

CHAPTER 3

MANIPULATING STRINGS

PHP PROGRAMMING WITH MYSQL
2ND EDITION

Objectives

2

In this chapter, you will:

- ❑ Construct text strings
- ❑ Work with single strings
- ❑ Work with multiple strings and parse strings
- ❑ Compare strings
- ❑ Use regular expressions

Constructing Text Strings

3

- A text string contains zero or more characters surrounded by double or single quotation marks
- Text strings can be used as literal values or be assigned to a variable

```
echo "<p>PHP literal text string</p>";  
$StringVariable = "<p>PHP string variable</p>";  
echo $StringVariable;
```

- A string must begin and end with a matching quotation mark (single or double)

Constructing Text Strings (continued)

- ❑ To include a quoted string within a literal string surrounded by double quotation marks, you surround the quoted string with single quotation marks
- ❑ To include a quoted string within a literal string surrounded by single quotation marks, you surround the quoted string with double quotation marks

Constructing Text Strings (continued)

5

```
$LatinQuote = '<p>"Et tu, Brute!"</p>';  
echo $LatinQuote;
```

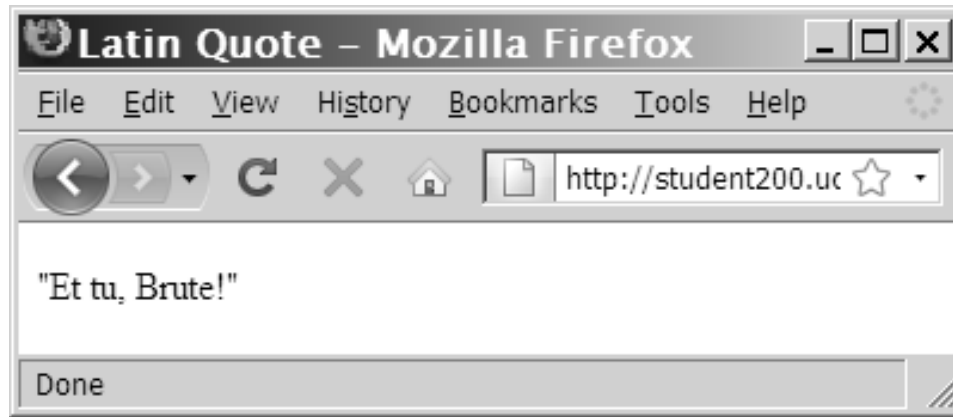


Figure 3-2 Output of a text string containing double quotation marks

Working with String Operators

6

In PHP, you use two operators to combine strings:

- **Concatenation operator (.)** combines two strings and assigns the new value to a variable

```
$City = "Paris";  
$Country = "France";  
$Destination = "<p>" . $City . " is in "  
                . $Country . "</p>";  
echo $Destination;
```

Working with String Operators (continued)

7

- You can also combine strings using the **concatenation assignment operator (.=)**

```
$Destination = "<p>Paris";  
$Destination .= "is in France.</p>";  
echo $Destination;
```

Adding Escape Characters and Sequences

8

- An **escape character** tells the compiler or interpreter that the character that follows it has a special purpose
- In PHP, the escape character is the backslash (\)
`echo '<p>This code\'s going to work</p>';`
- There's no need for a backslash before an apostrophe if you surround the text string with double quotation marks
`echo "<p>This code's going to work.</p>";`

Adding Escape Characters and Sequences (continued)

9

- The escape character combined with one or more other characters is an **escape sequence**

| Escape Sequence | Description |
|-----------------|--|
| \\ | Inserts a backslash |
| \\$ | Inserts a dollar sign |
| \r | Inserts a carriage return |
| \f | Inserts a form feed |
| \" | Inserts a double quotation mark |
| \t | Inserts a horizontal tab |
| \v | Inserts a vertical tab |
| \n | Inserts a new line |
| \x <h></h> | Inserts a character whose hexadecimal value is <i>h</i> , where <i>h</i> is one or two hexadecimal digits (0-9, A-F), case insensitive |
| \o | Inserts a character whose octal value is <i>o</i> , where <i>o</i> is one, two, or three octal digits (0-7) |

Table 3-1 PHP escape sequences within double quotation marks

Adding Escape Characters and Sequences (continued)

10

```
$Speaker = "Julius Caesar";  
echo "<p>\"Et tu, Brute!\" exclaimed  
$Speaker.</p>";
```



Figure 3-4 Output of literal text containing double quotation escape sequences

Simple and Complex String Syntax

11

- **Simple string syntax** uses the value of a variable within a string by including the variable name inside a text string with double quotation marks

```
$Vegetable = "broccoli";
```

```
echo "<p>Do you have any $Vegetable?</p>";
```

- When variables are placed within curly braces inside of a string, it is called **complex string syntax**

```
$Vegetable = "carrot";
```

```
echo "<p>Do you have any {$Vegetable}s?</p>";
```

Short Quiz, p. 133

12

1. Explain the difference between a concatenation operator and a concatenation assignment operator.
 - a. the assignment operator appends to a variable where a concatenation operator is part of the variable declaration.
2. Describe two ways to display double quotations marks within a literal string.
 - a. `' "stuff" '`
3. Describe the use of curly braces in complex string syntax.
 - a. to differentiate variable names in a string especially in the use of plurality.

Working with a Single String

13

- PHP provides a number of functions for analyzing, altering, and parsing text strings including:
 - ▣ Counting characters and words
 - ▣ Transposing, converting, and changing the case of text within a string

Counting Characters and Words in a String

14

- The most commonly used string counting function is the `strlen()` function, which returns the total number of characters in a string
- Escape sequences, such as `\n`, are counted as one character

```
$BookTitle = "The Cask of Amontillado";  
echo "<p>The book title contains " .  
    strlen($BookTitle) . " characters.</p>";
```

Counting Characters and Words in a String (continued)

15

- The `str_word_count()` function returns the number of words in a string
- Pass the `str_word_count()` function a literal string or the name of a string variable whose words you want to count

```
$BookTitle = "The Cask of Amontillado";  
echo "<p>The book title contains " .  
    str_word_count($BookTitle) . " words.</p>";
```

Modifying the Case of a String

16

- PHP provides several functions to manipulate the case of a string
 - ▣ The `strtoupper()` function converts all letters in a string to uppercase
 - ▣ The `strtolower()` function converts all letters in a string to lowercase
 - ▣ The `ucfirst()` function ensures that the first character of a word is uppercase
 - ▣ The `lcfirst()` function ensures that the first character of a word is lowercase

Modifying the Case of a String (continued)

17

- Functions to manipulate the case of a string:
 - The `ucwords()` function changes the first character of each word
 - Use the `strtolower()` function on a string before using the `ucfirst()` and `ucwords()` to ensure that the remaining characters in a string are in lowercase
 - Use the `strtoupper()` function on a string before using the `lcfirst()` and `lcwords()` to ensure that the remaining characters in a string are in uppercase

Encoding and Decoding a String

18

- PHP has several built-in functions to use with Web pages:
- Some characters in XHTML have a special meaning and must be encoded using HTML entities in order to preserve that meaning
 - ▣ The `htmlspecialchars()` function converts special characters to HTML entities
 - ▣ The `htmlspecialchars_decode()` function converts HTML character entities into their equivalent characters

Encoding and Decoding a String (continued)

19

- The characters that are converted with the `htmlspecialchars()` function are:
 - `'&'` (ampersand) becomes `'&'`
 - `'"'` (double quote) becomes `'"'` when `ENT_NOQUOTES` is disabled.
 - `'''` (single quote) becomes `'''` only when `ENT_QUOTES` is enabled.
 - `'<'` (less than) becomes `'<'`
 - `'>'` (greater than) becomes `'>'`

Encoding and Decoding a String (continued)

20

- If `ENT_QUOTES` is enabled in the PHP configuration, both single and double quotes are converted
- If `ENT_QUOTES` is disabled in the PHP configuration, neither single nor double quotes are converted

Encoding and Decoding a String (continued)

21

- The `md5 ()` function uses a strong encryption algorithm (called the Message-Digest Algorithm) to create a one-way hash
 - ▣ A **one-way hash** is a fixed-length string based on the entered text, from which it is nearly impossible to determine the original text
 - ▣ The `md5 ()` function does not have an equivalent decode function, which makes it a useful function for storing passwords in a database

Other Ways to Manipulate a String

22

- PHP provides three functions that remove leading or trailing spaces in a string
 - ▣ The `trim()` function will strip (remove) leading or trailing spaces in a string
 - ▣ The `ltrim()` function removes only the leading spaces
 - ▣ The `rtrim()` function removes only the trailing spaces

Other Ways to Manipulate a String (continued)

23

- The `substr()` function returns part of a string based on the values of the `start` and `length` parameters
- The syntax for the `substr()` function is:

```
substr(string, start, optional length);
```

- A positive number in the `start` parameter indicates how many characters to skip at the beginning of the string
- A negative number in the `start` parameter indicates how many characters to count in from the end of the string

Other Ways to Manipulate a String (continued)

24

- A positive value in the `length` parameter determines how many characters to return
- A negative value in the `length` parameter skip that many characters at the end of the string and returns the middle portion
- If the `length` is omitted or is greater than the remaining length of the string, the entire remainder of the string is returned

Other Ways to Manipulate a String (continued)

25

```
$ExampleString = "woodworking project";  
echo substr($ExampleString,4) . "<br />\n";  
echo substr($ExampleString,4,7) . "<br />\n";  
echo substr($ExampleString,0,8) . "<br />\n";  
echo substr($ExampleString,-7) . "<br />\n";  
echo substr($ExampleString,-12,4) . "<br />\n";
```

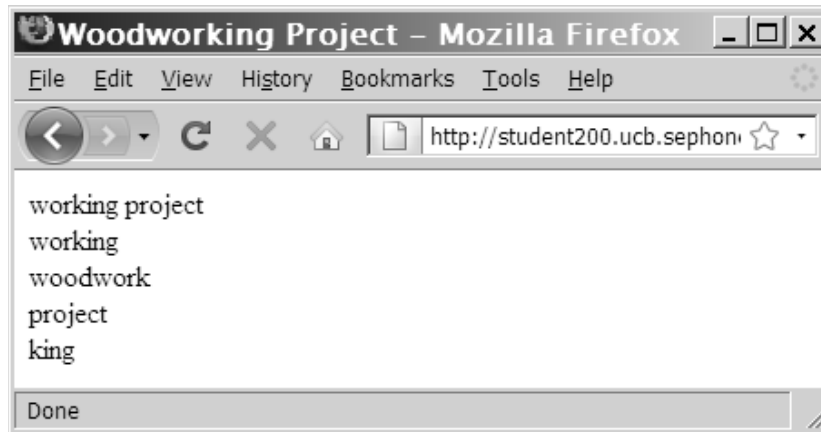


Figure 3-10 Some examples using the `substr()` function

Short Quiz, p. 141

26

1. What string function would you use to determine the number of characters in a password that user has entered?
2. What string function would you use to determine if an essay keyed in a `<textarea>` form input field exceeds the maximum number of words allowed?
3. What two string functions could be used to convert the case of text strings to all uppercase or all lowercase?

Working with Multiple Strings

27

- **Parsing** is the act of dividing a string into logical component substrings or tokens
- At the programming level, parsing refers to the extraction of information from string literals and variables

Finding and Extracting Characters and Substrings

28

- There are 2 types of string search and extraction functions:
 1. Functions that return a numeric position in a text string
 2. Functions that return a character or substring
- Both functions return a value of `FALSE` if the search string is not found

Finding and Extracting Characters and Substrings (continued)

29

- The `strpos()` function performs a case-sensitive search and returns the position of the first occurrence of one string in another string
- Pass two arguments to the `strpos()` function:
 - ▣ The first argument is the string you want to search
 - ▣ The second argument contains the characters for which you want to search
- If the search string is not found, the `strpos()` function returns a Boolean value of `FALSE`

Finding and Extracting Characters and Substrings (continued)

30

- To determine whether the `strpos()` function (and others) actually return a Boolean `FALSE` value and not a `0` representing the first character in a string, you must use the strict equal operator (`===`) or the strict not equal operator (`!==`)
- Example, p. 142, 143.

Finding and Extracting Characters and Substrings (continued)

31

- Pass to the `strchr()` and the `strrchr()` functions the string and the character for which you want to search
- Both functions return a substring from the specified characters to the end of the string
- `strchr()` function starts searching at the beginning of a string
- `strrchr()` function starts searching at the end of a string

Replacing Characters and Substrings

32

- **The `str_replace()` and `str_ireplace()` functions both accept three arguments:**
 - The string you want to search for
 - A replacement string
 - The string in which you want to replace characters

```
$Email = "president@whitehouse.gov";  
$NewEmail = str_replace("president", "vice.president", $Email);  
echo $NewEmail; // prints 'vice.president@whitehouse.gov'
```


Replacing Characters and Substrings

33

- The `substr_replace()` replaces characters within a specified portion of string - accept four arguments:
 - The string you want to search for
 - A replacement string
 - The starting and ending positions of the characters you want to replace
- If you do not include the last argument, the `substr_replace()` function replaces all the characters from the starting position to the end of the string

Dividing Strings into Smaller Pieces

34

- Use the `strtok()` function to break a string into smaller strings, called **tokens**
- The syntax for the **`strtok()`** function is:

```
$variable = strtok(string, separators);  
$variable = strtok(separators);
```
- The `strtok()` function returns the entire string if:
 - An empty string is specified as the second argument of the `strtok()` function
 - The string does not contain any of the separators specified

Dividing Strings into Smaller Pieces (continued)

35

```
$Presidents = "George Washington;John Adams;  
    Thomas Jefferson;James Madison;James Monroe";  
$President = strtok($Presidents, ";");  
while ($President != NULL) {  
    echo "$President<br />";  
    $President = strtok(";");  
}
```

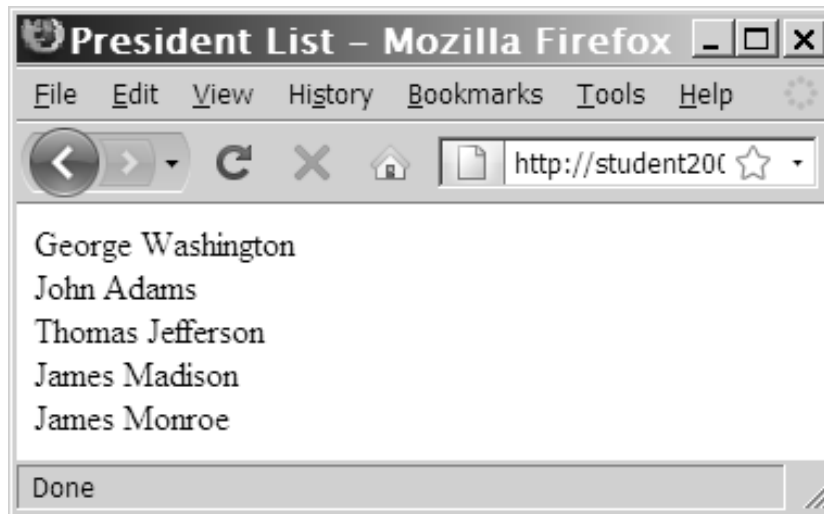


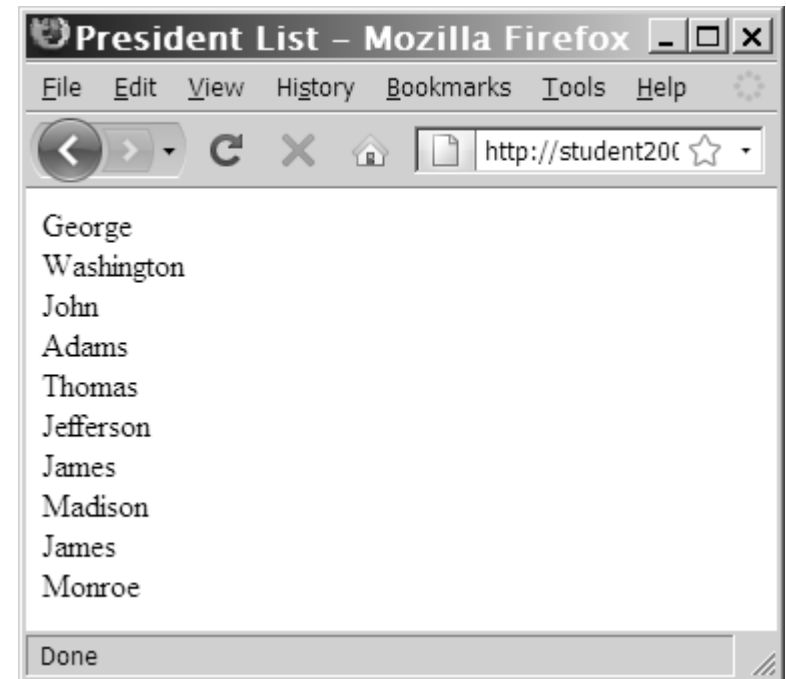
Figure 3-15 Output of a script that uses the `strtok()` function

Dividing Strings into Smaller Pieces(continued)

36

```
$Presidents = "George Washington;John Adams;Thomas  
Jefferson;James Madison;James Monroe";  
$President = strtok($Presidents, "; ");  
while ($President != NULL) {  
    echo "$President<br />";  
    $President = strtok("; ");  
}
```

**Figure 3-16 Output of a script
with a `strtok()` function
that uses two separators**



Converting between Strings and Arrays

37

- The `str_split()` and `explode()` functions split a string into an indexed array
- The `str_split()` function splits each character in a string into an array element using the syntax:

```
$array = str_split(string[, length]);
```

- The *length* argument represents the number of characters you want assigned to each array element

Converting between Strings and Arrays (continued)

38

- The `explode()` function splits a string into an indexed array at a specified separator

- The syntax for the `explode()` function is:

```
$array = explode(separators, string);
```

- The order of the arguments for the `explode()` function is the reverse of the arguments for the `strtok()` function

Converting between Strings and Arrays (continued)

39

```
$Presidents = "George Washington;John Adams;  
    Thomas Jefferson;James Madison;James Monroe";  
$PresidentArray = explode(";", $Presidents);  
foreach ($PresidentArray as $President) {  
    echo "$President<br />";  
}
```

Converting between Strings and Arrays (continued)

40

- The `explode()` function
 - Does not separate a string at each character that is included in the *separator* argument
 - Evaluates the characters in the *separator* argument as a substring
 - If you pass to the `explode()` function an empty string as the *separator* argument, the function returns a Boolean value of `FALSE`
 - If the string does not contain the specified separators, the entire string is assigned to the first element of the array

Converting between Strings and Arrays (continued)

41

- The `implode()` function combines an array's elements into a single string, separated by specified characters
- The syntax is:

```
$variable = implode(separators, array);
```

Converting between Strings and Arrays (continued)

42

```
$PresidentsArray = array("George Washington", "John Adams",  
"Thomas Jefferson", "James Madison", "James Monroe");  
$Presidents = implode(", ", $PresidentsArray);  
echo $Presidents;
```



Figure 3-18 Output of a string created with the `implode()` function

Short Quiz, p. 153

43

1. What function can be used to determine if a specific character exists in a string?
2. What is the difference between the `str_replace()` function and the `str_ireplace()` function?
3. What functions are used to split a string into an indexed array?

Comparing Strings

44

- Comparison operators compare individual characters by their position in the **American Standard Code for Information Interchange (ASCII)**, which are numeric representations of English characters

```
$FirstLetter = "A";  
$SecondLetter = "B";  
if ($SecondLetter > $FirstLetter)  
    echo "<p>The second letter is higher in the alphabet  
        than the first letter.</p>";  
else  
    echo "<p>The second letter is lower in the alphabet than  
        The first letter.</p>";
```

Comparing Strings (continued)

45

- American Standard Code for Information Interchange (ASCII) values range from 0 to 255
- Lowercase letters are represented by the values 97 (“a”) to 122 (“z”)
- Uppercase letters are represented by the values 65 (“A”) to 90 (“Z”)

String Comparison Functions

46

- ❑ The `strcasecmp()` function performs a case-insensitive comparison of strings
- ❑ The `strcmp()` function performs a case-sensitive comparison of strings
- ❑ Both functions accept two arguments representing the strings you want to compare
- ❑ Most string comparison functions compare strings based on their ASCII values

Determining the Similarity of Two Strings

47

- The `similar_text()` and `levenshtein()` functions are used to determine the similarity between two strings
- The `similar_text()` function returns the number of characters that two strings have in common
- The `levenshtein()` function returns the number of characters you need to change for two strings to be the same

Determining the Similarity of Two Strings (continued)

48

- Both functions accept two string arguments representing the values you want to compare

```
$FirstName = "Don";  
$SecondName = "Dan";  
echo "<p>The names \"$FirstName\" and \"$SecondName\" have " .  
similar_text($FirstName, $SecondName) . " characters in  
common.</p>";  
echo "<p>You must change " . levenshtein($FirstName, $SecondName)  
. " character(s) to make the names \"$FirstName\" and  
\"$SecondName\" the same.</p>";
```


Determining the Similarity of Two Strings (continued)

49

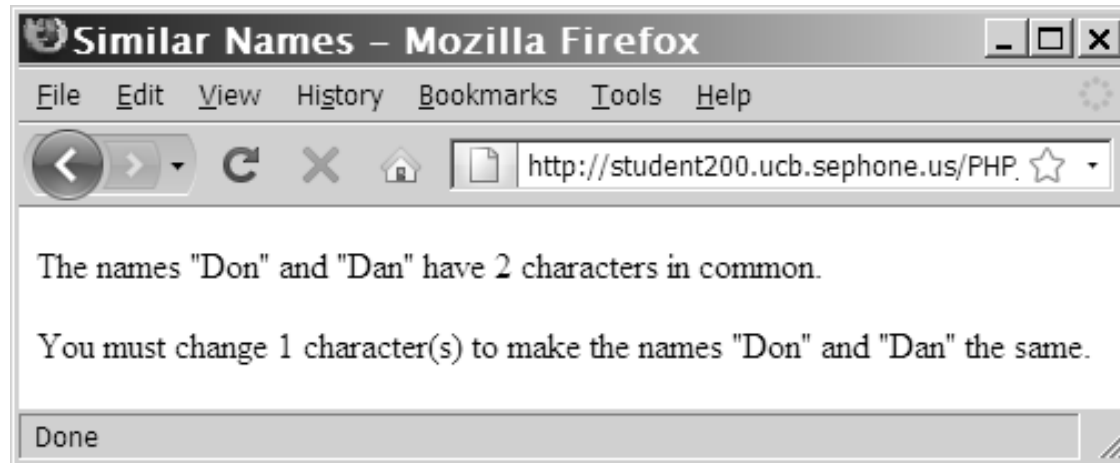


Figure 3-20 Output of a script with the `similar_text()` and `levenshtein()` functions

Determining if Words are Pronounced Similarly

50

- The `soundex()` and `metaphone()` functions determine whether two strings are pronounced similarly
- Both functions return a value representing how words sound
- The `soundex()` function returns a value representing a name's phonetic equivalent
- The `metaphone()` function returns a code representing an English word's approximate sound

Determining if Words are Pronounced Similarly (continued)

51

```
$FirstName = "Gosselin";  
$SecondName = "Gauselin";  
$FirstNameSoundsLike = metaphone($FirstName);  
$SecondNameSoundsLike = metaphone($SecondName);  
  
if ($FirstNameSoundsLike == $SecondNameSoundsLike)  
    echo "<p>The names are pronounced the same.</p>";  
else  
    echo "<p>The names are not pronounced the same.</p>";
```

Short Quiz, p. 159

52

1. What is the difference between the `strcasecmp()` function and the `strcmp()` function?
2. Why is the lowercase “a” considered to occur later in the alphabet than the uppercase “A”?
3. Explain the difference between the `similar_text()` function and the `levenshtein()` function?

Working with Regular Expressions

53

- **Regular Expressions** are patterns that are used for matching and manipulating strings according to specified rules
- PHP supports two types of regular expressions:
 - ▣ POSIX Extended – deprecated in PHP 5.3 and has no support in PHP 6.0
 - ▣ Perl Compatible Regular Expressions (PCRE)

Working with Regular Expressions (continued)

54

| Function | Description |
|---|---|
| <code>preg_match(<i>pattern</i>, <i>string</i>)</code> | Performs a search for a matching pattern |
| <code>preg_match_all(<i>pattern</i>, <i>string</i>)</code> | Performs a search for a matching pattern, returns the number of matches found |
| <code>preg_replace(<i>pattern</i>, <i>replacement</i>, <i>string</i> [, <i>limit</i>])</code> | Performs a replacement of a matching pattern |
| <code>preg_split(<i>pattern</i>, <i>string</i> [, <i>limit</i>])</code> | Divides an input string into an array of strings that are separated by a specified matching pattern |
| <code>preg_grep(<i>pattern</i>, <i>array</i>)</code> | Filters an input array and returns an array of those elements that match the specified pattern |
| <code>preg_quote(<i>string</i>)</code> | Returns a string that is the input string with any character that has special meaning for a PCRE preceded by the escape character (\) |

Table 3-2 PCRE functions

Working with Regular Expressions (continued)

55

- Pass to the `preg_match()` the regular expression pattern as the first argument and a string containing the text you want to search as the second argument

```
preg_match(pattern, string);
```

Writing Regular Expression Patterns

56

- A **regular expression pattern** is a special text string that describes a search pattern
- Regular expression patterns consist of literal characters and **metacharacters**, which are special characters that define the pattern-matching rules
- Regular expression patterns are enclosed in opening and closing **delimiters**
 - The most common character delimiter is the forward slash (/)

Writing Regular Expression Patterns (continued)

57

| Metacharacter | Description |
|---------------|--|
| . | Matches any single character |
| \ | Identifies the next character as a literal value |
| ^ | Anchors characters to the beginning of a string |
| \$ | Anchors characters to the end of a string |
| () | Specifies required characters to include in a pattern match |
| [] | Specifies alternate characters allowed in a pattern match |
| [^] | Specifies characters to exclude in a pattern match |
| - | Identifies a possible range of characters to match |
| | Specifies alternate sets of characters to include in a pattern match |

Table 3-3 PCRE metacharacters

Matching Any Character

58

- A period (.) in a regular expression pattern specifies that the pattern must contain a value at the location of the period
- A return value of 0 indicates that the string does not match the pattern and 1 if it does

```
$ZIP = "015";
```

```
preg_match("/...../", $ZIP); // returns 0
```

```
$ZIP = "01562";
```

```
preg_match("/...../", $ZIP); // returns 1
```

Matching Characters at the Beginning or End of a String (continued)

59

- An **anchor** specifies that the pattern must appear at a particular position in a string
- To specify an anchor at the beginning of a string, the pattern must begin with a **^** metcharacter

```
$URL = "http://www.dongosselin.com";  
preg_match("^http", $URL); // returns 1;
```

- To specify an anchor at the end of a line, the pattern must end with the **\$** metacharacter

```
$Identifier = "http://www.dongosselin.com";  
preg_match("com$", $Identifier); // returns 1
```

Matching Special Characters

60

- To match any metacharacters as literal values in a regular expression, escape the character with a backslash

(in the following example, the last four characters in the string must be '.com')

```
$Identifier = http://www.dongosselin,com";  
preg_match("/.com$/", $Identifier); //returns 1  
  
$Identifier = http://www.dongosselin,com";  
preg_match("/\.  
com$/", $Identifier); //returns 0
```

Specifying Quantity

61

- Metacharacters that specify the quantity of a match are called **quantifiers**

| Quantifier | Description |
|---------------------------|---|
| ? | Specifies that the preceding character is optional |
| + | Specifies that one or more of the preceding characters must match |
| * | Specifies that zero or more of the preceding characters can match |
| { <i>n</i> } | Specifies that the preceding character repeat exactly <i>n</i> times |
| { <i>n</i> , } | Specifies that the preceding character repeat at least <i>n</i> times |
| {, <i>n</i> } | Specifies that the preceding character repeat up to <i>n</i> times |
| { <i>n1</i> , <i>n2</i> } | Specifies that the preceding character repeat at least <i>n1</i> times but no more than <i>n2</i> times |

Table 3-4 PCRE quantifiers

Specifying Quantity (continued)

62

- A question mark (?) quantifier specifies that the preceding character in the pattern is optional (in the following example, the string must begin with 'http' or 'https')

```
$URL = "http://www.dongosselin.com";  
preg_match("/^http[s?]/", $URL); // returns 1
```

Specifying Quantity (continued)

63

- The addition (+) quantifier specifies that one or more sequential occurrences of the preceding characters match

(in the following example, the string must have at least one character)

```
$Name = "Don";  
preg_match("/.+/", $Name); // returns 1
```

Specifying Quantity (continued)

64

- A asterisk (*) quantifier specifies that zero or more sequential occurrences of the preceding characters match

(in the following example, the string must begin with zero or more leading zeros)

```
NumberString = "00125";  
preg_match("/^0*/", $NumberString); //returns 1
```


Specifying Quantity (continued)

65

- The `{ }` quantifiers specify the number of times that a character must repeat sequentially

(in the following example, the string must contain at least five characters)

```
preg_match("/ZIP: .{5}$/", "ZIP: 01562");  
// returns 1
```

- The `{ }` quantifiers can also specify the quantity as a range (in the following example, the string must contain between five and ten characters)

```
preg_match("/(ZIP: .{5,10})$/", "ZIP: 01562-2607");  
// returns 1
```

Specifying Subexpressions

66

- When a set of characters enclosed in parentheses are treated as a group, they are referred to as a **subexpression** or **subpattern**
(in the example below, the 1 and the area code are optional, but if included must be in the following format:)

1 (707) 555-1234

```
preg_match("/^(1 )?( \(.{3}\) )?(.{3}) (\-.{4}) $/
```

Defining Character Classes

67

- ❑ **Character classes** in regular expressions treat multiple characters as a single item
- ❑ Characters enclosed with the (`[]`) metacharacters represent alternate characters that are allowed in a pattern match

```
preg_match("/analy[sz]e/", "analyse");//returns 1  
preg_match("/analy[sz]e/", "analyze");//returns 1  
preg_match("/analy[sz]e/", "analyce");//returns 0
```

Defining Character Classes (continued)

68

- The hyphen metacharacter (–) specifies a range of values in a character class
[A-Z], [a-z], [A-Za-z], [0-9], [0-9A-Za-z]

(the following example ensures that A, B, C, D, or F are the only values assigned to the `$LetterGrade` variable)

```
$LetterGrade = "B";  
echo preg_match("[A-DF]", $LetterGrade);  
// returns 1
```

Defining Character Classes (continued)

69

- The `^` metacharacter (placed immediately after the opening bracket of a character class) specifies optional characters to **exclude** in a pattern match (the following example excludes the letter E and G-Z from an acceptable pattern match in the `$LetterGrade` variable)

```
$LetterGrade = "A";
```

```
echo preg_match("[^EG-Z]", $LetterGrade); // returns 1
```

Defining Character Classes (continued)

70

```
echo preg_match("/^(1 )?(\\([0-9]{3}\\) )?([0-9]{3})  
\\-[0-9]{4})$/", "1 (707) 555-1234");  
// returns 1
```

```
echo preg_match("/^[_a-z0-9-]+(\\.[_a-z0-9-]+)*@[_a-z0-9-  
]+(\\.[_a-z0-9-]+)*\\.([a-z]{2,3})$/i", $Email);
```

Defining Character Classes

(continued)

71

- ❑ To include a **literal hyphen** (`-`) in a character class, it must be the final character before the closing bracket, otherwise, it is interpreted as a range indicator.
- ❑ To include a **literal circumflex** (`^`), it must be the final character before the closing bracket or the literal hyphen.
- ❑ To include a **literal closing bracket** (`]`), it must be the first character after the opening bracket or negation symbol.

Defining Character Classes

(continued)

72

- PCRE includes special character types that you can use to represent different types of data.

| Escape Sequence | Description |
|-----------------|---|
| \a | alarm (hex 07) |
| \cx | “control-x”, where x is any character |
| \d | any decimal digit |
| \D | any character not in \d |
| \e | escape (hex 1B) |
| \f | formfeed (hex 0C) |
| \h | any horizontal whitespace character |
| \H | any character not in \h |
| \n | newline (hex 0A) |
| \r | carriage return (hex 0D) |
| \s | any whitespace character |
| \S | any character not in \s |
| \t | tab (hex 09) |
| \v | any vertical whitespace character |
| \V | any character not in \v |
| \w | any letter, number, or underscore character |
| \W | any character not in \w |

Table 3-5 PCRE character types

PHPEmail.php, p. 170

73

- `echo preg_match("/^[_a-z0-9-]+(\\.[_a-z0-9-]+)*@[_a-z0-9-]+(\\.[_a-z0-9-]+)* (\\.[a-z]{2,3})$/i", "$Email");`
- `echo preg_match("/^\\w-+(\\.\\w-+)*@[\\w-]+(\\.\\w-+)* (\\.[a-z]{2,3})$/i", $Email);`

Matching Multiple Pattern Choices

74

- The `|` metacharacter is used to specify an alternate set of patterns
 - ▣ The `|` metacharacter is essentially the same as using the OR operator to perform multiple evaluations in a conditional expression

```
echo preg_match("/\.(com|org|net)$/i",  
"http://www.dongosselin.gov");  
// returns 0
```

Pattern Modifiers

75

- **Pattern modifiers** are letters placed after the closing delimiter that change the default rules for interpreting matches
 - ▣ The pattern modifier, `i`, indicates that the case of the letter does not matter when searching
 - ▣ The pattern modifier, `m`, allows searches across newline characters
 - ▣ The pattern modifier, `s`, changes how the `.` (period) metacharacter works

Short Quiz, p. 170-171

76

1. What character is used to match any single character in a pattern?
2. How do you specify that you want to ensure that a string contains an actual period and not just any character?
3. Describe the purpose of quantifiers in determining if a string matches a pattern?
4. How are subexpressions or subpatterns used in a regular expression?
5. Describe the purpose of the `|` metacharacter.

Summary

77

- The **concatenation operator** (.) and the **concatenation assignment operator** (. =) can be used to combine two strings
- An **escape character** tells the compiler or interpreter that the character following the escape character has a special purpose. An escape character combined with one or more other characters is called an **escape sequence**

Summary (continued)

78

- ❑ **Simple string syntax** allows you to use the value of a variable within a string by including the variable name inside a text string with double quotation marks
- ❑ The type of structure in which variables are placed within curly braces inside of a string is called **complex string syntax**
- ❑ The most commonly used string-counting function is the `strlen()` function, which returns the total number of characters in a string

Summary (continued)

79

- The `str_word_count()` function returns the number of words in a string
- The `strtoupper()`, `strtolower()`, `ucfirst()`, `lcfirst()`, and `ucwords()` functions all change the case of characters in the string
- The `substr()` function returns the specified portion of a string

Summary (continued)

80

- When applied to text strings, **parsing** refers to the act of dividing a string into logical component substrings or **tokens**
- There are two types of string search and extraction functions: functions that return a numeric position in a text string and those that return a character or substring
- You use the `str_replace()`, `str_ireplace()`, and `substr_replace()` functions to replace text in strings

Summary (continued)

81

- The `strtok()` function breaks a string into smaller strings, called tokens
- You use the `str_split()` or `explode()` function to split a string into an indexed array, in which each character in the string becomes a separate element in the array
- The `implode()` function combines an array's elements into a single string, separated by specified characters

Summary (continued)

82

- The `strcasecmp()` function performs a case-insensitive comparison of strings, whereas the `strcmp()` function performs a case-sensitive comparison of strings
- The `similar_text()` and `levenshtein()` functions are used to determine the similarity of two strings
- You can use the `soundex()` and `metaphone()` functions to determine whether two strings are pronounced similarly

Summary (continued)

83

- ❑ **Regular expressions** are a pattern of specially formatted strings that can be used to validate the structure of a string
- ❑ Regular expressions are made up of both literal characters and special characters, called **metacharacters**, which define the pattern-matching rules
- ❑ In a regular expression, a backslash character (\\) is used to match metacharacters as literal values

Summary (continued)

84

- ❑ **Quantifiers** are metacharacters that specify the number of times a particular match may occur
- ❑ **Subexpressions** are characters contained in parentheses within a regular expression
- ❑ The format and quantity of the characters in the subexpression can be defined as a group
- ❑ A **character class** is multiple characters enclosed in square brackets (`[]`) that are treated as a single unit

Summary (continued)

85

- The `|` metacharacter allows a string to be comprised of an alternate set of substrings. The `|` metacharacter performs essentially the same function as the Or `(||)` operator in conditional expressions