regexp, regexpi

Match regular expression

Syntax

```
regexp(parseStr, matchExpr)
[startIndex, endIndex, tokIndex, matchStr, tokenStr,
exprNames, splitStr] = regexp( parseStr, matchExpr)
[outVal1, outVal5, ...] = regexp( str, expr, outSel1, outSel5,
...)
[v1 v2 ...] = regexp( str, expr, ..., options)
```

Description

Each of the above syntaxes applies to both `regexp` and `regexpi`. The `regexp` function is case sensitive in matching regular expressions to a string, and `regexpi` is case insensitive.

`regexp(`*`parseStr`*`, `*`matchExpr`*`)` returns a row vector containing the starting index of each substring of *`parseStr`* that matches the regular expression string *`matchExpr`*. If no matches are found, `regexp` returns an empty array. The *`parseStr`* and *`matchExpr`* arguments can also be cell arrays of strings. See Regular Expressions in the MATLAB Programming Fundamentals documentation for more information.

To specify more than one string to parse or more than one expression to match, see the guidelines listed below under Multiple Strings or Expressions.

`[startIndex, endIndex, tokIndex, matchStr, tokenStr, exprNames, splitStr] = regexp(`*`parseStr`*`, `*`matchExpr`*`)` returns up to six values, one for each output variable you specify, and in the default order (as shown in the table below).

> **Note** The *`str`* and *`expr`* inputs are required and must be entered as the first and second arguments, respectively. Any other input arguments (all are described below) are optional and can be entered following the two required inputs in any order.

`[outVal1, outVal5, ...] = regexp( `*`str`*`, `*`expr`*`, `*`outSel1`*`, `*`outSel5`*`, ...)` returns up to six values, one for each output variable you specify, and ordered according to the order of the qualifier arguments, *`q1`*, *`q2`*, etc.

> **Tip** When using the `split` option, `regexp` always returns one more string than it does with the `match` option. Also, you can always put the original input string back together from the substrings obtained from both `split` and `match`. See Example 4 — Splitting the Input String.

`[v1 v2 ...] = regexp( `*`str`*`, `*`expr`*`, ..., `*`options`*`)` calls `regexp` with one or more of the nondefault options listed in the following table. These options must follow *`str`* and *`expr`* in the input argument list.

Option	Description
`mode`	See the section on `Modes` under `Inputs`, below.
'once'	Return only the first match found.
'warnings'	Display any hidden warning messages issued by MATLAB during the execution of the command. This option only enables warnings for the one command being executed. See Example 11 — Displaying Parsing Warnings.

Input Arguments

str

A string MATLAB that searches for a substring that matches the regular expression. It can be of any length and may contain any characters.

expr

A combination of text and operators that enable you to specify the content of the phrase you are looking for in the parse string. Any text in the expression must be an exact match for at least part of the text in the parse string. Operators, on the other hand, are symbolic. Each operator symbol stands for a *type* of character (e.g., an uppercase letter (`[A-Z]`), a space character (`\s`), four characters of any type (`.{4}`)).

MATLAB parses the input string from left to right, attempting to match text in the string with the first element of the regular expression. During this process, **MATLAB** skips over any text that does not match. When it finds the first match, it continues parsing the string, this time attempting to match the second piece of the expression, and so on. If characters are detected in the string that do not match the expression, then MATLAB drops the current match candidate and again starts looking for a match with the first element of the expression.

outputSelect

One to seven keywords with which you can select which output values `regexp` is to return and in what order.

Qualifier	Description	Default Order
start	Row vector containing the starting index of each substring of `str` that matches `expr`.	1
end	Row vector containing the ending index of each substring of `str` that matches `expr`.	2

tokenExtents	Cell array containing the starting and ending indices of each substring of `str` that matches a token in `expr`. (This is a `double` array when used with `'once'`.)	3
match	Cell array containing the text of each substring of `str` that matches `expr`. (This is a string when used with `'once'`.)	4
tokens	Cell array of cell arrays of strings containing the text of each token captured by `regexp`. (This is a cell array of strings when used with `'once'`.)	5
names	Structure array containing the name and text of each *named* token captured by `regexp`. If there are no named tokens in `expr`, `regexp` returns a structure array with no fields. Field names of the returned structure are set to the token names, and field values are the text of those tokens. Named tokens are generated by the expression `(?<tokenname>)`.	6
split	Cell array containing those parts of the input string that are delimited by substrings returned when using the `regexp 'match'` option.	7

mode

You can specify one or more of the following modes with the `regexp`, `regexpi`, and `regexprep` functions. You can enable or disable any of these modes using the mode specifier keyword (e.g., `'lineanchors'`) or the mode flag (e.g., `(?m)`). Both are shown in the tables that follow. Use the keyword to enable or disable the mode for the entire string being parsed. Use the flag to both enable and disable the mode for selected pieces of the string.

For more information about modes, see Modifying Parameters of the Search in the MATLAB "Programming Fundamentals" documentation.

Case-Sensitivity Mode

Use the Case-Sensitivity mode to control whether or not MATLAB considers letter case when matching an expression to a string. Example 7 — Using the Case-Sensitive Mode illustrates this mode.

Mode Keyword	Flag	Description
'**matchcase**'	`(?-i)`	Letter case must match when matching patterns to a string. (The default for `regexp`).
'**ignorecase**'	`(?i)`	Do not consider letter case when matching patterns to a string. (The default for `regexpi`).

Dot Matching Mode

Use the Dot Matching mode to control whether or not MATLAB includes the newline (`\n`) character when matching the dot (`.`) metacharacter in a regular expression. Example 8 — Using the Dot Matching Mode illustrates the Dot Matching mode.

Mode Keyword	Flag	Description
'**dotall**'	`(?s)`	Match dot ('`.`') in the pattern string with any character. (This is the default).
'**dotexceptnewline**'	`(?-s)`	Match dot in the pattern with any character that is not a newline.

Anchor Type Mode

Use the Anchor Type mode to control whether MATLAB considers the `^` and `$` metacharacters to represent the beginning and end of a string or the beginning and end of a line. Example 9 — Using the Anchor Type Mode illustrates the Anchor mode.

Mode Keyword	Flag	Description
'**stringanchors**'	`(?-m)`	Match the `^` and `$` metacharacters at the beginning and end of a string. (This is the default).
'**lineanchors**'	`(?m)`	Match the `^` and `$` metacharacters at the beginning and end of a line.

Spacing Mode

Use the Spacing mode to control how MATLAB interprets space characters and comments within the parsing string. Note that spacing mode applies to the parsing string (the second input argument that contains the metacharacters (e.g., `\w`) and not the string being parsed. Example 10 — Using the Spacing Mode illustrates the Spacing mode.

Mode Keyword	Flag	Description
'**literalspacing**'	`(?-x)`	Parse space characters and comments (the `#` character and any text to the right of it) in the same way as any other characters in the string. (This is the default).
'**freespacing**'	`(?x)`	Ignore spaces and comments when parsing the string. (You must use `'\ '` and `'\#'` to match space and `#` characters.)

`once`	Specify the `'once'` option to return only the first match found from the parse.
`warning`	Display any hidden warning messages issued by MATLAB during the execution of the command. This option only enables warnings for the one command being executed.

Output Arguments

Return Values for Regular Expressions

Default Order	Description	Qualifier
1	Row vector containing the starting index of each substring of `str` that matches `expr`.	**start**
2	Row vector containing the ending index of each substring of `str` that matches `expr`.	**end**
3	Cell array containing the starting and ending indices of each substring of `str` that matches a token in `expr`. (This is a `double` array when used with `'once'`.)	**tokenExtents**

4	Cell array containing the text of each substring of `str` that matches `expr`. (This is a string when used with `'once'`.)	**`match`**
5	Cell array of cell arrays of strings containing the text of each token captured by `regexp`. (This is a cell array of strings when used with `'once'`.)	**`tokens`**
6	Structure array containing the name and text of each *named* token captured by `regexp`. If there are no named tokens in `expr`, `regexp` returns a structure array with no fields. Field names of the returned structure are set to the token names, and field values are the text of those tokens. Named tokens are generated by the expression `(?<tokenname>)`.	**`names`**
7	Cell array containing those parts of the input string that are delimited by substrings returned when using the `regexp 'match'` option.	**`split`**

`endIndex`	Row vector containing the ending index of each substring of str that matches expr.
`tokenExtents`	Cell array containing the starting and ending indices of each substring of str that matches a token in expr. (This is a double array when used with 'once'.)
`matchString`	Cell array containing the text of each substring of str that matches expr. (This is a string when used with 'once'.)
`tokenStrings`	Cell array of cell arrays of strings containing the text of each token captured by regexp. (This is a cell array of strings when used with 'once'.)
`tokenNames`	Structure array containing the name and text of each named token captured by regexp. If there are no named tokens in expr, regexp returns a structure array with no fields. Field names of the returned structure are set to the token names, and field values are the text of those tokens. Named tokens are generated by the expression `(?<tokenName>)`.
`splitString`	Cell array containing those parts of the input string that are delimited by substrings returned when using the regexp 'match' option.

Remarks

See Regular Expressions in the MATLAB Programming Fundamentals documentation for a listing of all regular expression elements supported by MATLAB.

Multiple Strings or Expressions

Either the `str` or `expr` argument, or both, can be a cell array of strings, according to the following guidelines:

- If `str` is a cell array of strings, then each of the `regexp` outputs is a cell array having the same dimensions as `str`.
- If `str` is a single string but `expr` is a cell array of strings, then each of the `regexp` outputs is a cell array having the same dimensions as `expr`.
- If both `str` and `expr` are cell arrays of strings, these two cell arrays must contain the same number of elements.

Examples

For more examples than those shown below, see Regular Expressions in the MATLAB Programming Fundamentals documentation.

Example 1 — Matching a Simple Pattern

Return a row vector of indices that match words that start with `c`, end with `t`, and contain one or more vowels between them. Make the matches insensitive to letter case (by using `regexpi`):

```
str = 'bat cat can car COAT court cut ct CAT-scan';
regexpi(str, 'c[aeiou]+t')
ans =
     5    17    28    35
```

Example 2 — Parsing Multiple Input Strings

Return a cell array of row vectors of indices that match capital letters and white spaces in the cell array of strings `str`:

```
str = {'Madrid, Spain' 'Romeo and Juliet' 'MATLAB is great'};
s1 = regexp(str, '[A-Z]');
s2 = regexp(str, '\s');
```

Capital letters, `'[A-Z]'`, were found at these `str` indices:

```
s1{:}
ans =
     1     9
ans =
     1    11
ans =
     1     2     3     4     5     6
```

Space characters, `'\s'`, were found at these `str` indices:

```
s2{:}
ans =
     8
ans =
     6    10
ans =
     7    10
```

Example 3 — Selecting Return Values

Return the text and the starting and ending indices of words containing the letter `x`:

```
str = 'regexp helps you relax';
[m s e] = regexp(str, '\w*x\w*', 'match', 'start', 'end')
m =
    'regexp'    'relax'
s =
     1    18
e =
     6    22
```

Example 4 — Splitting the Input String

Find the substrings delimited by the `^` character:

```
s1 = ['Use REGEXP to split ^this string into ' ...
      'several ^individual pieces'];

s2 = regexp(s1, '\^', 'split');

s2(:)
ans =
    'Use REGEXP to split '
    'this string into several '
    'individual pieces'
```

The `split` option returns those parts of the input string that are not returned when using the `'match'` option. Note that when you match the beginning or ending characters in a string (as is done in this example), the first (or last) return value is always an empty string:

```
str = 'She sells sea shells by the seashore.';

[matchstr splitstr] = regexp(str, '[Ss]h.', 'match', ...
                             'split')
matchstr =
    'She'    'she'    'sho'
splitstr =
     ''    ' sells sea '    'lls by the sea'    're.'
```

For any string that has been split, you can reassemble the pieces into the initial string using

the command

```
j = [splitstr; [matchstr {''}]]; [j{:}]

ans =
   She sells sea shells by the seashore.
```

Example 5 — Using Tokens

Search a string for opening and closing HTML tags. Use the expression `<(\w+)` to find the opening tag (e.g., `'<tagname'`) and to create a token for it. Use the expression `</\1>` to find another occurrence of the same token, but formatted as a closing tag (e.g., `'</tagname>'`):

```
str = ['if <code>A</code> == x<sup>2</sup>, ' ...
       '<em>disp(x)</em>']
str =
if <code>A</code> == x<sup>2</sup>, <em>disp(x)</em>

expr = '<(\w+).*?>.*?</\1>';

[tok mat] = regexp(str, expr, 'tokens', 'match');

tok{:}
ans =
    'code'
ans =
    'sup'
ans =
    'em'

mat{:}
ans =
    <code>A</code>
ans =
    <sup>2</sup>
ans =
    <em>disp(x)</em>
```

See Tokens in the MATLAB Programming Fundamentals documentation for information on using tokens.

Example 6 — Using Named Capture

Enter a string containing two names, the first and last names being in a different order:

```
str = sprintf('John Davis\nRogers, James')
str =
    John Davis
    Rogers, James
```

Create an expression that generates first and last name tokens, assigning the names `first` and `last` to the tokens. Call `regexp` to get the text and names of each token found:

```
expr = ...
   '(?<first>\w+)\s+(?<last>\w+)|(?<last>\w+),\s+(?<first>\w+)';

[tokens names] = regexp(str, expr, 'tokens', 'names');
```

Examine the `tokens` cell array that was returned. The first and last name tokens appear in the order in which they were generated: first name–last name, then last name–first name:

```
tokens{:}
ans =
    'John'    'Davis'
ans =
    'Rogers'    'James'
```

Now examine the `names` structure that was returned. First and last names appear in a more usable order:

```
names(:,1)
ans =
    first: 'John'
     last: 'Davis'

names(:,2)
ans =
    first: 'James'
     last: 'Rogers'
```

Example 7 — Using the Case-Sensitive Mode

Given a string that has both uppercase and lowercase letters,

```
str = 'A string with UPPERCASE and lowercase text.';
```

Use the `regexp` default mode (case-sensitive) to locate only the lowercase instance of the word `case`:

```
regexp(str, 'case', 'match')
ans =
    'case'
```

Now disable case-sensitive matching to find both instances of `case`:

```
regexp(str, 'case', 'ignorecase', 'match')
ans =
    'CASE'    'case'
```

Match 5 letters that are followed by '`CASE`'. Use the `(?-i)` flag to turn on case-sensitivity

for the first match and `(?i)` to turn it off for the second:

```
M = regexp(str, {'(?-i)\w{5}(?=CASE)', ...
                 '(?i)\w{5}(?=CASE)'}, 'match');

M{:}
ans =
    'UPPER'
ans =
    'UPPER'    'lower'
```

Example 8 — Using the Dot Matching Mode

Parse the following string that contains a newline (`\n`) character:

```
str = sprintf('abc\ndef')
str =
   abc
   def
```

When you use the default mode, `dotall`, MATLAB includes the newline in the characters matched:

```
regexp(str, '.', 'match')
ans =
   'a'   'b'   'c'   [1x1 char]   'd'   'e'   'f'
```

When you use the `dotexceptnewline` mode, MATLAB skips the newline character:

```
regexp(str, '.', 'match', 'dotexceptnewline')
ans =
   'a'   'b'   'c'   'd'   'e'   'f'
```

Example 9 — Using the Anchor Type Mode

Given the following two-line string,

```
str = sprintf('%s\n%s', 'Here is the first line', ...
              'followed by the second line')
str =
   Here is the first line
   followed by the second line
```

In `stringanchors` mode, MATLAB interprets the `$` metacharacter as an end-of-string specifier, and thus finds the last two words of the entire *string*:

```
regexp(str, '\w+\W\w+$', 'match', 'stringanchors')
ans =
    'second line'
```

While in `lineanchors` mode, MATLAB interprets `$` as an end-of-line specifier, and finds the last two words of each *line*:

```
regexp(str, '\w+\W\w+$', 'match', 'lineanchors')
ans =
    'first line'    'second line'
```

Example 10 — Using the Spacing Mode

Create a file called `regexp_str.txt` containing the following text.

```
(?x)    # turn on freespacing.

# This pattern matches a string with a repeated letter.

\w*     # First, match any number of preceding word characters.

(       # Mark a token.
.       # Match a character of any type.
)       # Finish capturing said token.
\1      # Backreference to match what token #1 matched.

\w*     # Finally, match the remainder of the word.
```

Because the first line enables freespacing mode, MATLAB ignores all spaces and comments that appear in the file. Here is the string to parse:

```
str = ['Looking for words with letters that ' ...
       'appear twice in succession.'];
```

Use the pattern expression read from the file to find those words that have consecutive matching letters:

```
patt = fileread('regexp_str.txt');
regexp(str, patt, 'match')
ans =
    'Looking'    'letters'    'appear'    'succession'
```

Example 11 — Displaying Parsing Warnings

To help debug problems in parsing a string with `regexp`, `regexpi`, or `regexprep`, use the '`warnings`' option to view all warning messages:

```
regexp('$.', '[a-]','warnings')
Warning: Unbound range.
 [a-]
   |
```

See Also

Regular Expressions, `regexprep`, `regexptranslate`, `strfind`, `strcmp`, `strcmpi`, `strncmp`, `strncmpi`