

Regular Expressions

User Guide
March 18, 2008

Contents

Introduction to regular expressions

- Examples of regular expressions
 - Fill in the Blanks (FIB) examples
 - Short Answer (SA) examples

Complete list of meta-characters and their behaviour

Introduction to regular expressions

The Learning Environment's Quizzes tool uses regular expressions to give users grading a quiz the ability to evaluate answers for short answer and fill-in-the-blank questions against a set of acceptable values.

A regular expression uses alpha-numeric and special characters, known as meta-characters to create a pattern that describes one or more strings to match when searching a body of text. The regular expression is a template for matching a character pattern in the text you are searching. This helps automate the grading process for instructors for short answer and fill-in-the-blank questions.

Note Short answer and fill-in-the-blank questions are the only two question types for which you can use regular expressions.

Examples of regular expressions

Fill in the Blanks (FIB) examples

Question A ____ wags his tail. He eats dog _____ twice a day.

Answer Blank 1 = [D|d]og. Blank 2 = [F|f]ood

Question The classic movie "Close Encounter of Third Kind" was directed by none other than Steven _____ who also directed E.T and Indiana Jones series.

Answer [S|s]pielberg

Short Answer (SA) examples

Question What word describes red, blue, green, yellow, pink, etc.?

Answer colour?r*

Question What kind of animal meows?

Answer The answer used in the sample question: [C|c]at. Size used in the sample question: row size = 2, column size = 20

► To use regular expressions

- 1 In a short answer (SA) or fill-in-the-blank (FIB) question, select the **Regular Expression** radio button under **Evaluation**.
- 2 For short answer questions choose the size of the answer field using the **Rows** and **Columns** drop-down menus.
- 3 Type the answer in the **Answer** field using meta-characters to set the allowable values. If you have multiple blanks in a FIB question type repeat this to fill all **Answer** fields.

- 4 Click **Save** (or click **Preview**, to see the question before it is saved).

Complete list of meta-characters and their behaviour

Character	Description	Example
\	Marks the next character as a special character, a literal, a back reference, or an octal escape.	The sequence '\\ matches "\" and \"\" matches "(\" n matches the character n . \n matches a newline character.
^	Matches the position at the beginning of the input string. If the RegExp object's Multiline property is set, ^ also matches the position following '\n' or '\r'.	^cat matches strings that begin with cat
\$	Matches the position at the end of the input string. If the RegExp object's Multiline property is set, \$ also matches the position preceding '\n' or '\r'.	cat\$ matches any string that ends with cat
*	Matches the preceding character or sub-expression zero or more times. * equals {0,}	be* matches b or be or beeeeeeeeee zo* matches z and zoo .
+	Matches the preceding character or sub-expression one or more times. + equals {1,}.	be+ matches be or bee but not b
?	Matches the preceding character or sub-expression zero or one time. ? equals {0,1}	abc? matches ab or abc colou?r matches color or colour but not colourr do(es)? matches the do in do or does .
?	When this character immediately follows any of the other quantifiers (*, +, ?, {n}, {n,}, {n,m}), the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible.	In the string oooo , o+? matches a single o , while o+ matches all os .
()	Parentheses create a substring or item that you can apply meta-characters to.	a(bee)?t matches at or abeet but not abet

Character	Description	Example
{n}	n is a nonnegative integer. Matches exactly n times.	[0-9]{3} matches any three digits o{2} does not match the o in Bob , but matches the two os in food . b{4} matches bbbb
{n,}	n is a nonnegative integer. Matches at least n times.	[0-9]{3,} matches any three or more digits o{2,} does not match the "o" in "Bob" and matches all the o 's in "foooood". 'o{1,}' is equivalent to 'o+'. 'o{0,}' is equivalent to 'o*'. o{1,3} matches the first three o's in "foooood". 'o{0,1}' is equivalent to 'o?'. c{2, 4} matches cc, ccc, cccc
{n,m}	m and n are nonnegative integers, where n <= m. Matches at least n and at most m times. Note You cannot put a space between the comma and the numbers.	[0-9]{3,5} matches any three, four, or five digits o{1,3} matches the first three o's in "foooood". 'o{0,1}' is equivalent to 'o?'. c{2, 4} matches cc, ccc, cccc
.	Matches any single character except "\n". To match any character including the '\n', use a pattern such as '[\s\S]'.	cat. matches catT and cat2 but not catty
(?)	Makes the remainder of the regular expression case insensitive.	ca(?)se matches caSE but not CASE
(pattern)	Matches pattern and captures the match. The captured match can be retrieved from the resulting Matches collection, using the SubMatches collection in VBScript or the \$0-\$9 properties in JScript. To match parentheses characters (), use '\(' or '\)'	(jam){2} matches jamjam . First group matches jam .
(?:pattern)	Matches pattern but does not capture the match, that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character ().	'industr(?:y ies) is a more economical expression than 'industry industries' .
(?=pattern)	Positive lookahead matches the search string at any point where a string matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.	'Windows (?:=95 98 NT 2000)' matches "Windows" in "Windows 2000" but not "Windows" in "Windows 3.1" .

Character	Description	Example
(?!pattern)	Negative lookahead matches the search string at any point where a string not matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.	'Windows (?!95 98 NT 2000)' matches "Windows" in "Windows 3.1" but does not match "Windows" in "Windows 2000" .
x y	Matches x or y.	July (first 1st 1) will match July 1st but not July 2 'z food' matches "z" or "food" . '(z f)ood' matches "zood" or "food" .
[xyz]	A character set. Matches any one of the enclosed characters.	gr[ae]y matches gray or grey '[abc]' matches the 'a' in "plain" .
[^xyz]	A negative character set. Matches any character not enclosed.	1[^02] matches 13 or 11 but not 10 or 12 '[^abc]' matches the 'p' in "plain" .
[a-z]	A range of characters. Matches any character in the specified range.	[1-9] matches any single digit EXCEPT 0 '[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z' .
[^a-z]	A negative range characters. Matches any character not in the specified range.	'[^a-z]' matches any character not in the range 'a' through 'z'
\b	Matches a word boundary, that is, the position between a word and a space.	'er\b' matches the 'er' in "never" but not the 'er' in "verb" .
\B	Matches a nonword boundary.	'er\B' matches the 'er' in "verb" but not the 'er' in "never" .
\cx	Matches the control character indicated by x. The value of x must be in the range of A-Z or a-z. If not, c is assumed to be a literal 'c' character.	\cM matches a Control-M or carriage return character .
\d	Matches a digit character. Equivalent to [0-9]	
\D	Matches a nondigit character Equivalent to [^0-9]	

Character	Description	Example
\f	Matches a form-feed character. Equivalent to \x0c and \cL	
\n	Matches a newline character. Equivalent to \x0a and \cJ	
\r	Matches a carriage return character. Equivalent to \x0d and \cM	
\s	Matches any whitespace character including space, tab, form-feed, etc. Equivalent to [\f\n\r\t\v]	Can be combined in the same way as [\d\s], which matches a character that is a digit or whitespace .
\S	Matches any non-white space character. Equivalent to [^\f\n\r\t\v]	
\t	Matches a tab character. Equivalent to \x09 and \cI	
\v	Matches a vertical tab character. Equivalent to \x0b and \cK	
\w	Matches any word character including underscore. Equivalent to '[A-Za-z0-9_]'	
\W	Matches any nonword character. Equivalent to '[^A-Za-z0-9_]'	
	You should only use \D, \W and \S outside character classes.	
\Z	Matches the end of the string the regular expression is applied to. Matches a position, but never matches before line breaks.	\Z matches k in joI\hok
\xn	Matches n, where n is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. Allows ASCII codes to be used in regular expressions.	'\x41' matches "A". '\x041' is equivalent to '\x04' & "1"
\num	Matches num, where num is a positive integer. A reference back to captured matches.	'(.)\1' matches two consecutive identical characters

Character	Description	Example
<code>\n</code>	<p>Identifies either an octal escape value or a backreference.</p> <p>If <code>\n</code> is preceded by at least <code>n</code> captured subexpressions, <code>n</code> is a backreference.</p> <p>Otherwise, <code>n</code> is an octal escape value if <code>n</code> is an octal digit (0-7).</p>	<p>“<code>\11</code>” and “<code>\011</code>” both match a tab character. “<code>\0011</code>” is the equivalent of “<code>\001</code>” & “<code>1</code>”.</p>
<code>\nm</code>	<p>Identifies either an octal escape value or a backreference.</p> <p>If <code>\nm</code> is preceded by at least <code>nm</code> captured subexpressions, <code>nm</code> is a backreference.</p> <p>If <code>\nm</code> is preceded by at least <code>n</code> captures, <code>n</code> is a backreference followed by literal <code>m</code>.</p> <p>If neither of the preceding conditions exists, <code>\nm</code> matches octal escape value <code>nm</code> when <code>n</code> and <code>m</code> are octal digits (0-7).</p>	
<code>\nml</code>	<p>Matches octal escape value <code>nml</code> when <code>n</code> is an octal digit (0-3) and <code>m</code> and <code>l</code> are octal digits (0-7).</p>	
<code>\un</code>	<p>Matches <code>n</code>, where <code>n</code> is a Unicode character expressed as four hexadecimal digits.</p>	<p>For example, <code>\u00A9</code> matches the copyright symbol (©).</p>