# Regex Email Address Tutorial.md

Regex Email Address Tutorial.md

Summary

Regex, short for regular expressions, define particular search patterns using a combination of characters and can be used for data validation, to match a string of text, and to find and replace operations.

A popular regex example is matching an email address. Take a look at the snippet below:

`/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,5})$/`

We will discuss in detail how you can use the regex email address within a string.

TABLE OF CONTENTS

REGEX COMPONENTS

An email address format is local-part@domain. Please see below for an example:

Email address: eight19@joy.com

eight19 is the local-part, @ = @, and joy.com is the domain.

To dive a little deeper into regex email addresses, let's take a look at a few ways that an email address can be validated via regex.

The following is an explanation for email regex:

`@` # must contain a @ symbol

`[^-]` # domain cannot start with a hyphen (-)

`[a-zA-Z0-9-]+` # Start with chars in the bracket [ ], one or more (+)

`(\\.[a-zA-Z0-9-]+)*` # follow by a dot (.), optional

`(\\.[A-Za-z]{2,})` # the last tld, chars in the bracket [ ], min 2

`(?=.{1,64}@)` # local-part min 1 max 64

`[a-zA-Z0-9_-]+` # Start with chars in the bracket [ ], one or more (+)

```
                          # dot (.) not in the bracket[] cannot start with a dot (.)
```

`(\\.[A-Za-z0-9_-]+)*` # follow by a do t (.), then chars in the bracket [ ] one or more (+)

```
                          # means this is optional
                          # this rule for two dots (.)
```

1. Simple-Email Regex - Checks to ensure an email contains a non whitespace character, an @ symbol, and at least one character. Example: `^(.+)@(\S+)$`
2. Strict-Email Regex - Must meet specific email requirements: Example: `^($=.{1,89}@)[A-Za-z0-9_-]+(\\.[A-Za-z0-9_-]+)*@[^-][A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)*(\\.[A-Za-z]{4,})$`

Example of local-part A. 64 characters is the Max B. Allow dot(.), underscore(_) and hyphen (-) C. dot (.) is not the first or last character and does not appear consecutively.
(Example: tsmile.smile@joy.com isn't valid) D. Uppercase and lowercase latin letters a-z and A-Z.

Example of domain A. tld min 2 characters B. Digits 0 to 9 C. Uppercase and lowercase Latin letters A to Z and a to z D. Hyphen (-) is not the first or last character E. dot (.) is not the first or last character and doesn't appear consecutively

To support special characters such as `+=!$%|` in local-part, just add the brackets []: Example: `[a-zA-Z0-9_-+=!$%|]`

3. Non-Latin or Unicode-Email Regex Example: Non-Latin and Unicode are character sets that are used to define all character and glyphs using different character sets for different languages. Example: `"二ノ宮@黒川.日本"`

4. APACHE COMMONS VALIDATION v1.7

The Apachce Commons Validation addressed issues such as speed development and the maintenance of validation rules. This validator contains two distinct sets of functionality: A. Reusable "primitive" validation methods B. A configurable validation engine (typically XML)

ANCHORS

Anchors don't match any character at all. They, instead, match a position before, after, or between characters. At a certain position, they can be used to “anchor” the regex match. A regex that contains an anchor by itself can only find zero-length matches. They ensure that the current positioning in the string matches a certain position: the beginning, the end, or in the case of `\G,` the position immediately following the last match. Example: `^$`

QUANTIFIERS

All of the following quantifiers are (Greedy Matches) by default, which means that they match as many characters as possible.

{n}: match n times {n,}: match n or more times {n,m}: match at least n times, at most m times ?: match never or once *: match zero or more times +: match one or more times

Example: `> /".*"/.exec('"abc"def"')[0]  // greedy'"abc"def"'`

To make them (Lazy Matches) where they match as few characters as possible, insert question marks (?) after them. Example:

Example: `> /".*?"/.exec('"abc"def"')[0] // Lazy '"abc"'`

OR OPERATORS

Used to match characters in text strings, such as to define patterns.

CHARACTER CLASSES

Allows the regex engine to match only one out of several characters by placing the characters you want to match between square brackets. If you want to match an a or an e, use `[ae]`

Example:

`[0-9]` = Receives any digit between `0` and `9`

FLAGS

A flag is an optional parameter to regex that modifies its behavior of searching. Example:

Literal flag Property name ES Description d hasIndices ES2022 Switch on match indices g global ES3 Match multiple times i ignoreCase ES3 Match case-insensitively m multiline ES3 ^ and $ match per line s dotAll ES2018 Dot matches line terminators u unicode ES6 Unicode mode (recommended) y sticky ES6 No characters between matches

GROUPING AND CAPTURING

Groups and Capturing use the ( ) symbols and are useful for creating blocks of patterns to apply repetitions or other modifiers to them as a whole. Example: `([a-p]{2}[0-9])+` The `+` metacharacter is applied to the entire group.

BRACKET EXPRESSIONS

Multiple character classes can be combined in a single bracket expression, where the regex will match any letter or number. Example:

`[A-Z0-9]`

BOUNDARIES

Boundaries make assertions about what can be matched to the left and right of the current position of an anchor.

Different types of Boundaries:

Word Boundary: The word boundary `\b` matches positions where one side is a word character; a letter, digit or underscore

Not-a-word-boundary: `\B` It matches when neither side is a word character at any position in the string `$=(@-&++)` ,including the beginning and end of a string. Ir also matches when both sides are a word character, for instance between the "T" and the "o" in "To!"

Left-and Right-of-Word Boundaries: `[[:<:]]`

BACK-REFERENCES

These are regular expression commands that refer to a previous part of the matched regular expression and are specified with backslash and a single digit. The subexpression, a part of the regular expression, is designated with parentheses. Example:

```
 ' \1 '
```

LOOK AHEAD AND LOOK BEHIND

Lookahead = Can be positive or negative and allows to add a condition for “what follows”. It commands to look for A but match only when it is followed by Z. Any pattern can be used. Example: `A(?=Z)`

Lookbehind = Can be positive or negative and looks behind; it allows adding a pattern or condition, only if there's something before it. Example: `(?<=Y)X`

Author

Tiffany Jones is the author and can be reached via GitHub