SWINBURNE UNIVERSITY OF TECHNOLOGY

Advanced Web Development: Strings

Week 4

Outline

- Handle form submissions and processing
- Manipulate Strings
- Parse Strings
- Compare Strings
- Use Online PHP Manual to look for PHP string functions

- Reading: Textbook Chapter 3

HANDLE FORM SUBMISSIONS AND PROCESSING

Handling Form Submissions

`get` versus `post`

- text strings in `$_GET` and `$_POST` autoglobals

- A **query string** is a set of `name=value` pairs appended to a target URL
- Form data is submitted in `name=value` pairs, based on the `name` and `value` attributes of each element
- A question mark (`?`) and a query string are automatically appended to the URL of a server-side script for any forms that are submitted with the `get` method

Handling Form Submissions (continued)

- Each `name=value` pair within the query string is separated with ampersands (`&`)

```
<form action="processOrder.php" method="get" >
 ... <input type="text" name="book_title"
   value="technical" />
 ... <input type="text" name="number_of_copies"
   value="1" />
 ...
</form>
```

Query String:

`processOrder.php?books_title=technical&number_of_copies=1`

Note: You may need to validate submitted data!

Determining if Form Variables Contain Values

Use the `isset()` or `empty()` functions to ensure that a variable contains a value

- The `isset()` function determines whether a variable has been declared and initialised (or "set")
- The `empty()` function determines whether a variable is empty
- Parameter of both functions is the name of the variable you want to check

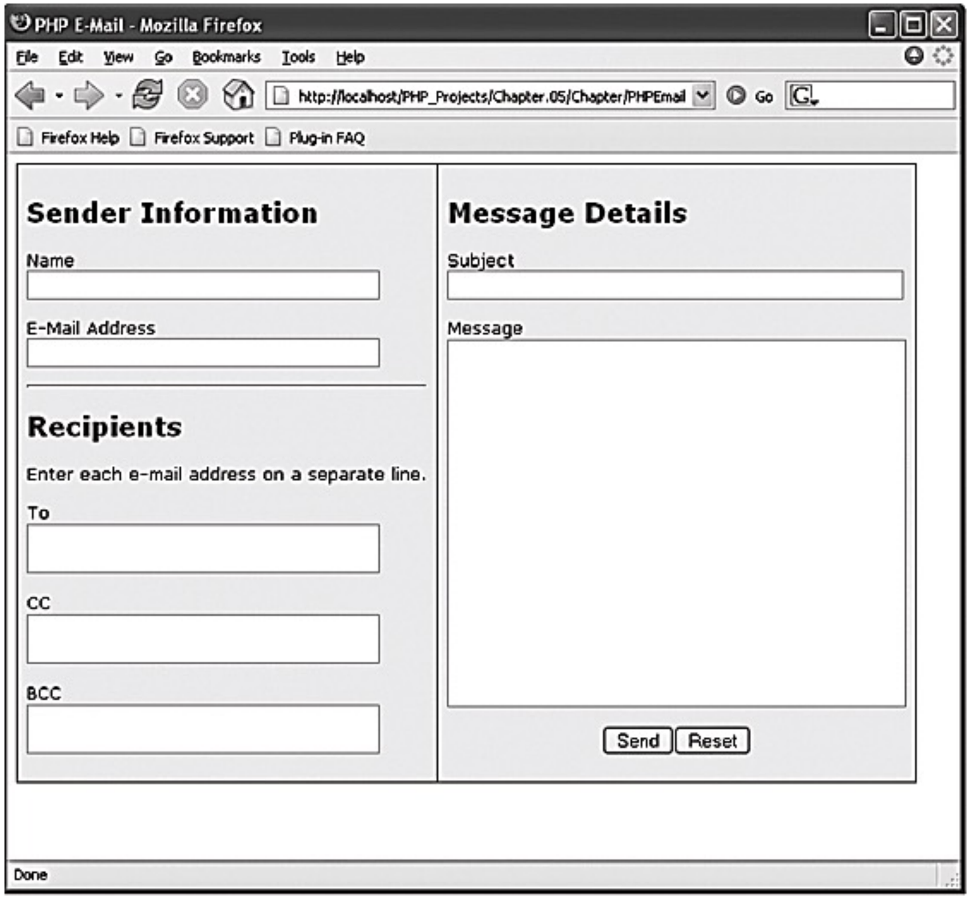
Testing if Form Variables Contain Numeric Values

Use the `is_numeric()` function to test whether a variable contains a numeric string

```
if (isset($_GET["height"]) && isset($_GET["weight"])) {
   if (is_numeric($_GET["weight"]) &&
        is_numeric($_GET["height"])) {
        $bodyMass = $_GET["weight"] / ($_GET["height"]
                * $_GET["height"]) * 703;
        printf("<p>Your body mass index is %d.</p>",
                $bodyMass);
   }
   else
        echo "<p>You must enter numeric values!</p>";
}
```

Note: Use `$_POST` when `method=post` is used instead of `get` for `form` submission.

Using `mail()` Function



PHP E-Mail - Mozilla Firefox

http://localhost/PHP_Projects/Chapter.05/Chapter/PHPEmail

Sender Information

Name

E-Mail Address

Recipients

Enter each e-mail address on a separate line.

To

CC

BCC

Message Details

Subject

Message

phpemail.html in a Web browser

Using `mail()` Function (continued)

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

```
mail(recipient(s), subject,
 message[, additional_headers])
```

- The `mail()` function returns a value of true if a message was delivered successfully or false if it was not

```
$to = “amolnar@swin.edu.au";
$subject = "This is the subject";
$message = "This is the message.";
$headers = "From: Andreea Molnar
  <amolnar@swin.edu.au>";
mail($to, $subject, $message, $headers);
```

MANIPULATE STRINGS

Constructing Text Strings

- A text string contains 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 "<p>Dr. Livingstone, I presume?</p>";
$explorer = "Henry M. Stanley";
echo $explorer;
```

- Text strings can also be surrounded with single quotation marks

Note: no data type for a single character in PHP

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

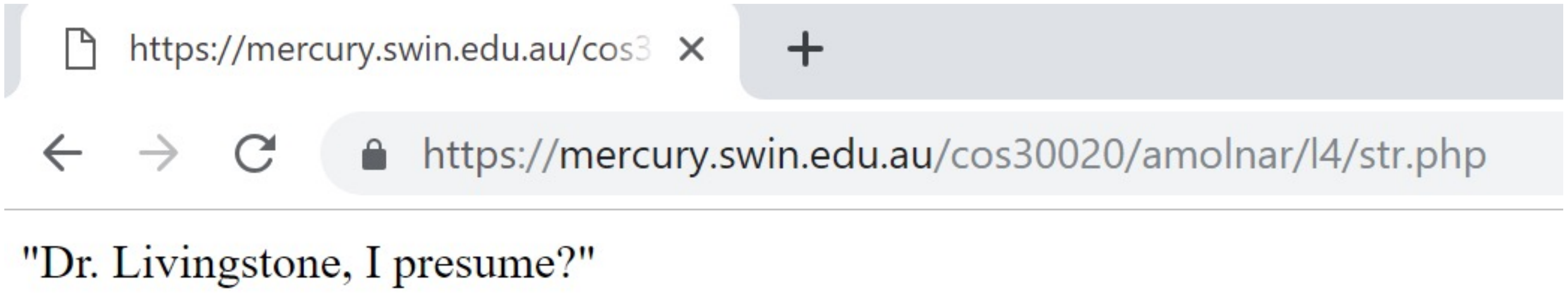
```
$explorerQuote = '<p>"Dr. Livingstone, I presume?"</p>';
```

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

```
$explorerQuote = "<p>'Dr. Livingstone, I presume?'</p>";
```

Constructing Text Strings (continued)

```
$explorerQuote = '<p>"Dr. Livingstone, I presume?"</p>';
echo $explorerQuote;
```


https://mercury.swin.edu.au/cos3
https://mercury.swin.edu.au/cos30020/amolnar/l4/str.php

```
"Dr. Livingstone, I presume?"
```

Output of a text string containing double quotation marks

Working with String Operators

In PHP, you use two operators to combine strings

- **Concatenation operator .**

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

- **Concatenation assignment operator .=**

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

SWINBURNE UNIVERSITY OF TECHNOLOGY

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>Marilyn Monroe\'s real name was Norma Jean
   Baker.</p>';
```

- Do not add a backslash before an apostrophe
  if you surround the text string with double quotation marks

```
echo "<p>Marilyn Monroe's real name was Norma Jean
   Baker.</p>";
```

Adding Escape Characters and Sequences (continued)

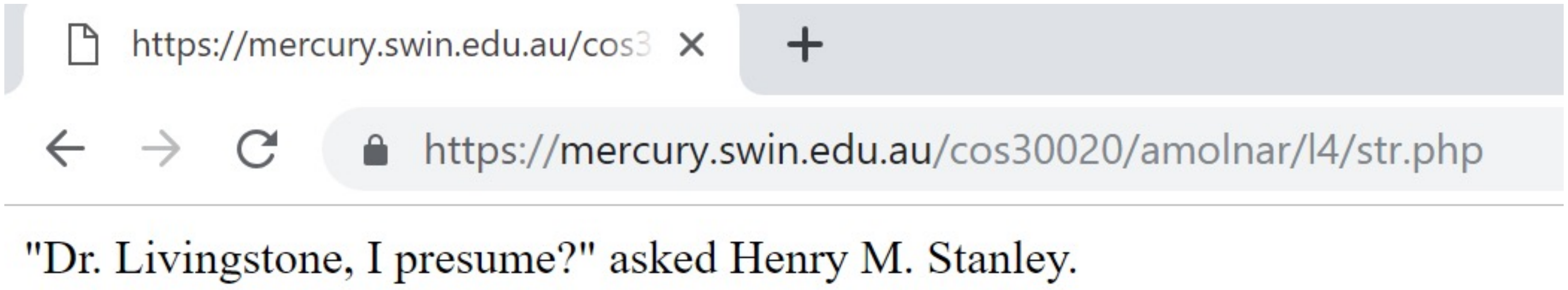
- The escape character combined with one or more other characters is called an escape sequence

Table of PHP escape sequences within double quotation marks

Escape Sequence	Description
\\	Inserts a backslash
\$	Inserts a dollar sign
\r	Inserts a carriage return
\"	Inserts a double quotation mark
\t	Inserts a horizontal tab
\n	Inserts a new line
\regular expression	Inserts a character in hexadecimal notation that matches the regular expression

Adding Escape Characters and Sequences (continued)

```
$explorer = "Henry M. Stanley";
echo "<p>\"Dr. Livingstone, I presume?\" asked
   $explorer.</p>";
```


https://mercury.swin.edu.au/cos3
https://mercury.swin.edu.au/cos30020/amolnar/l4/str.php

"Dr. Livingstone, I presume?" asked Henry M. Stanley.

Output of literal text containing double quotation escape sequences

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>";
```

How about: `echo "<p>Do you have any $vegetables?</p>";`

```
  //causes an error, variable not declared.
```

- 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>";
```

How about: `echo "<p>Do you have any {$vegetable}s?</p>";`

```
  //output is: Do you have any carrots?
```

COMPARE STRINGS

Comparing Strings

Using Comparison Operator in Module 2

```
$loc01 = "Miami is in Florida.";
$loc02 = "Havana is in Cuba.";
if ($loc01 == $loc02)
     echo "<p>Same location.</p>";
else
     echo "<p>Different location.</p>";
```

Comparing Strings (continued)

```
$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>";
```

ASCII American Standard Code for Information Interchange

- Numeric representations of English characters
- 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”)
- Since lowercase letters have higher values than uppercase letters, they are evaluated as being “greater” than the uppercase letters

Note: UTF-8 is a strict superset of ASCII with the same physical encoding for ASCII characters

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 parameters representing the strings you want to compare
- Most string comparison functions compare strings based on their ASCII values – returns <0 (if smaller); >0 (if larger); =0 (if same)

Using Similarity Functions to Compare

- 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

Using Similarity Functions to Compare (continued)

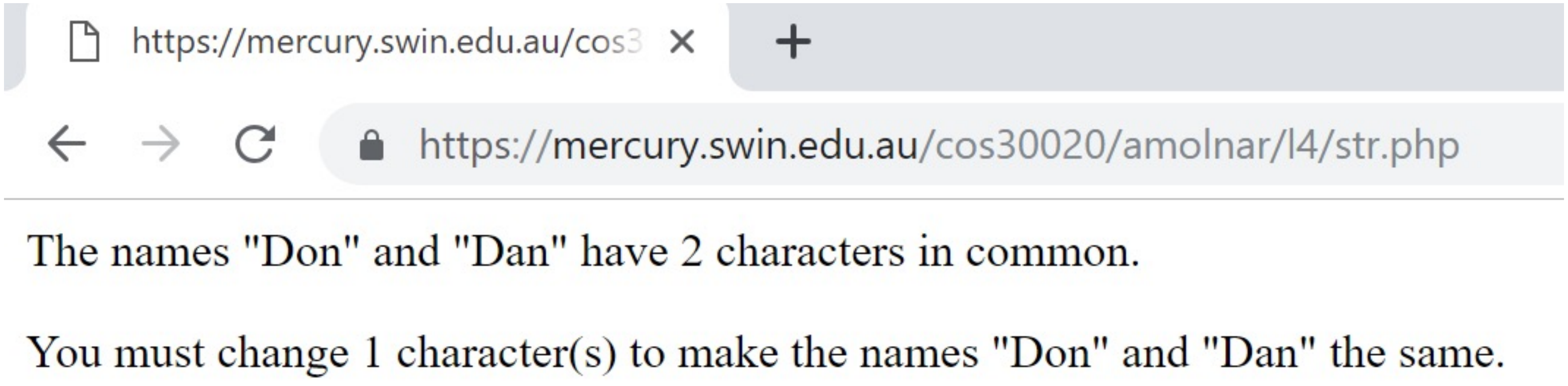
- Both functions accept two string arguments representing the values you want to compare

```
$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>";
```

Using Similarity Functions to Compare (continued)


https://mercury.swin.edu.au/cos3
https://mercury.swin.edu.au/cos30020/amolnar/l4/str.php

```
The names "Don" and "Dan" have 2 characters in common.

You must change 1 character(s) to make the names "Don" and "Dan" the same.
```

Output of a script with the `similar_text()` and `levenshtein()` functions

Using Pronunciation Functions to Compare

- 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
  e.g. `soundex("internet")` returns `"I536"`
- The `metaphone()` function returns a code representing an English word's approximate sound
  e.g. `metaphone("internet")` returns
  `"INTRNT"`

Using Pronunciation Functions to Compare

(continued)

```
$firstName = "internet";
$secondName = "intranet";
$firstNameSoundsLike = metaphone($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>";
```

Note: Different words may sound the same using the metaphone function

PARSE STRINGS

Parsing Strings

- **Parsing** is the act of extracting characters or substrings from a larger string
- When programming, parsing refers to the extraction of information from string literals and variables

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

```
$myStr = ' ab cd ';
echo strlen($myStr); // 7
```

- The `str_word_count()` function returns the number of words inside a string
- Parameter of the `str_word_count()` function can be a literal string or the name of a string variable

```
$bookTitle = "The Cask of Amontillado";

echo "<p>The book title contains " .
  str_word_count($bookTitle) . " words.</p>";
```

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

`strpos()` Function

- Performs a case-sensitive search and returns the position of the first occurrence of one string in another string
  Note: begins with a value of 0 // at the first character
- Two parameters for the `strpos()` function:
  - ☐ The first is the string you want to search
  - ☐ The second 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`

`strpos()` Function (Continued)

```
$email = "president@whitehouse.gov";
  echo strpos($email, "@"); //returns 9
  echo strpos($email, "p"); //returns 0
```

`strchr()` and `strrchr()` Functions

- Parameters of both functions are 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, *i.e. last portion*
- `strchr()` function starts searching at the beginning of a string
- `strrchr()` function starts searching at the end of a string Note: Extra ‘r’ means reverse

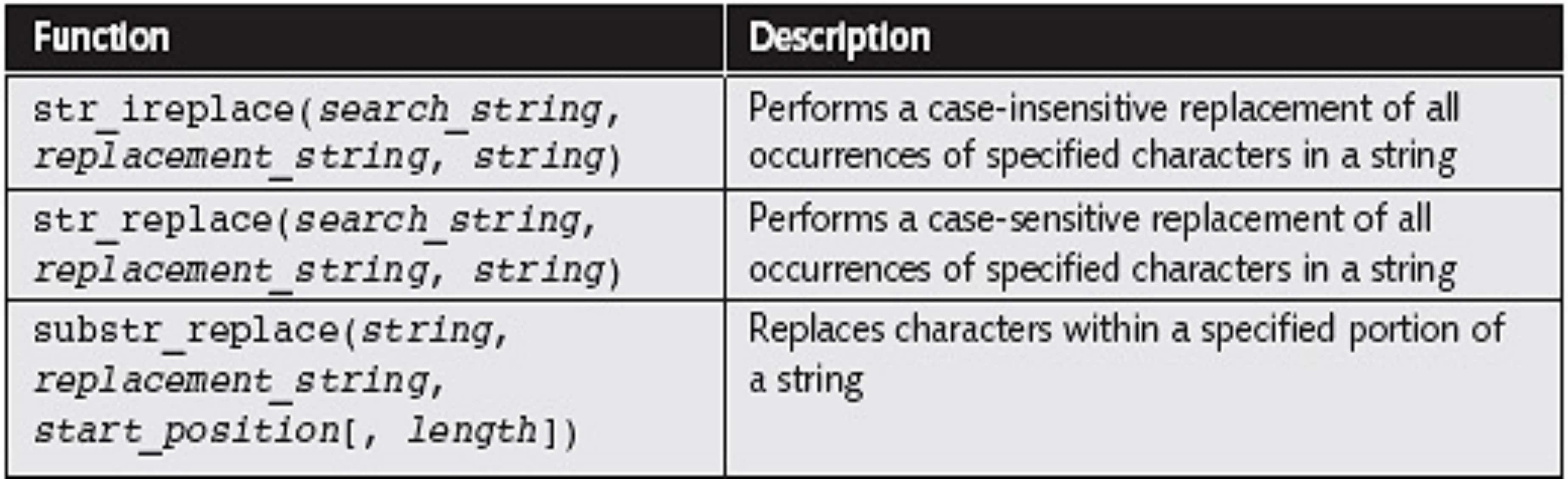
`substr()` Function

- To extract characters from the beginning or middle of a string, combine the `substr()` function with other functions
- Parameters of the `substr()` function: a text string, the starting position and length of the substring you want to extract

```
$email = "president@whitehouse.gov";
$nameEnd = strpos($email, "@");
echo "<p>The name portion of the e-mail address
  is '" . substr($email, 0, $nameEnd) . "'.</p>";
```

Replacing Characters and Substrings

PHP string replacement functions



Function	Description
str_ireplace(*search_string*, *replacement_string*, *string*)	Performs a case-insensitive replacement of all occurrences of specified characters in a string
str_replace(*search_string*, *replacement_string*, *string*)	Performs a case-sensitive replacement of all occurrences of specified characters in a string
substr_replace(*string*, *replacement_string*, *start_position*[, *length*])	Replaces characters within a specified portion of a string

Note: Extra 'i' means case-insensitive

`str_replace()` and `str_ireplace()` Functions

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

Dividing Strings into Smaller Pieces

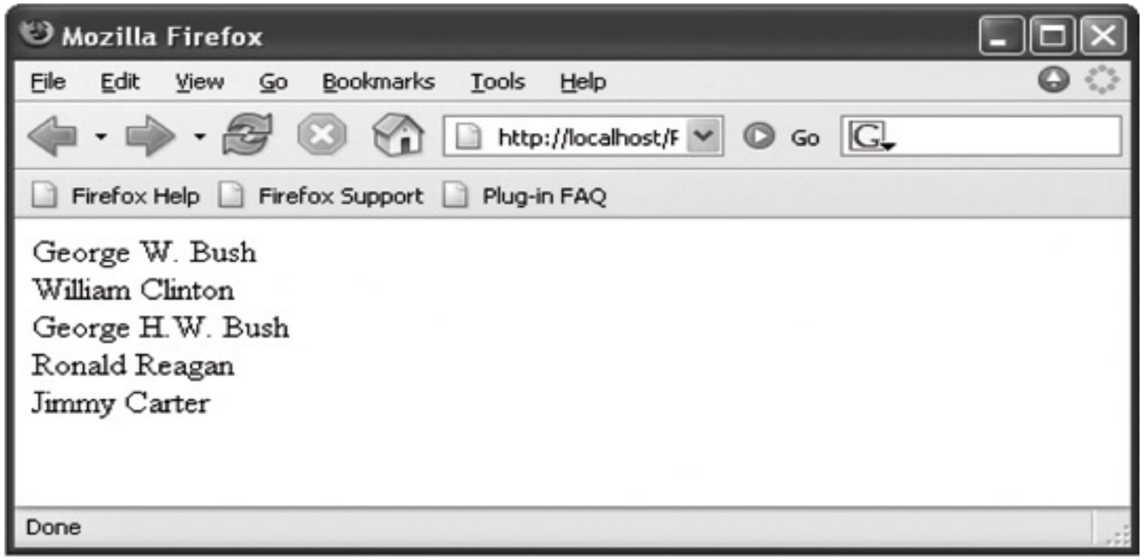
- Use the `strtok()` function to break a string into smaller strings, called **tokens** (one by one)
- The syntax for the **`strtok()`** function is:

```
$variable = strtok(string, 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
- The `strtok()` function returns tokens one by one

`strtok()` Function

```
$presidents = "George W. Bush;William Clinton;
   George H.W. Bush;Ronald Reagan;Jimmy Carter";
$president = strtok($presidents, ";");
while ($president != NULL) {
      echo "$president<br/>";
      $president = strtok(";"); //only the separator ";"
   here. The PHP scripting engine keeps track of the
   current token and next token.
}
```



```
George W. Bush
William Clinton
George H.W. Bush
Ronald Reagan
Jimmy Carter
```

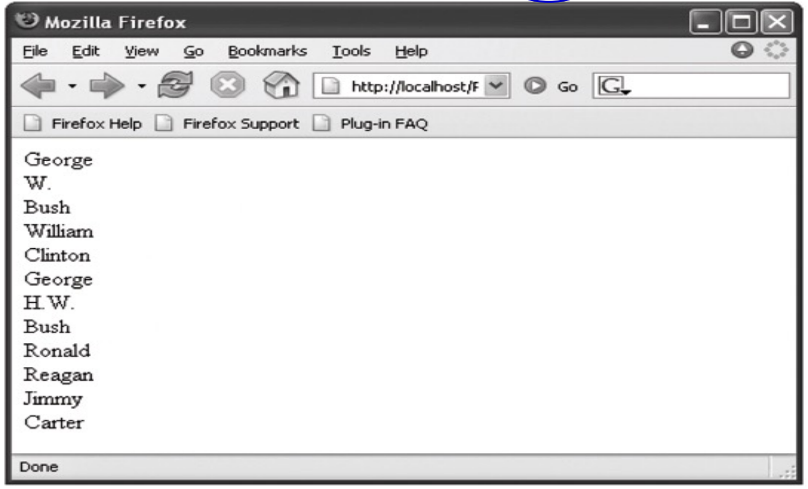
Output of a script that uses `strtok()`

`strtok()` Function (continued)

- `strtok()` divides a string into tokens using any of the characters that are passed

```
 $presidents = "George W. Bush;William Clinton;
George H.W. Bush;Ronald Reagan;Jimmy Carter";
 $president = strtok($presidents, "; ");
 while ($president != NULL) {
    echo "$president<br />";
    $president = strtok("; ");
 }
```

Two separators used: ";" and " "



```
George
W.
Bush
William
Clinton
George
H.W.
Bush
Ronald
Reagan
Jimmy
Carter
```

Output of a script with a `strtok()` function that uses two separators

Converting Between Strings and Arrays

Can also split a string into an array

- 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)

- 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);
```

- Note: 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 (continued)

```
$presidents = "George W. Bush;William Clinton;
   George H.W. Bush;Ronald Reagan;Jimmy Carter";
$presidentArray = explode(";", $presidents);
   //how about "; " ???
foreach ($presidentArray as $president) {
      echo "$president<br />";
}
```

- 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 value of `false`

`implode()` Function

Opposite to `explode()`

- Combines an array's elements into a single string, separated by specified characters
- The syntax is:

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

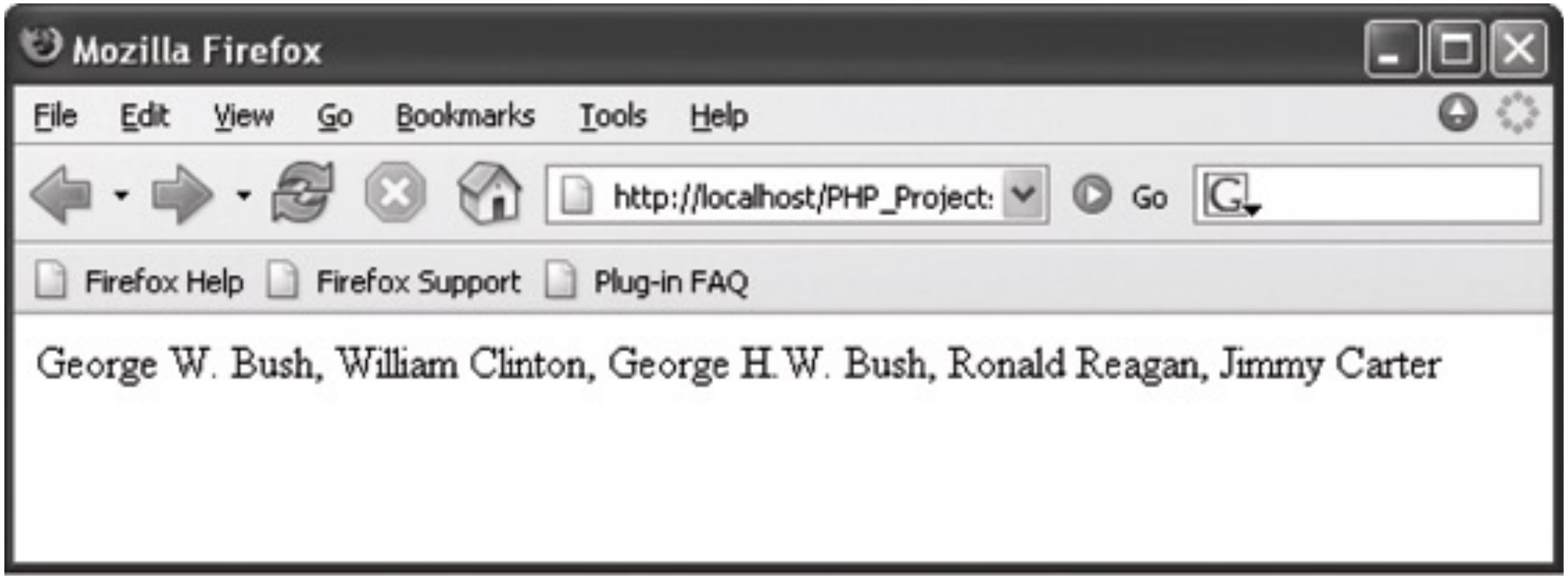
SWINBURNE UNIVERSITY OF TECHNOLOGY

`implode()` Function (continued)

```
$presidentsArray = array("George W. Bush",
   "William Clinton", "George H.W. Bush", "Ronald
   Reagan", "Jimmy Carter");
```

```
$presidents = implode(", ", $presidentsArray);
```

```
echo $presidents;
```


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Project:
Go
Firefox Help Firefox Support Plug-in FAQ

```
George W. Bush, William Clinton, George H.W. Bush, Ronald Reagan, Jimmy Carter
```

Output of a string created with the `implode()` function

USE REGULAR EXPRESSION

Using Regular Expressions

Web Developers should understand the concepts and value of using Regular Expressions

- Regular Expressions are a useful way to concisely define the syntax and 'pattern' of textural data.
- Simple functions can be used to test or 'match' data against the 'pattern'.
- Regular Expressions can be used in both client-side and server-side scripts, so the same 'pattern' can be consistently applied to verify data formats.

And in particular be able to:

- Use Regular Expressions to check values entered in HTML forms.

http://www.php.net/manual/en/book.pcre.php

What are Regular Expressions?

- are strings that describe the ‘pattern’ or ‘rules’ for strings
- are strings that follow a set of syntax rules
- can be used as a concise and consistent way to test for matching patterns
- ***are great for checking form values!***

Regular Expressions - Basic Syntax

`/pattern/modifiers`

- **Quantifiers**

*	0 or more
+	1 or more
?	0 or 1
{4}	exactly 4
{4.}	4 or more
{4,6}	4,5 or 6

- **Groups & Ranges**

.	Any character (except \n)
(a|b)	a or b
(…)	group
(?:…)	passive group
[abc]	set ("range") a, b or c
[^abc]	not a, b or c
[a-g]	set range a to g
[3-6]	set range of digits 3,4,5 and 6
\n	"nth" group or subpattern

Regular Expressions - Basic Syntax

`/pattern/modifiers`

http://php.net/manual/en/reference.pcre.pattern.modifiers.php

- **Pattern Basics**

^	Start of string
$	End of string
.	Match any single character
(a|b)	a or b
(…)	Group section
[abc]	match any character in the set
[^abc]	not match in the set
[a-z]	match the range
\d	match a single digit from 0 to 9 *shortcut for [0-9]*

- **Pattern Quantifiers**

a?	0 or 1 of a
a*	0 or more of a
a+	one or more instance of a
a{3}	exactly 3 a's = aaa
a{3,}	3 or more a's
a{3,6}	between 3 to 6 a's
!(pattern)	"not" pattern

[\ ^ $. | ? * + ( ) are the 11 meta-characters, or special characters, used in the syntax.
If you want to include these, you need to escape them with \

eg. \(

Regular Expressions - Basic Syntax

```
/pattern/modifiers
```

- **Pattern Modifiers**

/g	global matching
/i	case insensitive
/s	single line mode
/m	multiple-line mode
/x	allow comments and white space in pattern
/e	evaluate replacement
/U	ungreedy replacement

There are many useful online syntax references about Regular Expressions, such as:
http://www.regular-expressions.info/

Regular Expressions - Basic Examples

/WebProg/	matches "Isn't WebProg great?"
/^WebProg/	matches "WebProg rules!", not "What is WebProg?"
/WebProg$/	matches "I love WebProg", not "WebProg is great!"
/^WebProg$/	matches "WebProg", and nothing else
/bana?na/	matches "banana" and "banna", but not "banaana".
/bana+na/	matches "banana" and "banaana", but not "banna".
/bana*na/	matches "banna", "banana", and "banaaana", but not "bnana"
/^[a-zA-z]+$/	matches any string of one or more letters and nothing else.

Regular Expressions in PHP

- PHP uses Perl Compatible Regular Expressions (PCRE) and has a range of pre-defined PCRE functions

  http://www.php.net/manual/en/ref.pcre.php

- Common functions

  `preg_match(),rpreg_replace(),preg_split()`

- Initialise a Regular Expression pattern, and test string

```
$pattern = "/(chapter \d+(\.\d)*)/i";
$str = "For more information, see Chapter 3.4.5.1";
if (preg_match($pattern, $str) {
    echo "A match was found.";
} else {
    echo "A match was not found.";
}
```

Regular Expressions in PHP

- A simple regular expression can be the equivalent of many lines of code.
- Simply define the 'pattern' of the Regular Expression, e.g. 'pattern' for a phone number such as

  (03) 9214-8000

  you could use

  $pattern = "/^\(\d\d\) \d\d\d\d-\d\d\d\d$/" ;
- Then 'match' the input string against the 'pattern'

  preg_match($pattern, $inputString) // true if OK

Regular Expressions in PHP - Example

- Simple check for a phone number using **`preg_match()`**

```
function checkPhoneNumber($phoneNo) {
 $phoneRE = "/^\(\d\d\) \d\d\d\d-\d\d\d\d$/";
 if (preg_match($phoneRE, $phoneNo)) {
   return true;
 } else {
   return false;
 }
}
```

```
<form action="..." >
<p><label ...>Enter phone number (e.g.(03) 3456-7890):
  </label>
  <input type="text" name="phone" /></p>
<p><input type="submit" value="Send" /></p>
</form>
```

http://php.net/manual/en/reference.pcre.pattern.syntax.php

USE ONLINE PHP MANUAL TO LOOK FOR PHP STRING FUNCTIONS

Other String Functions

- There are many useful string functions – see http://php.net/manual/en/ref.strings.php for a full list.
- Just a few:
  - `trim()` Strip whitespace (or other characters) from the beginning and end of a string
  - `htmlspecialchars()` Some characters have a special meaning in HTML and have to be escaped if they appear in your text. It can also help to prevent XSS (cross-site-scripting) attacks.
  - `strtoupper(), strtolower()`
    Return a string in upper case or lower case.

Summary

- 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
- The most commonly used string counting function is the `strlen()` function, which returns the total number of characters in a string

Summary (continued)

- Use the `str_replace()`, `str_ireplace()`, and `substr_replace()` functions to replace text in strings
- The `strtok()` function breaks a string into smaller strings, called tokens
- The `str_split()` and `explode()` functions split a string into an indexed array
- The `implode()` function combines an array's elements into a single string, separated by specified characters

Summary (continued)

- 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
- The `soundex()` and `metaphone()` functions determine whether two strings are pronounced similarly