

ISIT307 - WEB SERVER PROGRAMMING

LECTURE 2.1 – MANIPULATING STRINGS



LECTURE PLAN

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

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

WORKING WITH STRING OPERATORS

In PHP, there are 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)

- 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

- 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 (\)

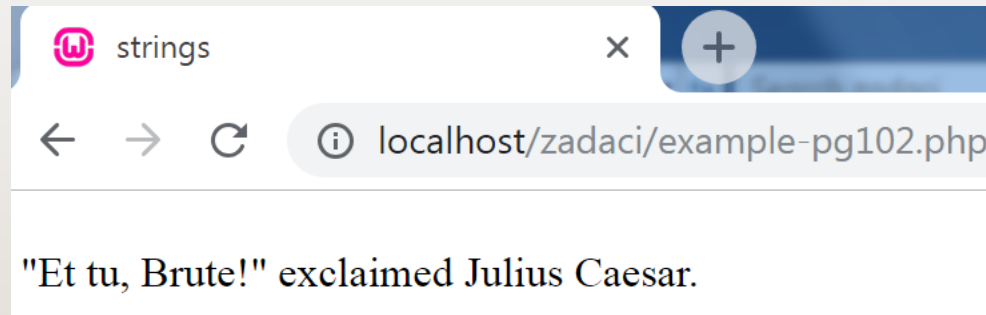
```
echo '<p>This code\'s going to work</p>';
```

- 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 - EXAMPLE

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



ADDING ESCAPE CHARACTERS AND SEQUENCES (CONTINUED)

- The escape character combined with one or more other characters is an **escape sequence**
- PHP escape sequences within double quotation marks are as follows

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)

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>";
```

- 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>";
```

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>";
```

COUNTING CHARACTERS AND WORDS IN A STRING (CONTINUED)

- 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

- 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 string is uppercase
 - The `lcfirst()` function ensures that the first character of a string is lowercase
 - The `ucwords()` function changes the first character of each word

ENCODING AND DECODING A STRING

- PHP has several built-in functions for processing strings to be used with Web pages
- Some characters in HTML have a special meaning and must be encoded using HTML entities in order to be displayed as text
 - The `htmlspecialchars()` function converts special characters to HTML entities
 - The `html_specialcharacters_decode()` function converts HTML character entities into their equivalent characters

ENCODING AND DECODING A STRING (CONTINUED)

- 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 `'C;'`
 - `'>'` (greater than) becomes `'E;'`

ENCODING AND DECODING A STRING (CONTINUED)

- 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

- 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)

- 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 (CONTINUED)

- 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

- EXAMPLE

```
$ExampleString = "woodworking project";  
echo substr($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";  
echo substr($ExampleString,5,-2) . "<br />\n";
```

```
echo strrev($ExampleString) . "<br />\n";  
echo str_shuffle($ExampleString) . "<br />\n";
```

WORKING WITH MULTIPLE STRINGS

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

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, EXTRACTING, REPLACING CHARACTERS AND SUBSTRINGS

- The `strchr()` and the `strrchr()` 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
- The `str_replace()` and `str_ireplace()` functions replace the substring within the string. Both accept three arguments:
 - The string you want to search for
 - A replacement string
 - The string in which you want to replace characters

FINDING, EXTRACTING, REPLACING CHARACTERS AND SUBSTRINGS

```
$Email = "my.email@uow.edu.au";  
  
echo "<p>if I use strchr - " . strchr($Email, ".") . "</p>";  
  
echo "<p>if I use strrchr - " . strrchr($Email, ".") . "</p>";  
  
echo "<p> the @ is at position - " . strpos($Email, "@") . "</p>";  
  
echo "<p>if I replace the email - " .  
    str_replace("email", "e-mail", $Email) . "</p>";
```

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(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

DIVIDING STRINGS INTO SMALLER PIECES - EXAMPLE

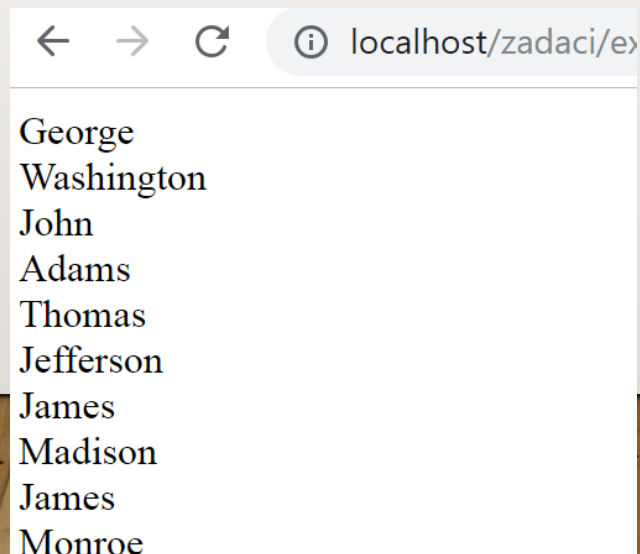
```
$Presidents = "George Washington;John Adams;Thomas  
                Jefferson;James Madison;James Monroe";  
$President = strtok($Presidents, ";");  
while ($President != NULL) {  
    echo "$President<br />";  
    $President = strtok(";");  
}
```

← → ↻ ⓘ localhost/zadaci/example-pg148.php

George Washington
John Adams
Thomas Jefferson
James Madison
James Monroe

DIVIDING STRINGS INTO SMALLER PIECES(CONTINUED)

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



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 (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);
```

- 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)

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

- 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)

- 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`

CONVERTING BETWEEN STRINGS AND ARRAYS (CONTINUED)

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

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

```
---
```

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

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

```
$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 and return 0 if the strings are the same
- The `strncasecmp()` and `strncmp()` have additional argument that represent the number of the first n characters to be compared in the strings

DETERMINING THE SIMILARITY OF TWO STRINGS

- 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 - EXAMPLE

- 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 – EXAMPLE OUTPUT

← → ↻ ⓘ localhost/zadaci/example-pg158.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.

DETERMINING IF WORDS ARE PRONOUNCED SIMILARLY

- 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 - EXAMPLE

```
$FirstName = "Keen";  
$SecondName = "Kean";  
$FirstNameSoundsLike = metaphone($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>";
```

EXAMPLE – WHAT IS THE OUTPUT?

```
<?php
$my_str = "Bob is working";
echo "<p>$my_str</p>";

$my_str[0] = "R";
echo "<p>$my_str</p>";

for($i=0; $i< strlen($my_str); $i++)
    echo "<p>$my_str[$i]</p>";
?>
```

WORKING WITH REGULAR EXPRESSIONS

- **Regular Expressions** are patterns that are used for matching and manipulating strings according to specified rules
- Most commonly these expressions are used to ensure that the user has entered the data in correct format
- PHP supports two types of regular expressions:
 - POSIX Extended
 - Perl Compatible Regular Expressions (PCRE)

WORKING WITH REGULAR EXPRESSIONS

- PCRE FUNCTIONS

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 (\)

WORKING WITH REGULAR EXPRESSIONS

- 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);
```

- The function returns 1 if the specified pattern is matched or a value of 0 if it is not matched
- The pattern is case-sensitive by default, and need to use following syntax for case-insensitive matching

```
preg_match("/pattern/i", string);
```

WRITING REGULAR EXPRESSION PATTERNS

- 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 - METACHARACTERS

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

PHP Programming with MySQL, 2011, Cengage Learning.

MATCHING ANY CHARACTER

- 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

- An **anchor** specifies that the pattern must appear at a particular position in a string
- The `^` metacharacter anchors characters to the beginning of a string
- The `$` metacharacter anchors characters to the end of a string

```
$URL = "http://www.education.com";
```

```
preg_match("/^http/", $URL); // returns 1
```

```
$URL = "http://www.education.com";
```

```
preg_match("/com$/", $URL); // returns 1
```

MATCHING SPECIAL CHARACTERS

- To match any metacharacters as literal values in a regular expression, escape the character with a backslash
- For example if we want to ensure that the string contains actual period, can be used

```
$Identifier = "http://www.education.com";
```

```
preg_match("/\com$/", $Identifier); //returns 1
```

MATCHING SPECIAL CHARACTERS

- With some metacharacters the escape sequence is more complicated, so single quotes instead can be used
- For example if we want to ensure that the string contains dollar sign, can be used

```
$Identifier = "$1234.56";
```

```
preg_match('/^\$/', $Identifier);//returns 1
```

```
preg_match("/^\\\$/", $Identifier);//returns ?
```

SPECIFYING QUANTITY

- 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>n</i> 1, <i>n</i> 2}	Specifies that the preceding character repeat at least <i>n</i> 1 times but no more than <i>n</i> 2 times

SPECIFYING QUANTITY

- 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.education.com";  
preg_match("/^https?/", $URL); // returns 1
```


SPECIFYING QUANTITY (CONTINUED)

- 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)

- A asterisk (`*`) quantifier specifies that zero or more sequential occurrences of the preceding characters match
(in the following examples, the string might begin with one or more leading zeros)

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

```
NumberString = "1234056";  
preg_match("/^0*/", $NumberString); //returns 1
```

SPECIFYING QUANTITY (CONTINUED)

- The { } quantifiers specify the number of times that a character must repeat sequentially
(in the following example, the string must contain “ZIP:” plus 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 “ZIP:” plus between five and ten characters)

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

SPECIFYING SUBEXPRESSIONS

- 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 | and the (nnn) are optional, but if included must be in the following format “| (nnn)nnn-nnnn”)

```
preg_match ("/^(1 )?(\.{3}\.)?(\.{3})(\-.{4})$/",  
                                                    "555-1234"); //return 1
```

```
preg_match ("/^(1 )?(\.{3}\.)?(\.{3})(\-.{4})$/",  
                                                    "(707) 555-1234"); //return 1
```

```
preg_match ("/^(1 )?(\.{3}\.)?(\.{3})(\-.{4})$/",  
                                                    "1 (707) 555-1234"); //return 1
```

DEFINING CHARACTER CLASSES

- **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)

- The hyphen metacharacter (–) specifies a range of values in a character class
(the following example ensures that A, B, C, D, or F are the only values assigned to the `$LetterGrade` variable)

```
$LetterGrade = "B";
```

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

DEFINING CHARACTER CLASSES (CONTINUED)

- 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";  
preg_match("[^EG-Z]", $LetterGrade);  
                                                    //returns 1  
  
$LetterGrade = "E";  
preg_match("[^EG-Z]", $LetterGrade);  
                                                    //returns 0
```


DEFINING CHARACTER CLASSES – PCRE CHARACTER TYPES

- Also, there are special characters that can be used to represent different types of data

```
preg_match ("/^[\\w-]+(\\. [\\w-]+)*@[\\w-]+  
        (\\. [\\w-]+)* (\\. [a-zA-Z]{2,})$/", $Email);
```

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

MATCHING MULTIPLE PATTERN CHOICES

- 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

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

```
preg_match("/\.(com|org|net)$/i",  
           "http://www.education.com"); // returns 1
```

PATTERN MODIFIERS

- **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