Institiúid Teicneolaíochta Chorcaí
Cork Institute of Technology

SOFT7008
Server Side Web Development

Manipulating Strings in PHP

Méabh O'Connor
meabh.oconnor@cit.ie

Strings in PHP

- PHP is most commonly used for:
  - producing valid HTML
  - processing form data submitted by users
- *String processing is vital* in coding PHP and it can be tricky

Constructing Text Strings

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

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

- A string must ***begin and end with the same type*** of **quotation mark** – single **or** double

Constructing Text Strings

```
echo "<p>This is a text string</p>"; //valid
echo '<p>This is a text string</p>'; //valid
echo "<p>This is a text string</p>'; //invalid
```

- The 3rd string would display incorrectly because the PHP scripting engine cannot tell where the literal string begins and ends .
- When it parses this string it searches for a double quote to mark the end of the string but does not find it.

Constructing Text Strings

- To include a **quoted string *within*** a **literal string surrounded by double quotation marks**

  => you surround the quoted string with single quotation marks

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

- To include a **quoted string *within*** a **literal string surrounded by single quotation marks**

  => you surround the quoted string with double quotation marks

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


Constructing Text Strings

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

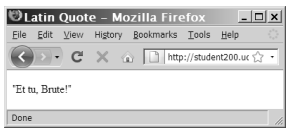


Figure 3-2 Output of a text string containing double quotation marks

Displaying Variables in Strings

- The output of variable names *inside a text string* **depends** on *whether the string is surrounded by double or single quotation marks*
- If double quotation marks are used – the *value assigned to a variable will appear*. This is called **variable interpolation**.

```
$votingAge = 18;
echo "<p> The legal voting age is $votingAge. </p>";
```

will output the following HTML to the browser:

```
<p> The legal voting age is 18. </p>
```


Displaying Variables in Strings

BUT

```
$votingAge = 18;
echo '<p> The legal voting age is $votingAge. </p>';
```

Figure 1-12 Output of an `echo` statement that includes text and a variable surrounded by *single* quotation marks

Working with String Operators

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

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

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

HTML string output to browser:

```
<p>Paris is in France.</p>
```

browser displays:

Paris is in France.

Working with String Operators

- Variable assignment, string concatenation and variable interpolation in action:

```
$var1 = 'PHP';          // Assigns a value of 'PHP' to $var1
$var2 = 5;              // Assigns a value of 5 to $var2
$var3 = $var2 + 1;      // Assigns a value of 6 to $var3
$var2 = $var1;          // Assigns a value of 'PHP' to $var2
echo $var1;             // Outputs 'PHP'
echo $var2;             // Outputs 'PHP'
echo $var3;             // Outputs '6'
echo $var1 . ' rules!'; // Outputs 'PHP rules!'
echo "$var1 rules!";    // Outputs 'PHP rules!'
echo '$var1 rules!';    // Outputs '$var1 rules!'
```


Adding Escape Characters and Sequences

- Take extra care when *using single quotation marks* with **possessives** and **contractions** in *strings surrounded by single quotation marks*.

```
echo '<p>This code's not going to work</p>';
```

- PHP assumes that the literal string ***ends*** with the apostrophe following "code".

Adding Escape Characters and Sequences

- 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>';
```

- Placing a backslash in front of the apostrophe tells the PHP scripting engine to *treat the apostrophe as a regular keyboard character* – such as "a", "b", "1" or "2" NOT as part of a **single quotation mark pair** than encloses a text string.

Adding Escape Characters and Sequences

- Do not add 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

- 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
\xh	Inserts a character whose hexadecimal value is *h*, where h is one or two hexadecimal digits (0-9, A-F), case insensitive
\o	Inserts a character whose octal value is *o*, where o is one, two, or three octal digits (0-7)

Table 3-1 PHP escape sequences within double quotation marks

Adding Escape Characters and Sequences

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

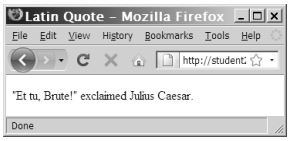


Figure 3-4 Output of literal text containing double quotation escape sequences

Adding Escape Characters and Sequences

- It is good programming practice to include an `\n` escape sequence *at the end* of an `echo` statement output string so that the HTML output by the PHP script is *easier to read and debug*.

```
echo "<p>This code's going to work.</p> \n";
```

- This has no effect on the browser display.

Adding Escape Characters and Sequences

HTML output without `\n` in generating PHP script – difficult to read

Edit the PHP code:

```
echo $Days[0], "<br /> \n";
echo $Days[1], "<br /> \n";
echo $Days[2], "<br /> \n";
echo $Days[3], "<br /> \n";
echo $Days[4], "<br /> \n";
echo $Days[5], "<br /> \n";
echo $Days[6], "<br /> \n";
```

Adding Escape Characters and Sequences

Now the HTML generated by the PHP script is easier to read:

Simple and Complex String Syntax

- **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>";
```

- PHP **interpolates** the value of the `$Vegetable` variable to display this text in the browser:

Do you have any broccoli?

Simple and Complex String Syntax

- When PHP encounters a **dollar sign** $ within a text string – it attempts to *evaluate any characters that follow the $ as* ***part of the variable*** *name* until it comes to a character that is not allowed in an identifier e.g. a space

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

- So in this sample code the `$Vegetable` variable is interpreted correctly because the **question mark** is *not a legal character for an identifier*

Simple and Complex String Syntax

- However – consider the following version of the preceding sample code:

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

- Because an '**s**' is appended to the `$Vegetable` variable name – this echo statement displays **incorrectly** as the *PHP scripting engine is attempting to locate a variable named* `$Vegetables?` (plural) which has not been declared
- **So an error will be returned for this code:**

```
Notice: Undefined variable: Vegetables……
```


Simple and Complex String Syntax

- To make the preceding example work – you need to ***surround the variable name with curly braces*** **{}** as shown below:

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

- This is called **complex string syntax**
- Complex string syntax is *only recognized if the* ***opening brace is immediately before or after*** a variable's dollar sign $!

```
${Vegetable}s // valid
```


Simple and Complex String Syntax

- However – if you place *any characters between the opening brace and the dollar sign* – the contents of the string are *interpreted as literal values*:

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

- Displays the text string in the browser:

Do you have any { carrot}s?

Working with a Single String

- 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

- 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>";
```

- displays:

The book title contains 23 characters.

Counting Characters and Words in a String

- 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>";
```

- displays:

The book title contains 4 words.

Modifying the Case of a String

- PHP provides several functions to manipulate the case of a string
  - The `strtoupper()` function converts all letters in a string to uppercase
    - www.w3schools.com/php/strtoupper
  - The `strtolower()` function converts all letters in a string to lowercase
    - www.w3schools.com/php/strtolower

Modifying the Case of a String

- When working with natural languages – more complex conversions are needed.
- Sentences in English start with an uppercase letter.
  - Use the `ucfirst()` function to ensure the first character of a string is uppercase.
- Titles of books, songs poems and articles usually have the first letter of each word capitalised.
  - Use the `ucwords()` function to convert the first character of each word in a string to uppercase
- The reverse functions are `lcfirst()` and `lcwords()`

Modifying the Case of a String

- Since the `ucfirst()`, `ucwords()`, `lcfirst()` and `lcwords()` functions only change the first character in the string/words
  - 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

- PHP has several built-in functions to use with the HTML in web pages.
- Some characters in HTML have a special meaning and must be encoded using HTML entities in order to preserve that meaning (see http://www.w3schools.com/html/html_entities.asp)
  - The `htmlspecialchars()` function converts special characters to HTML entities
    - The predefined characters are:
    - & (ampersand) *becomes* &
    - " (double quote) *becomes* "
    - ' (single quote) *becomes* '
    - < (less than) *becomes* <
    - > (greater than) *becomes* >

Encoding and Decoding a String

Example:

```
<?php
$str = "This is some <b>bold</b> text.";
echo htmlspecialchars($str);
?>
```

- The HTML output of the code above will be using **View Source** in browser:

```
<!DOCTYPE html>
<html>
<body>
This is some &lt;b&gt;bold&lt;/b&gt; text.
</body>
</html>
```


Other Ways to Manipulate a String

- 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

- 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 character 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

- A positive value in the in the `length` parameter determines how many characters to return
- A negative value in the `length` parameter skips 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

```
$ExampleString = "woodworking project";
echo substr($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";
```

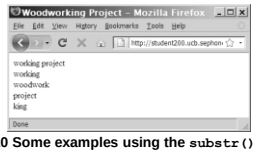


Figure 3-10 Some examples using the `substr()` function

Working with Multiple Strings

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

Working with Multiple Strings

- If you receive a text string that contains multiple data elements separated by a common **delimiter** – you will probably want to the split the string into its individual elements
- A delimiter is a character or string that is used to separate components in a list. The delimiter is usually not found in any of the elements.
-
  - e.g a list of names separated by commas (CSV)

Finding and Extracting Characters and Substrings

- There are two types of string *search* and *extraction* functions:
  - Functions that return a numeric **position** in a text string
  - 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

- 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)

- Example: `strpos()`

Find the position of the first occurrence of "php" inside the string:

```
<?php
echo strpos("I love php, I love php too!","php");
?>
```

0	1	2	3	4	5	6	7	8	9	10	11	12	..
I		l	o	v	e		p	h	p	,		l	

- This returns a value of : 7

Finding and Extracting Characters and Substrings (continued)

- Related functions:
- strrpos() – Finds the position of the last occurrence of a string inside another string (case-sensitive)
- stripos() – Finds the position of the first occurrence of a string inside another string (case-insensitive)
- strripos() – Finds the position of the last occurrence of a string inside another string (case-insensitive)

Finding and Extracting Characters and Substrings

- 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

- The `str_replace()` and `str_ireplace()` functions both accept three arguments:

```
strreplace(find,replace,string,count)
```

Parameter	Description
find	Required. Specifies the value to find
replace	Required. Specifies the value to replace the value in *find*
string	Required. Specifies the string to be searched
count	Optional. A variable that counts the number of replacements

Replacing Characters and Substrings

Example:

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


Dividing Strings into Smaller Pieces

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

```
$variable = strtok(searchstring, separators);
```

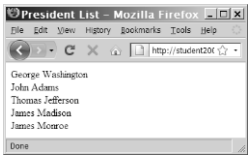
- The PHP scripting engine *keeps track of the current token and where it is within the search string* each time the `strtok()` is called.
- Often used within a loop to break a string up into component parts based the specified separator parameter.

Dividing Strings into Smaller Pieces

- 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

```
$Presidents = "George Washington;John Thomas
  Jefferson;James Madison;James Monroe";
// separator is a semi-colon
$President = strtok($Presidents, ";");
while ($President != NULL)
{
     echo "$President<br />";
     // move on to next token in string
     $President = strtok(";");
}
```



Dividing Strings into Smaller Pieces

- The `strtok()` function divides a string into tokens using ***ANY*** of the characters that are passed into it in the 2nd argument
- So – in the previous example – if you include a semicolon **and** a space ("; ") in the 2nd `searchstring` argument – the string is split into tokens at
  - each semi-colon **OR**
  - each space in the string

Dividing Strings into Smaller Pieces

```
$Presidents = "George Washington;John Adams;Thomas
  Jefferson;James Madison;James Monroe";
// separator is a semi-colon and a space
$President = strtok($Presidents, "; ");
while ($President != NULL) {
     echo "$President<br />";
     $President = strtok("; ");
}
```

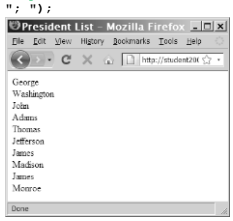


Figure 3-16 Output of a script with a `strtok()` function that uses the semi-colon and space separators

Converting between Strings and Arrays

- 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

Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
    print_r(str_split("Hello"));
?>

</body>
</html>
```

Output Result:

Array ([0] => H [1] => e [2] => l [3] => l [4] => o)

Converting between Strings and Arrays

- 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
- 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

Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
   $str = "Hello world. It's a beautiful day.";
   print_r (explode(" ",$str));
?>

</body>
</html>
```

- **Output Result:**

  Array ([0] => Hello [1] => world. [2] => It's [3] => a [4] => beautiful [5] => day.)

Converting between Strings and Arrays

- The `explode()` function
  - If you pass to the `explode()` function an empty string as the *separator* argument, the function returns a Boolean value of `FALSE`

Converting between Strings and Arrays (continued)

- The `implode()` function combines an array's elements into a single string, separated by character(s) specified in the *`separators`* string parameters
- The syntax is:

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


Converting between Strings and Arrays

```
$PresidentsArray = array("George Washington", "John Adams",
"Thomas Jefferson", "James Madison", "James Monroe");
// create string from array elements separated by comma
$Presidents = implode(", ", $PresidentsArray);
echo $Presidents;
```

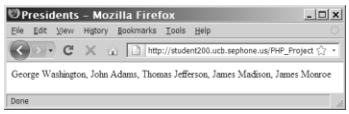


Figure 3-18 Output of a string created with the `implode()` function

Comparing Strings

- 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)

- 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

- 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
- These string comparison functions compare strings based on their ASCII values

String Comparison Functions

Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
echo strcmp("Hello","Hello");
echo "<br>";
echo strcmp("Hello","hELLo");
?>

</body>
</html>
```

Output Result:

```
0
-1
```