RegEx

<https://regexone.com>

Table of Contents

[abc… Letters 4](#_Toc37115579)

[123… Digits 4](#_Toc37115580)

[\d Any Digit 4](#_Toc37115581)

[\D Any Non-digit character 4](#_Toc37115582)

[. Any Character 4](#_Toc37115583)

[\. Period 4](#_Toc37115584)

[[abc] Only a, b, or c 4](#_Toc37115585)

[[^abc] Not a, b, nor c 5](#_Toc37115586)

[[a-z] Characters a to z 5](#_Toc37115587)

[[0-9] Numbers 0 to 9 5](#_Toc37115588)

[\w Any Alphanumeric character 5](#_Toc37115589)

[\W Any Non-alphanumeric character 5](#_Toc37115590)

[{m} m Repetitions 5](#_Toc37115591)

[{m,n} m to n Repetitions 5](#_Toc37115592)

[\* Zero or more repetitions 6](#_Toc37115593)

[+ One or more repetitions 6](#_Toc37115594)

[? Optional character 6](#_Toc37115595)

[\s Any Whitespace 6](#_Toc37115596)

[\S Any Non-whitespace character 6](#_Toc37115597)

[^…$ Starts and ends 6](#_Toc37115598)

[(…) Capture Group 7](#_Toc37115599)

[(a(bc)) Capture Sub-group 7](#_Toc37115600)

[(.\*) Capture all 7](#_Toc37115601)

[(abc|def) Matches abc or def 7](#_Toc37115602)

[decimal numbers 7](#_Toc37115603)

[phone numbers 8](#_Toc37115604)

[emails 8](#_Toc37115605)

[Html 1 8](#_Toc37115606)

[Html 2 9](#_Toc37115607)

[Html 3 9](#_Toc37115608)

[filenames 9](#_Toc37115609)

[Trimming whitespace from start and end of line 10](#_Toc37115610)

[Extracting information from a log file 10](#_Toc37115611)

[Url 1 10](#_Toc37115612)

[Url 2 11](#_Toc37115613)

[Url 3 11](#_Toc37115614)

[Url 4 11](#_Toc37115615)

[Using Regular Expressions in C# 12](#_Toc37115616)

[Verbatim String Literals 12](#_Toc37115617)

[Matching a string 12](#_Toc37115618)

[Capturing groups 13](#_Toc37115619)

[Finding and replacing strings 14](#_Toc37115620)

[RegexOptions Enums 15](#_Toc37115621)

[Compiling a pattern for performance 16](#_Toc37115622)

[Links 16](#_Toc37115623)

# abc… Letters

|  |  |  |
| --- | --- | --- |
| Match | abcdefg | Success |
| Match | Abcde | Success |
| Match | Abc |  |
|  | abc |  |

# 123… Digits

# \d Any Digit

# \D Any Non-digit character

|  |  |  |
| --- | --- | --- |
| Match | abc123xyz | Success |
| Match | define "123" | Success |
| Match | var g = 123; | Success |
|  | \d\d\d or 123 |  |

Top of Form

Bottom of Form

# . Any Character

# \. Period

|  |  |  |
| --- | --- | --- |
| Match | cat. | Success |
| Match | 896. | Success |
| Match | ?=+. | Success |
| Skip | abc1 | To be completed |
|  | ...\. |  |

Top of Form

Bottom of Form

# [abc] Only a, b, or c

|  |  |  |
| --- | --- | --- |
| Match | can | Success |
| Match | man | Success |
| Match | fan | Success |
| Skip | dan | To be completed |
| Skip | ran | To be completed |
| Skip | pan |  |
|  | [cmf]an or [^drp]an |  |

# [^abc] Not a, b, nor c

|  |  |  |
| --- | --- | --- |
| Match | hog | Success |
| Match | dog | Success |
| Skip | bog | To be completed |
|  | [^b]og or [hd]og] |  |

Top of Form

Bottom of Form

# [a-z] Characters a to z

# [0-9] Numbers 0 to 9

# \w Any Alphanumeric character

# \W Any Non-alphanumeric character

|  |  |  |
| --- | --- | --- |
| Match | Ana | Success |
| Match | Bob | Success |
| Match | Cpc | Success |
| Skip | aax | To be completed |
| Skip | bby | To be completed |
| Skip | ccz |  |
|  | [A-C][n-p][a-c] |  |

# {m} m Repetitions

# {m,n} m to n Repetitions

|  |  |  |
| --- | --- | --- |
| Match | wazzzzzup | Success |
| Match | wazzzup | Success |
| Skip | wazup |  |
|  | waz{3,5}up |  |

# \* Zero or more repetitions

# + One or more repetitions

|  |  |  |
| --- | --- | --- |
| Match | aaaabcc | Success |
| Match | aabbbbc | Success |
| Match | aacc | Success |
| Skip | a | To be completed |
|  | '[aa+b\*c+](https://regexone.com/lesson/kleene_operators) or [a{2,4}b{0,4}c{1,2}](https://regexone.com/lesson/kleene_operators) |  |

Top of Form

Bottom of Form

# ? Optional character

|  |  |  |
| --- | --- | --- |
| Match | 1 file found? | Success |
| Match | 2 files found? | Success |
| Match | 24 files found? | Success |
| Skip | No files found. | To be completed |
|  | \d+ files? found\? |  |

Top of Form

Bottom of Form

# \s Any Whitespace

# \S Any Non-whitespace character

|  |  |  |
| --- | --- | --- |
| Match | 1. abc | Success |
| Match | 2. abc | Success |
| Match | 3. abc | Success |
| Skip | 4.abc |  |
|  | \d\.\s+abc |  |

# ^…$ Starts and ends

|  |  |  |
| --- | --- | --- |
| Match | Mission: successful | Success |
| Skip | Last Mission: unsuccessful | To be completed |
| Skip | Next Mission: successful upon capture of target | To be completed |
|  | ^Mission: successful$ |  |

Top of Form

Bottom of Form

# (…) Capture Group

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | file\_record\_transcript.pdf | file\_record\_transcript | Success |
| Capture | file\_07241999.pdf | file\_07241999 | Success |
| Skip | testfile\_fake.pdf.tmp |  | To be completed |
|  | ^(file.+)\.pdf$ |  |  |

Top of Form

Bottom of Form

# (a(bc)) Capture Sub-group

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | Jan 1987 | Jan 1987 1987 | Success |
| Capture | May 1969 | May 1969 1969 | Success |
| Capture | Aug 2011 | Aug 2011 2011 | Success |
|  | (\w+ (\d+)) |  |  |

Top of Form

Bottom of Form

# (.\*) Capture all

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | 1280x720 | 1280 720 | Success |
| Capture | 1920x1600 | 1920 1600 | Success |
| Capture | 1024x768 | 1024 768 | Success |
|  | (\d+)x(\d+) |  |  |

Top of Form

Bottom of Form

# (abc|def) Matches abc or def

|  |  |  |
| --- | --- | --- |
| Match | I love cats | Success |
| Match | I love dogs | Success |
| Skip | I love logs | To be completed |
| Skip | I love cogs |  |
|  | I love (cats|dogs) |  |

# decimal numbers

|  |  |  |
| --- | --- | --- |
| Match | 3.14529 | Success |
| Match | -255.34 | Success |
| Match | 128 | Success |
| Match | 1.9e10 | Success |
| Match | 123,340.00 | Success |
| Skip | 720p |  |
|  | ^-?\d+(,\d+)\*(\.\d+(e\d+)?)?$ |  |

# phone numbers

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | 415-555-1234 | 415 | Success |
| Capture | 650-555-2345 | 650 | Success |
| Capture | (416)555-3456 | 416 | Success |
| Capture | 202 555 4567 | 202 | Success |
| Capture | 4035555678 | 403 | Success |
| Capture | 1 416 555 9292 | 416 | Success |
|  | 1?[\s-]?\(?(\d{3})\)?[\s-]?\d{3}[\s-]?\d{4} |  |  |

Top of Form

Bottom of Form

# emails

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | tom@hogwarts.com | tom | Success |
| Capture | tom.riddle@hogwarts.com | tom.riddle | Success |
| Capture | tom.riddle+regexone@hogwarts.com | tom.riddle | Success |
| Capture | tom@hogwarts.eu.com | tom | Success |
| Capture | potter@hogwarts.com | potter | Success |
| Capture | harry@hogwarts.com | harry | Success |
| Capture | hermione+regexone@hogwarts.com | hermione | Success |
|  | ^([\w\.]\*) |  |  |

Top of Form

Bottom of Form

# Html 1

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | <a>This is a link</a> | a | Success |
| Capture | <a href='https://regexone.com'>Link</a> | a | Success |
| Capture | <div class='test\_style'>Test</div> | div | Success |
| Capture | <div>Hello <span>world</span></div> | div | Success |
|  | <(\w+) |  |  |

# Html 2

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | <a>This is a link</a> | a | Failed |
| Capture | <a href='https://regexone.com'>Link</a> | a | Failed |
| Capture | <div class='test\_style'>Test</div> | div | Failed |
| Capture | <div>Hello <span>world</span></div> | div | Failed |
|  | >([\w\s]\*)< |  |  |

Top of Form

Bottom of Form

# Html 3

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | <a>This is a link</a> | a | Failed |
| Capture | <a href='https://regexone.com'>Link</a> | a | Failed |
| Capture | <div class='test\_style'>Test</div> | div | Failed |
| Capture | <div>Hello <span>world</span></div> | div | Failed |
|  | ='([\w://.]\*)' |  |  |

# filenames

|  |  |  |  |
| --- | --- | --- | --- |
| Skip | .bash\_profile |  | To be completed |
| Skip | workspace.doc |  | To be completed |
| Capture | img0912.jpg | img0912 jpg | Success |
| Capture | updated\_img0912.png | updated\_img0912png | Success |
| Skip | documentation.html |  | To be completed |
| Capture | favicon.gif | favicon gif | Success |
| Skip | img0912.jpg.tmp |  | To be completed |
| Skip | access.lock |  | To be completed |
|  | (\w+)\.(jpg|png|gif)$ |  |  |

# Trimming whitespace from start and end of line

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | The quick brown fox... | The quick brown fox... | Success |
| Capture | jumps over the lazy dog. | jumps over the lazy dog. | Success |
|  | ^\s\*(.\*)\s\*$ |  |  |

# Extracting information from a log file

|  |  |  |  |
| --- | --- | --- | --- |
| Skip | W/dalvikvm( 1553): threadid=1: uncaught exception |  | To be completed |
| Skip | E/( 1553): FATAL EXCEPTION: main |  | To be completed |
| Skip | E/( 1553): java.lang.StringIndexOutOfBoundsException |  | To be completed |
| Capture | E/( 1553): at widget.List.makeView(ListView.java:1727) | makeViewListView.java1727 | Success |
| Capture | E/( 1553): at widget.List.fillDown(ListView.java:652) | fillDownListView.java652 | Success |
| Capture | E/( 1553): at widget.List.fillFrom(ListView.java:709) | fillFromListView.java709 | Success |
|  | (\w+)\(([\w\.]+):(\d+)\) |  |  |

# Url 1

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | ftp://file\_server.com:21/top\_secret/life\_changing\_plans.pdf | ftpfile\_server.com21 | Failed |
| Capture | https://regexone.com/lesson/introduction#section | httpsregexone.com | Failed |
| Capture | file://localhost:4040/zip\_file | file localhost4040 | Failed |
| Capture | https://s3cur3-server.com:9999/ | httpss3cur3-server.com9999 | Failed |
| Capture | market://search/angry%20birds | marketsearch | Failed |
|  | (\w+):// |  |  |

# Url 2

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | ftp://file\_server.com:21/top\_secret/life\_changing\_plans.pdf | ftpfile\_server.com21 | Failed |
| Capture | https://regexone.com/lesson/introduction#section | httpsregexone.com | Failed |
| Capture | file://localhost:4040/zip\_file | file localhost4040 | Failed |
| Capture | https://s3cur3-server.com:9999/ | httpss3cur3-server.com9999 | Failed |
| Capture | market://search/angry%20birds | marketsearch | Failed |
|  | ://([\w\-\.]+) |  |  |

# Url 3

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | ftp://file\_server.com:21/top\_secret/life\_changing\_plans.pdf | ftpfile\_server.com21 | Failed |
| Capture | https://regexone.com/lesson/introduction#section | httpsregexone.com | Failed |
| Capture | file://localhost:4040/zip\_file | file localhost4040 | Failed |
| Capture | https://s3cur3-server.com:9999/ | httpss3cur3-server.com9999 | Failed |
| Capture | market://search/angry%20birds | marketsearch | Failed |
|  | (:(\d+)) |  |  |

Top of Form

Bottom of Form

# Url 4

|  |  |  |  |
| --- | --- | --- | --- |
| Capture | ftp://file\_server.com:21/top\_secret/life\_changing\_plans.pdf | ftpfile\_server.com21 | Success |
| Capture | https://regexone.com/lesson/introduction#section | httpsregexone.com | Success |
| Capture | file://localhost:4040/zip\_file | file localhost4040 | Success |
| Capture | https://s3cur3-server.com:9999/ | httpss3cur3-server.com9999 | Success |
| Capture | market://search/angry%20birds | marketsearch | Success |
|  | (\w+)://([\w\-\.]+)(:(\d+))? |  |  |

# Using Regular Expressions in C#

If you need a refresher on how Regular Expressions work, check out our [Interactive Tutorial](https://regexone.com/lesson/introduction_abcs) first!

C# supports regular expressions through the classes in the [System.Text.RegularExpressions](https://msdn.microsoft.com/en-us/library/system.text.regularexpressions" \o "Documentation for C# System.Text.RegularExpressions namespace) namespace in the standard .NET framework. While there are some differences in advanced features supported by the .NET regular expression library compared to PCRE, they both share a large part of the syntax and patterns and expressions can be used in C# and other languages.

In the examples below, be sure to import the following namespaces at the top of the source file if you are trying out the code:

using System.Text.RegularExpressions;

## Verbatim String Literals

When writing regular expression in C#, it is recommended that you use [verbatim strings](https://msdn.microsoft.com/en-us/library/aa691090) instead of regular strings. Verbatim strings begin with a special prefix (@) and signal C# not to interpret backslashes and special metacharacters in the string, allowing you to pass them through directly to the regular expression engine.

This means that a pattern like "\n\w" will not be interpreted and can be written as @"\n\w" instead of "\\n\\w" as in other languages, which is much easier to read.

## Matching a string

In the System.Text.RegularExpressions namespace is a [Regex](https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex) class, which encapsulates the interface to the regular expressions engine and allows you to perform matches and extract information from text using regular expressions.

To test whether a regular expressions matches a string, you can use the static method [Regex.Match()](https://msdn.microsoft.com/en-us/library/bk1x0726" \o "Documentation for Regex.Match()) which takes an optional set of RegexOptions enums. This returns a [Match](https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.match) object which contains information about where the match (if any) was found.

Method

Match match = Regex.Match(InputStr, Pattern, RegexOptions)

Example

// Lets use a regular expression to match a date string.

string pattern = @"([a-zA-Z]+) (\d+)";

// The RegexOptions are optional to this call, we will go into more detail about

// them below.

Match result = Regex.Match("June 24", pattern);

if (result.Success) {

// Indeed, the expression "([a-zA-Z]+) (\d+)" matches the date string

// To get the indices of the match, you can read the Match object's

// Index and Length values.

// This will print [0, 7], since it matches at the beginning and end of the

// string

Console.WriteLine("Match at index [{0}, {1})",

result.Index,

result.Index + result.Length);

// To get the fully matched text, you can read the Match object's Value

// This will print "June 24"

Console.WriteLine("Match: {0}", result.Value);

// If you want to iterate over each of the matches, you can call the

// Match object's NextMatch() method which will return the next Match

// object.

// This will print out each of the matches sequentially.

while (result.Success) {

Console.WriteLine("Match: {0}", result.Value);

result = result.NextMatch();

}

}

## Capturing groups

If we wanted to perform a global search over the whole input string and return all the matches with their corresponding capture data, we can instead use the static method [Regex.Matches()](https://msdn.microsoft.com/en-us/library/b49yw9s8" \o "Documentation for Regex.Matches()) to get a [MatchCollection](https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.matchcollection" \o "Documentation for System.Text.RegularExpressions.MatchCollection) which can be iterated over and processed as in the example above.

Method

MatchCollection matches = Regex.Matches(InputStr, Pattern, RegexOptions)

Example

// Lets use a regular expression to capture data from a few date strings.

string pattern = @"([a-zA-Z]+) (\d+)";

MatchCollection matches = Regex.Matches("June 24, August 9, Dec 12", pattern);

// This will print the number of matches

Console.WriteLine("{0} matches", matches.Count);

// This will print each of the matches and the index in the input string

// where the match was found:

// June 24 at index [0, 7)

// August 9 at index [9, 17)

// Dec 12 at index [19, 25)

foreach (Match match in matches) {

Console.WriteLine("Match: {0} at index [{1}, {2})",

match.Value,

match.Index,

match.Index + match.Length);

}

// For each match, we can extract the captured information by reading the

// captured groups.

foreach (Match match in matches) {

GroupCollection data = match.Groups;

// This will print the number of captured groups in this match

Console.WriteLine("{0} groups captured in {1}", data.Count, match.Value);

// This will print the month and day of each match. Remember that the

// first group is always the whole matched text, so the month starts at

// index 1 instead.

Console.WriteLine("Month: " + data[1] + ", Day: " + data[2]);

// Each Group in the collection also has an Index and Length member,

// which stores where in the input string that the group was found.

Console.WriteLine("Month found at[{0}, {1})",

data[1].Index,

data[1].Index + data[1].Length);

}

## Finding and replacing strings

Another common task is to find and replace a part of a string using regular expressions, for example, to replace all instances of an old email domain, or to swap the order of some text. You can do this in C# with the static method [Regex.Replace()](https://msdn.microsoft.com/en-us/library/e7f5w83z" \o "Documentation for Regex.Replace()).

The replacement string can either be a regular expression that contains references to captured groups in the pattern, or just a regular string.

Method

string replaced = Regex.Replace(InputStr, Pattern, ReplacementPattern, RegexOption)

Example

// Lets try and reverse the order of the day and month in a few date

// strings. Notice how the replacement string also contains metacharacters

// (the back references to the captured groups) so we use a verbatim

// string for that as well.

string pattern = @"([a-zA-Z]+) (\d+)";

// This will reorder the string inline and print:

// 24 of June, 9 of August, 12 of Dec

// Remember that the first group is always the full matched text, so the

// month and day indices start from 1 instead of zero.

string replacedString = Regex.Replace("June 24, August 9, Dec 12",

pattern, @"$2 of $1");

Console.WriteLine(replacedString);

## RegexOptions Enums

In the regular expression methods above, you will notice that each of them also take an optional [RegexOptions](https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regexoptions" \o "Documentation for RegexOptions) argument. Most of the available flags are a convenience and can be written into the into the regular expression itself directly, but some can be useful in certain cases.

* RegexOptions.Compiled speeds up matching multiple input strings with the same pattern by allowing the regular expression engine to compile the pattern first. On by default if no options are set.
* RegexOptions.IgnoreCase makes the pattern case insensitive so that it matches strings of different capitalizations
* RegexOptions.Multiline is necessary if your input string has newline characters (*\n*) and allows the start and end metacharacter (*^* and *$* respectively) to match at the beginning and end of each line instead of at the beginning and end of the whole input string
* RegexOptions.RightToLeft is useful for matching in RTL languages
* RegexOptions.Singleline allows the dot (*.*) metacharacter match all characters, including the newline character (*\n*)

## Compiling a pattern for performance

While the static methods above for normal processing, if you are testing millions of input strings against the same pattern, you can reduce object allocations and get faster performance by instantiating a Regex object with the pattern in the constructor.

With this Regex object, all the same methods above are available on the object, except you will not have to pass in the pattern again with each call.

For more information, you can read the documentation on [Static vs Instance methods](https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex#static_vs_instance)on MSDN.

## Links

For more information about using regular expressions in C#, please visit the following links:

* [.NET System.Text.RegularExpression Namespace](https://msdn.microsoft.com/en-us/library/system.text.regularexpressions)
* [.NET Regular Expression Language Syntax (Quick Reference)](https://msdn.microsoft.com/en-us/library/az24scfc)