RegEx:

* Decimal (numbers, int, digit), string, symbol, parenthesis, slash/balslach
* **The syntax is the same in all languages**
* In **.htaccess** regEx is used.
* [regex101.com](https://regex101.com/)
* [regular-expressions.info/refcharclass.html](https://www.regular-expressions.info/refcharclass.html)

## Syntax

* **?** // The char before it, is optional. The question mark makes the preceding token in the regular expression optional. colou?r matches both colour and color. The question mark is called **a quantifier**.
* **+** // More than one
* **\d** // one decimal
* **\w** // (a-z) (A-Z) **one char**
* **\s** // matches **one space**
* **\S** // everything **but space**
* **\b** // In regular expressions, \b is a special character that represents a word boundary. It matches a position between a word character (\w) and a non-word character (\W), or between a word character and the start or end of the string.

#python example:

import re

pattern = r'\bcat\b'

text = 'I have a cat and a caterpillar.'

matches = re.findall(pattern, text)

print(matches) # Output: ['cat']

* **[]** // one of. **character class**. [abc] matches a, b, **or** c. The [] construct in a regex is essentially shorthand for an | on all of the contents. The regex [a-z] will match any letter a through z.
* **()** // **capturing groups**, a-z0-9 -- Can be captured by (a-z0-9) (doesn't mean a range) and then can be referenced in a replacement and/or later in the expression. in Python matches the whole expression and gives a list of tuples of every group separately. /(ab|cd)/ matches 'ab' or 'cd'

#python example

import re

pattern = r'(\w+)\.(\w+)'

text = 'example.com.is.a.website'

matches = re.findall(pattern, text)

result = [match for match in matches]

print(result) # Output: [('example', 'com'), ('is', 'a')]

* + When we use capturing groups findall returns a list of tuples, anything outside the () will not be part of returned values. What is returned is the expression matched inside (). dot for example in the example above is not inside () but it should be matched.

import re

pattern = r'java(d?)'

text1 = 'java'

text2 = 'javad'

text3 = 'javascript'

matches1 = re.findall(pattern, text1)

print(matches1) # Output: ['']. Since the word java matches, and d is optional, but we don't have the letter d, it returns an empty string as a match.

matches2 = re.findall(pattern, text2)

print(matches2) # Output: ['d']

matches3 = re.findall(pattern, text3)

print(matches3) # Output: ['']

* **+** // one or more
* **.** // **Everything but \n** (enter)
* **.+** // Matches **a paragraph**
* **.\*** // Matches **everything as one**
* **|** // **or**, for logic like business email
* **[:chars:]** // picks just chars from text
* **{1,3}** // specify the exact repetition of a pattern
* **?=, ?<=** // assertions, look ahead and look behind if something happened first then check the rest. Example: (?=/).\*

| **Pattern** | **Name** |  |
| --- | --- | --- |
| (?=«pattern») | Positive lookahead | ES3 |
| (?!«pattern») | Negative lookahead | ES3 |
| (?<=«pattern») | Positive lookbehind | ES2018 |
| (?<!«pattern») | Negative lookbehind | ES2018 |

* + extract quoted words:

'how "are" "you" doing'.match(/(?<=")[a-z]+(?=")/g)# [ 'are', 'you' ]

## Flags

* / expr / **flags** // Example: /.\*/gim : g i m multi-line
* **i**n case sensitive
* **g**lobal // just return the first match, default
* **LOCALE / EXTENDED**
  + In regular expressions, the re.LOCALE or re.L flag is used to make certain pattern-matching operations dependent on the current locale. It affects the interpretation of special character classes such as \w, \W, \b, \B, and makes them dependent on the current locale settings.
  + There.EXTENDED or re.X flag, also known as the verbose flag, allows you to write regular expressions in a more readable and organized manner. It ignores whitespace characters and allows you to include comments within the pattern

#python example:

import re

# Using the LOCALE flag

pattern = r'\b(\w+)\b'

text = 'Hello, 世界!'

matches = re.findall(pattern, text, flags=re.L)

print(matches) # Output: ['Hello', '世界']

# Using the EXTENDED flag

pattern = r'''

\b # Word boundary

(\w+) # One or more word characters

\b # Word boundary

'''

text = 'Hello, World!'

matches = re.findall(pattern, text, flags=re.X)

print(matches) # Output: ['Hello', 'World']

* UNICODE / ASCII
* **VERBOSE/X** // Special chars usage to help with readability, for example, we can use comments in our regex with #

## Examples

[:RED:] // These letters R E D

\w+@(gmail|yahoo)\.\w+ // email, (ir|com|biz|xyz): check business emails

(\w+)\.(\w+) // 'example.com is a website' returns a list of tuples that are separated by one dot(period): [(name, family) , (name1, family1)]

java(d?) // in python matches only '' and d

(java)(d?) // returns java, '', d

/foo/boo/poo

/Sousan/Derry/Rose/Danny

(?=/Sousan).\* // we should have /Sousan first

([12]\d{3})[-:/.](0?\d|1[0-2])[-:/.]([0-2]?\d|3[01])\b // match 2009/11/12