Regular expressions (RegEx)

Drawn from: support.google.com/analytics

About regular expressions (regex)

Analytics supports regular expressions so you can create more flexible definitions for things like view filters, goals, segments, audiences, content groups, and channel groupings.

In the context of Analytics, regular expressions are specific sequences of characters that broadly or narrowly match patterns in your Analytics data.

For example, if you wanted to create a view filter to exclude site data generated by your own employees, you could use a regular expression to exclude any data from the entire range of IP addresses that serve your employees. Let's say those IP addresses range from 198.51.100.1 - 198.51.100.25. Rather than enter 25 different IP addresses, you could create a regular expression like **198\.51\.100\.\d*** that matches the entire range of addresses.

Or if you wanted to create a view filter that included only campaign data from two different cities, you could create a regular expression like **San Francisco|New York** (San Francisco or New York).

Regex metacharacters

Wildcards

.	Matches any single character (letter, number or symbol)	1. matches 10, 1A 1.1 matches 111, 1A1 Examples
?	Matches the preceding character 0 or 1 times	10? matches 1, 10 Examples
+	Matches the preceding character 1 or more times	10+ matches 10, 100 Examples
*	Matches the preceding character 0 or more times	1* matches 1, 10 Examples
|	Creates an OR match Do not use at the end of an expression	1|10 matches 1, 10 Examples

Regular expressions (RegEx)

Anchors

^	Matches the adjacent characters at the beginning of a string	^10 matches 10, 100, 10x ^10 does not match 110, 110x Examples
$	Matches the adjacent characters at the end of a string	10$ matches 110, 1010 10$ does not match 100, 10x Examples

Groups

( )	Matches the enclosed characters in exact order anywhere in a string Also used to group other expressions	(10) matches 10, 101, 1011 ([0-9]|[a-z]) matches any number or lower-case letter Examples
[ ]	Matches the enclosed characters in any order anywhere in a string	[10] matches 012, 123, 202, 120, 210 Examples
-	Creates a range of characters within brackets to match anywhere in a string	[0-9] matches any number 0 through 9 Examples

Escape

\	Indicates that the adjacent character should be interpreted literally rather than as a regex metacharacter	\. indicates that the adjacent dot should be interpreted as a period or decimal rather than as a wildcard. 216\.239\.32\.34 matches

		216.239.32.34 Examples

Tips

Keep your regular expressions simple. Simple regex is easier for another user to interpret and modify.

Use the backslash (\) to escape regex metacharacters when you need those characters to be interpreted literally. For example, if you use a dot as the decimal separator in an IP address, escape it with a backslash (\.) so that it isn't interpreted as a wildcard.

Regular expressions don't have to include regex metacharacters. For example, you can create a segment for all data from India with the following filter definition: **Country matches regex India**

Regular expressions are greedy by nature: if you don't tell them not to, they match what you specify plus any adjacent characters. For example, site matches mysite, yoursite, theirsite, parasite--any string that contains "site". If you need to make a specific match, construct you regex accordingly. For example, if you need to match only the string "site", then construct your regex so that "site" is the both the beginning and end of the string: ^site$.

Parantheses ( ) and pipe |

Parentheses ( )

Use parentheses to group parts of an expression together.

For example, if you need to match a set of characters that appear in a number of different product SKUs, then you can group those characters together in parentheses. Say you have a beach sandal that you sell for men and women, and your product SKUs look like this:

MN**BS**010212 (men's beach sandal, style 01, color 02, size 12)

WM**BS**020208 (women's beach sandal, style 02, color 02, size 08)

You could create the following regular expression to capture all beach-sandal SKUs:

\D+(BS)\d+

\D (non-numeric character)

+ (one or more times)

(BS) (character code for beach sandal)

\d (numeric character)

+ (one or more times)

Pipe |

Use the pipe to create an OR condition in an expression.

For example, if you wanted to create a segment that included data for your Spring campaign in London and Paris, you would configure the segment as follows:

Campaign exactly matches Spring

City matches regex London|Paris

You can also use the pipe within parentheses. For example, here's another way to create the expression to match your beach-sandal SKUs:

(MN|WM)BS\d+

(MN OR WM)

BS (character code for beach sandals)

\d (numeric character)

+ (one or more times)

Square brackets [ ] and the hyphen -

Square brackets [ ]

Use square brackets to create a set of characters to match.

When you include a series of characters in brackets, your expression matches 1 of those characters.

For example, the expression PART[123] matches:

PART1

PART2

PART3

The expression doesn't match:

PART12

PART23

PART123

Hyphen -

Use the hyphen along with the brackets to create a range of characters to match.

For example, if you need to match any uppercase letter, you can specify [A-Z].

If you need to match any digit, you can specify [0-9].

If you needed to match all of the part numbers above, you could use the following expression:

[A-Z]+[0-9]+

[A-Z] (matches any uppercase letter)

+ (one or more times)

[0-9] (matches any digit)

+ (one or more times)

Regular expressions (RegEx)

Question mark (?), plus sign (+), asterisk (*)

Question mark (?)

The question mark (?) matches the preceding character 0 or 1 times.

For example, 10? matches:

1

10

Example

Match an IP address with 1 or 2 digits in the last section.

For example, 216\.239\.32\.\d\d? matches:

216.239.32.2

216.239.32.34

This example uses the backslash to escape the decimal, and uses \d to match any digit.

Plus sign (+)

The plus sign (+) matches the preceding character 1 or more times.

For example, 10+ matches:

10

100

1000

etc.

Example

Match an IP address with 1 or more digits in the last section.

For example, 216\.239\.32\.\d+ matches:

216.239.32.2

216.239.32.34

216.239.32.567

This example uses the backslash to escape the decimal, and uses \d to match any digit.

Asterisk (*)

The asterisk or star matches the preceding character 0 or more times.

For example, 10* matches:

1

10

100

1000

etc.

Example

Match an IP address with 1 or more digits in the last section.

For example, 216\.239\.32\.\d* matches

216.239.32.2

216.239.32.34

216.239.32.567

This example uses the backslash to escape the decimal, and uses \d to match any digit.

If you need to match more than just the preceding item, you can combine the asterisk with the dot (.*). The dot matches any preceding item, and then the asterisk will match that item zero or more times, which lets you match things like all URIs that begin and end with the same characters, regardless of how many characters are in between. For example, /mens/.*html matches:

/mens/shirts/oxford.html

/mens/shirts/oxford/shortsleeve.html

Dot (.) and backslash (\)

Some characters have one meaning in regular expressions and completely different meanings in other contexts. For example, in regular expressions, the dot (.) is a special character used to match any one character. In written language, the period (.) is used to indicate the end of a sentence. In mathematics, the decimal point (.) is used to separate the whole part of a number from the fractional part.

Regular expressions first evaluates a special character in the context of regular expressions: if it sees a dot, then it knows to match any one character.

For example, the regular expression 1. matches:

11

1A

The regular expression 1.1 matches

111

1A1

If you were to provide an IP address as a regular expression, you would get unpredictable results. For example, the regular expression 0.0.0.0 matches:

0102030

0a0b0c0

In order to tell regular expressions to see the dot in its original context as a separator for the different parts of the IP address and not as a special character used to match any other character, you need to provide a signal to that effect. The backslash (\) is that signal. When regular expressions

sees a backslash, it knows that it should interpret the next character literally. A regular expression to match the IP address 0.0.0.0 would be:

0\.0\.0\.0

Use the backslash to escape any special character and interpret it literally; for example:

\\ (escapes the backslash)

\[(escapes the bracket)

\{ (escapes the curly brace)

\. (escapes the dot)

Caret (^)

Use the caret to match the following adjacent characters at the beginning of a string.

For example, ^St matches:

Start here

Stand here

Stop here

However, ^St doesn't match:

1 Start here

2 Stand here

3 Stop here

The preceding 3 lines start with a digit and a space, not with the letters "St".

Use this type of regular expression to create segments, filters, or goal steps that match a URI. For example, if you need to isolate data for a particular directory of pages, you can use expressions like:

^/mens/ (matches www.example.com**/mens/**)

^/womens/ (matches www.example.com**/womens/**)

Dollar sign ($)

Use the dollar sign to match the preceding, adjacent characters at the end of a string.

For example, end$ matches:

Temporarily susp**end**

We're going off the deep **end**

Match the characters that prep**end**

However, end$ doesn't match:

Temporarily suspend.

We're going off the deep end.

Match the characters that prepend.

The preceding 3 lines all end with **nd.** rather than the **end** characters you're matching with the regular expression.

Use this type of regular expression to create segments, filters, or goal steps that match a URI. For example, if you're you're testing the efficacy of a new .htm version page vs. an older .html version, you can identify the versions separately with regular expressions like:

email-signup\.htm$

email-signup\.html$

Use the slash to escape the dot and ensure that it is interpreted literally.

Test your regular expressions

An easy way to test a regular expression on your own data is to navigate directly to a report in your view and use the advanced table-filter feature.

For example, to try out a regular expression that filters pages:

1. Sign in to Google Analytics.
2. Navigate to your view.
3. Open **Reports**.
4. Select **Behavior > Site Content > All Pages**.
5. Above the table, click **advanced**.
6. Create a condition:
   [select: Include or Exclude] Page [select: Matching RegExp] [enter regular expression]
7. ***Click*** Apply.