Internally, both use functions provided by PHP's `ext/mcrypt` extension, which supports a wide variety of encryption algorithms (Blowfish, DES, TripleDES, IDEA, Rijndael, Serpent, and others) and cipher modes (CBC, CFB, OFB, and ECB). Both functions accept a string and a key, and use the latter to encrypt or decrypt the former.

The `encryptString()` function begins by seeding the random number generator and then generating an initialization vector (IV) with the `mcrypt_create_iv()` function. Once an IV has been generated, the `mcrypt_encrypt()` function performs the encryption using the supplied key. The encryption in this example uses the Blowfish algorithm in CFB mode. The IV is prepended to the encrypted string; this is normal and does not affect the security of the encryption.

The `decryptString()` function words in reverse, obtaining the IV size for the selected encryption algorithm and mode with the `mcrypt_get_iv_size()` function and then extracting the IV from the beginning of the encrypted string with the `substr()` function. The IV, encrypted string, and key are then used by the `mcrypt_decrypt()` function to retrieve the original cleartext string.

Read more about encryption algorithms and modes at `http://en.wikipedia.org/wiki/Encryption_algorithm`.

1.24 Generating Pronounceable Passwords

Problem

You want to generate a pronounceable password.

Solution

Use PEAR's `Text_Password` class:

```
<?php
// include Text_Password class
include "Text/Password.php";
```

```
// create object
$tp = new Text_Password();

// generate pronounceable password
// result: "sawralaeje" (example)
$password = $tp->create();
echo $password;
?>
```

Comments

If you're looking for a quick way to generate pronounceable passwords—perhaps for a Web site authentication system—look no further than the PEAR `Text_Password` class (available from `http://pear.php.net/package/Text_Password`). By default, the class method `create()` generates a ten-character pronounceable password using only vowels and consonants.

You can define a custom length for the password by passing an optional size argument to the `create()` method, as follows:

```
<?php
// include Text_Password class
include "Text/Password.php";

// create object
$tp = new Text_Password();

// generate 5-character pronounceable password
// result: "ookel" (example)
$password = $tp->create(5);
echo $password;
?>
```

1.25 Generating Unpronounceable Passwords

Problem

You want to generate an unpronounceable password.

Solution

Use PEAR's `Text_Password` class with some additional parameters:

```
<?php
// include Text_Password class
include "Text/Password.php";

// create object
$tp = new Text_Password();

// generate 7-character unpronounceable password
// result: "_nCx&h#" (example)
$password = $tp->create(7, 'unpronounceable');
echo $password;
?>
```

Comments

The PEAR `Text_Password` class (available from `http://pear.php.net/package/Text_Password`) is designed specifically to generate both pronounceable and unpronouceable passwords of varying lengths. To generate an unpronounceable password made up of alphabets, numbers, and special characters, call the class method `create()` with two additional flags: the desired password size and the keyword `unpronounceable` (the default behavior is to generate pronounceable passwords ten characters long).

If you'd like to restrict the characters that can appear in the password, you can pass the `create()` method a third argument: either of the keywords `'numeric'` or `'alphanumeric'`, or a comma-separated list of allowed characters. The following code snippets illustrate this:

```
<?php
// include Text_Password class
include "Text/Password.php";

// create object
$tp = new Text_Password();

// generate 7-character unpronounceable password
// using only numbers
// result: "0010287" (example)
$password = $tp->create(7, 'unpronounceable', 'numeric');
echo $password;

?>
```

```
<?php
// include Text_Password class
include "Text/Password.php";

// create object
$tp = new Text_Password();

// generate 12-character unpronounceable password
// using only letters and numbers
// result: "P44g62gk6YIp" (example)
$password = $tp->create(12, 'unpronounceable', 'alphanumeric');
echo $password;
?>

<?php
// include Text_Password class
include "Text/Password.php";

// create object
$tp = new Text_Password();

// generate 5-character unpronounceable password
// using a pre-defined character list
// result: "okjnn" (example)
$password = $tp->create(5, 'unpronounceable', 'i,j,k,l,m,n,o,p');
echo $password;
?>
```