Regular Expressions From Beginner To Expert By Varun

Date: November 12, 2018

In this document want to show how to master regex. I wasn't good at regex until I realized its power and the necessity to understand and master it.

---- Avoid writing lines and lines of code by using power of regex.

---- The regex shown here will take you from a simple example to the complex patterns.

---- Lot of tools apart from all the programming languages support regex.

---- I personally know integration tools such as **Ca-Api-Gateway**, **Apigee**, **TibcoFlogo**, **Mulesoft** where regex can be used and match complex patterns, save unnecessary code for pattern match.

---- Logging slutions like **ELK**, **Splunk** support full on regex.

---- Analytical tools such as **Tableau** support full on regex.

---- Please follow ths document and practice along by the end of it. You will start writing complex regex on your own.

Contents

1.1 Introduction

1.1.1 What are regular expressions

- Regular expressions are also called as Regex or RegexP

- Sequence of characters that defines a search pattern

1.1.2 Why to learn regular expressions

Used in all programming languages. Also used in caApiGateway and Apigee

- Can be used to validate a form

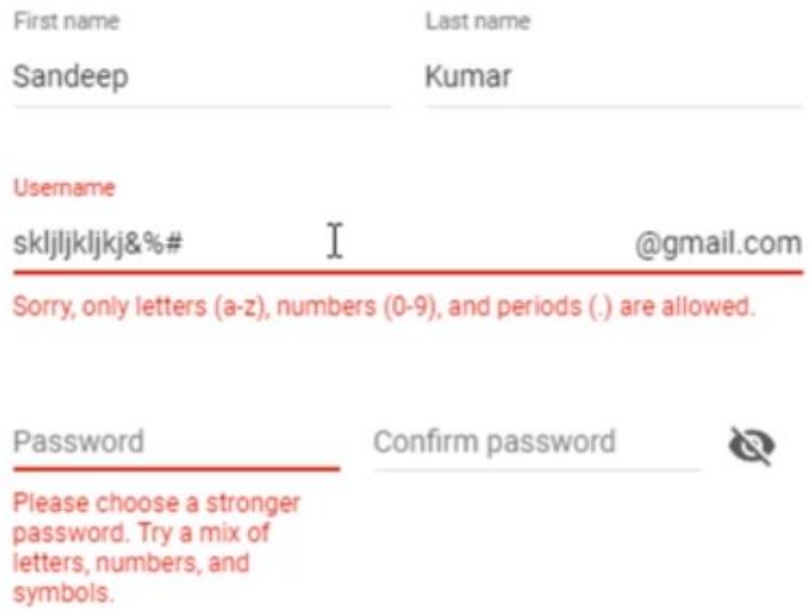
- To extract a part of string

- To clean up a file

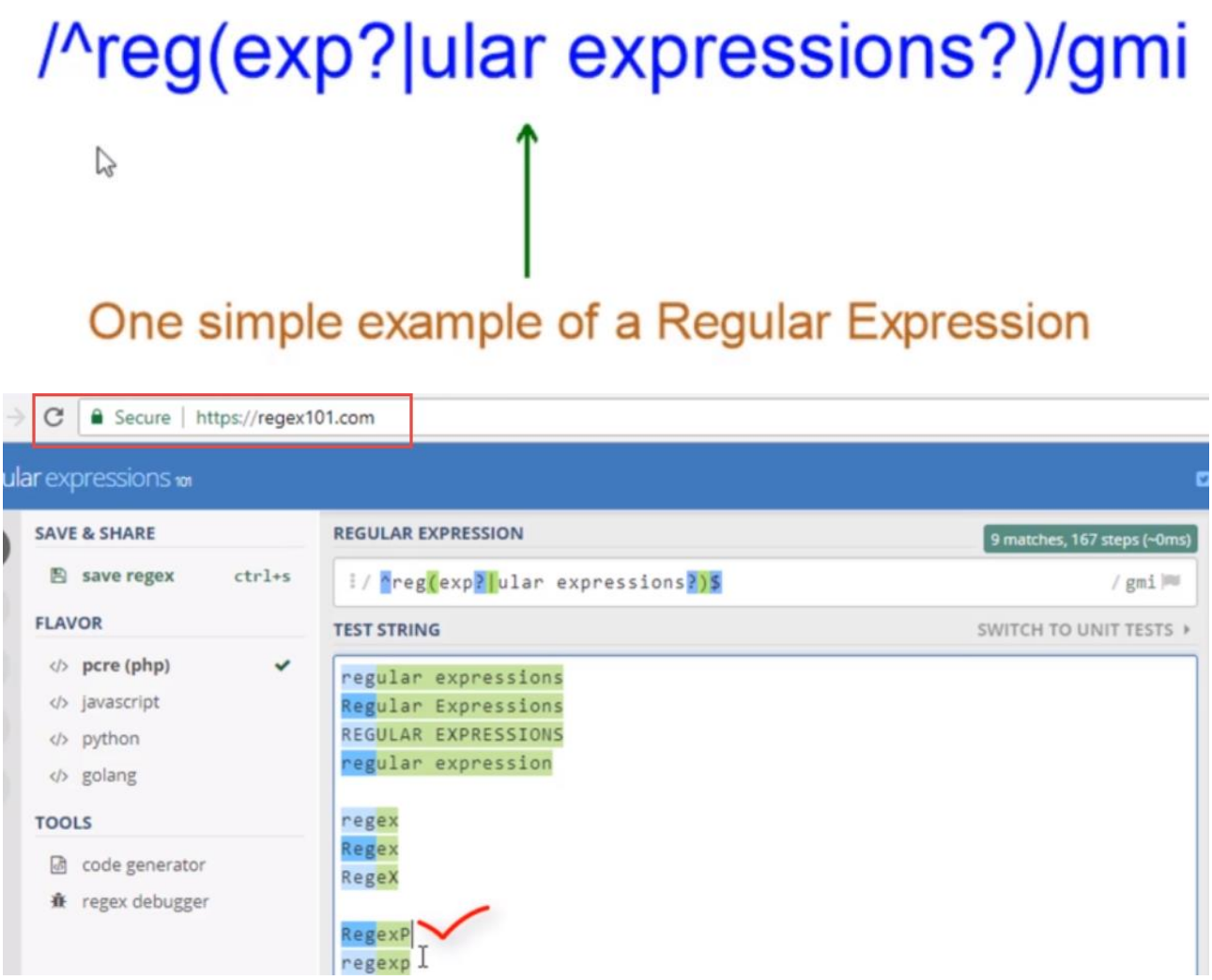
- used in database or command line as well

- used for other multiple purposes and list is on and on.

For example validating user input into a form



1.1.3 How regex is created



1.1.4 Tools Used

Tools used for learning the regex during the course of this document.

Online regex tester

https://regex101.com

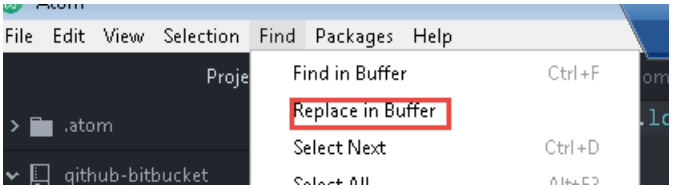
https://regexr.com

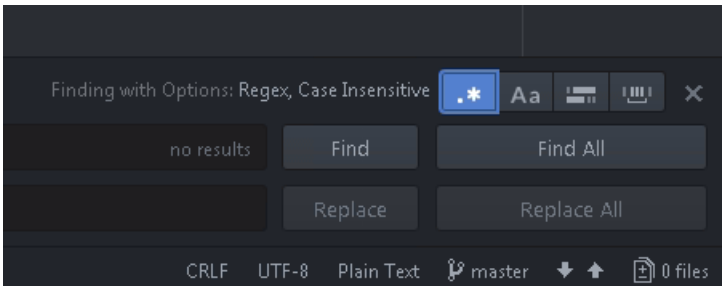
https://www/regextester.com

Offline Regex Testers

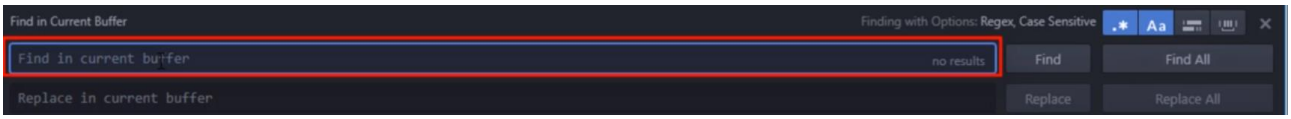
https://atom.io

reach regex by using ctrl+F or below steps



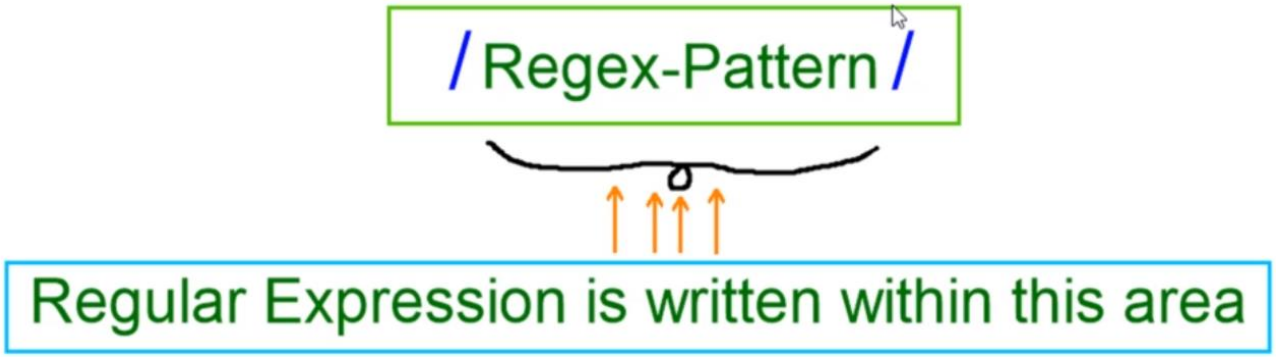


Regexp are case sensitive.



1.1.5 Basic Syntax Regex

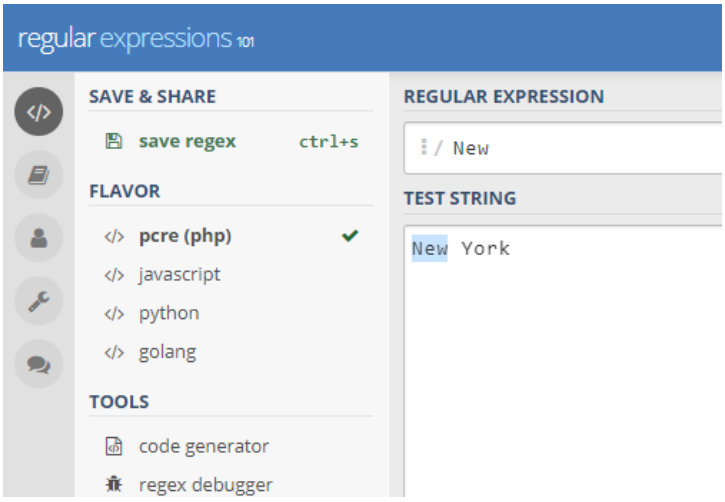
Used in all programming languages. Also used in Api Management tools such as, caApiGateway and Apigee



1.2 All About Characters

1.2.1 Character Literals

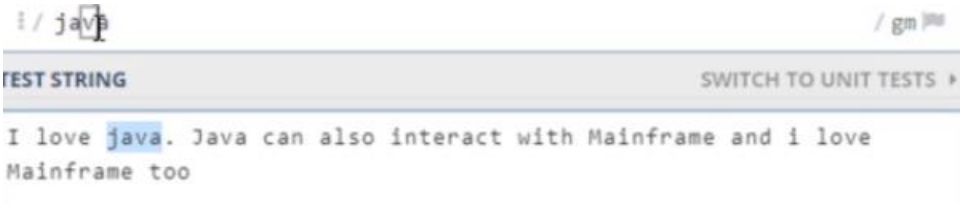
Takes the string as it is.



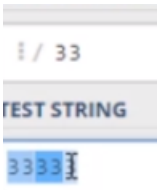
Regexby default is case sensitive.

:/Java/gm (Case sensitive)

:/Java/gmi (Case insensitive)

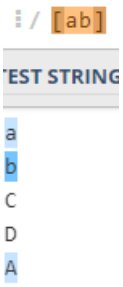


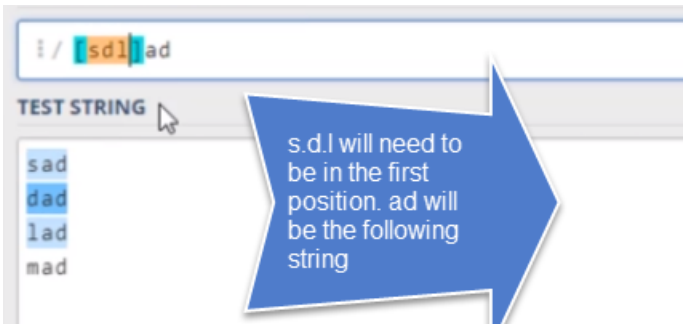
Regex starts searching from left to right

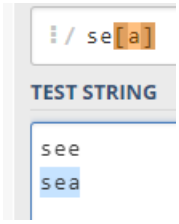


1.2.2 Character Classes

Character class are enclosed in [] brackets

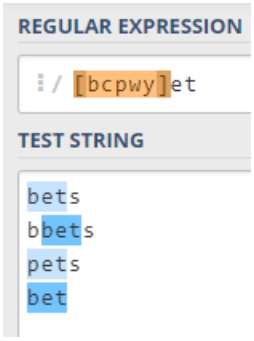




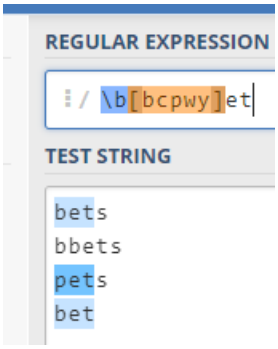


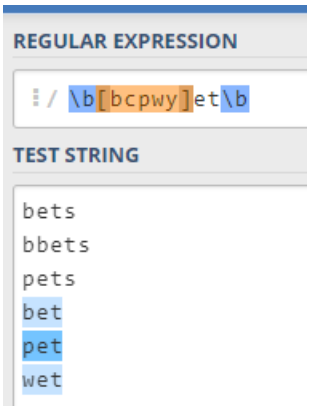
1.2.3 Forward Boundary

>> Forward boundary uses the following pattern '\b'

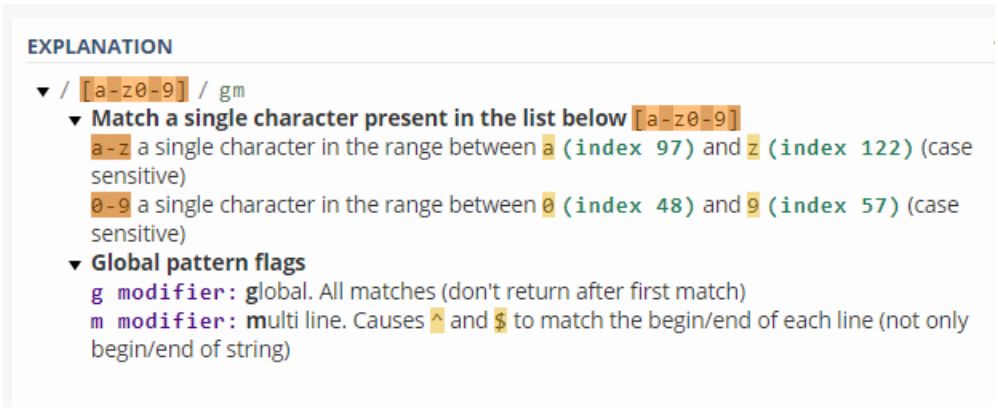


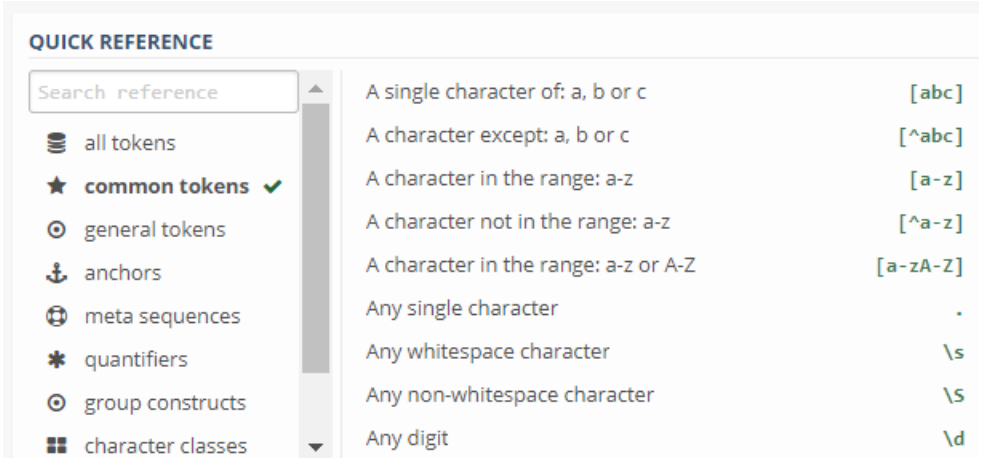
say in this you want to match only the word bet and not sub string that may contain bet.

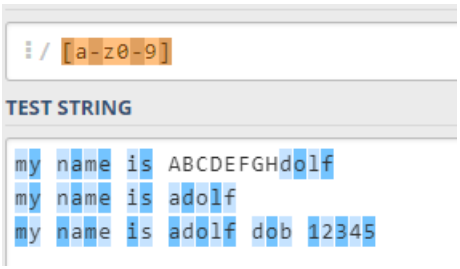




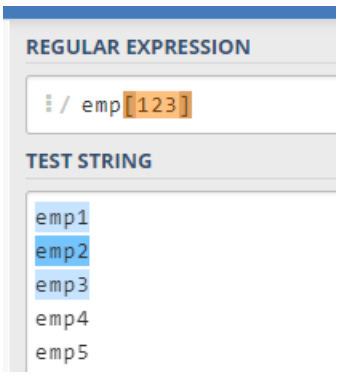
1.2.4 Character Range



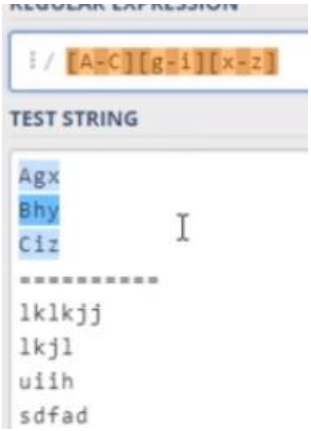




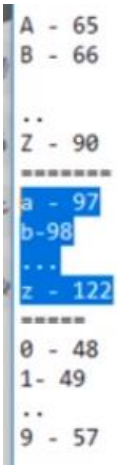
Say there are 5 employes and you want to match only the emp 1,2,3



Write a regex to match the first 3 words.



Indexes of alphabets and numbers



You needs to match the index when defining the character ranges you need to go from small to big

[A-z] This is correct

[a-Z} This is wrong.

1.2.5 **Regex Live Example**

```
<!DOCTYPE html>
<html>
<head>
<title>Testing Regular Expressions in HTML forms</title>
</head>
<body>
<h1>Test your Regular Expressions here</h1>
<fieldset>
<legend>Write your Regular expressions</legend>
<form>
<div>
<label for="display-name"> Prove that you are not a robot:
<span class="warning">*(Enter a single English capital letter.)</span>
</label>
<input type="text" id="display-name" name="ip-display"
pattern="[A-Z]"
title="Enter any single English capital Letter only"/>
</div>
<div>
<input type="submit" class="submit" value="Submit" />
</div>
</form>
</fieldset>
</body>
</html>
```

```html
<!DOCTYPE html>
<html>

<head>
    <title>Testing Regular Expressions in HTML forms</title>
</head>

<body>
    <h1>Test your Regular Expressions here</h1>

    <fieldset>
        <legend>Write your Regular expressions</legend>
            <form>
                <div>
                    <label for="display-name"> Prove that you are not a robot:
                        <span class="warning">*(Enter a single English capital letter.)</span>
                    </label>
                    <input type="text" id="display-name" name="ip-display"
                           pattern="[A-Z]"
                           title="Enter any single English capital Letter only"/>
                </div>
                <div>
                    <input type="submit"
                </div>

            </form>
    </fieldset>

</body>

</html>
```

This 'pattern' attribute in HTML uses Regular Expression

```html
<!DOCTYPE html>
<html>

<head>
    <title>Testing Regular Expressions in HTML forms</title
</head>

<body>
    <h1>Test your Regular Expressions here</h1>

    <fieldset>
        <legend>Write your Regular expressions</legend>
            <form>
                <div>
                    <label for="display-name"> Prove that
                        <span class="warning">*(Enter
                    </label>
                    <input type="text" id="displ
                           pattern="/[A-Z]/"
                           title="Enter any single English
                </div>
                <div>
                    <input type="submit" class="submit" valu
                </div>

            </form>
    </fieldset>

</body>

</html>
```

You do not need to put forward slash at start and end when you use pattern attribute.

Test your Regular Expressions here

Write your Regular expressions

Prove that you are not a robot: *(Enter a single English capital letter.) a

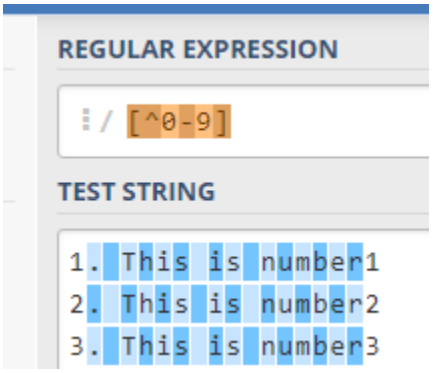
Submit

Please match the requested format.
Enter any single English capital Letter only

1.2.6 Negation Characters

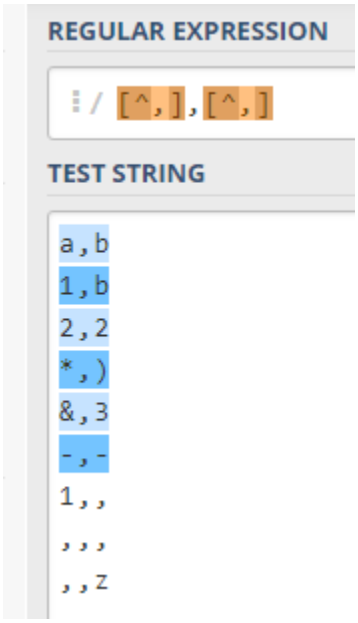
Are represented with a ^ symbol inside [] brackets.

[^a-e] will match all characters except a-e



Example 1:-

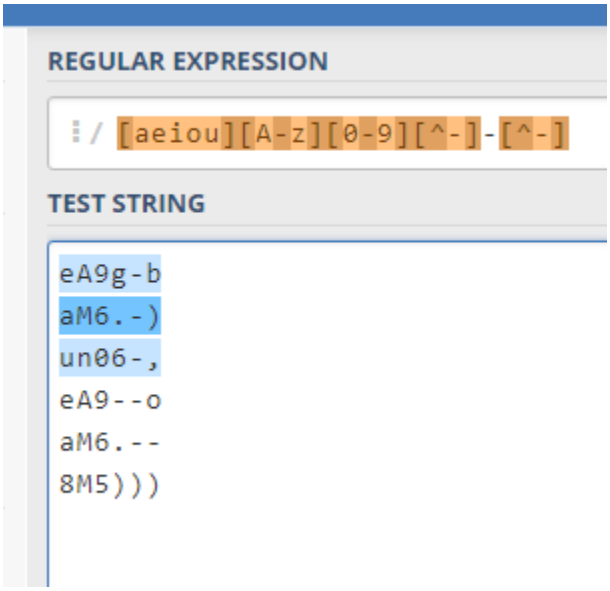
- The First Character should be any character except a comma(,)
- The Second Character must be a comma(,)
- The Third Character should be any character except a comma(,)



Example 2:-

- The First Character must be a vowel(a,e,i,o,u)
- The Second Character must be a English alphabet(it can be lowercase or uppercase English alphabets)

- The Third Character must a single number
- The Fourth Character should be any character except a Hyphen(-)
- The Fifth Character must be a Hyphen(-)
- The Sixth Character should be any character except a Hyphen(-)

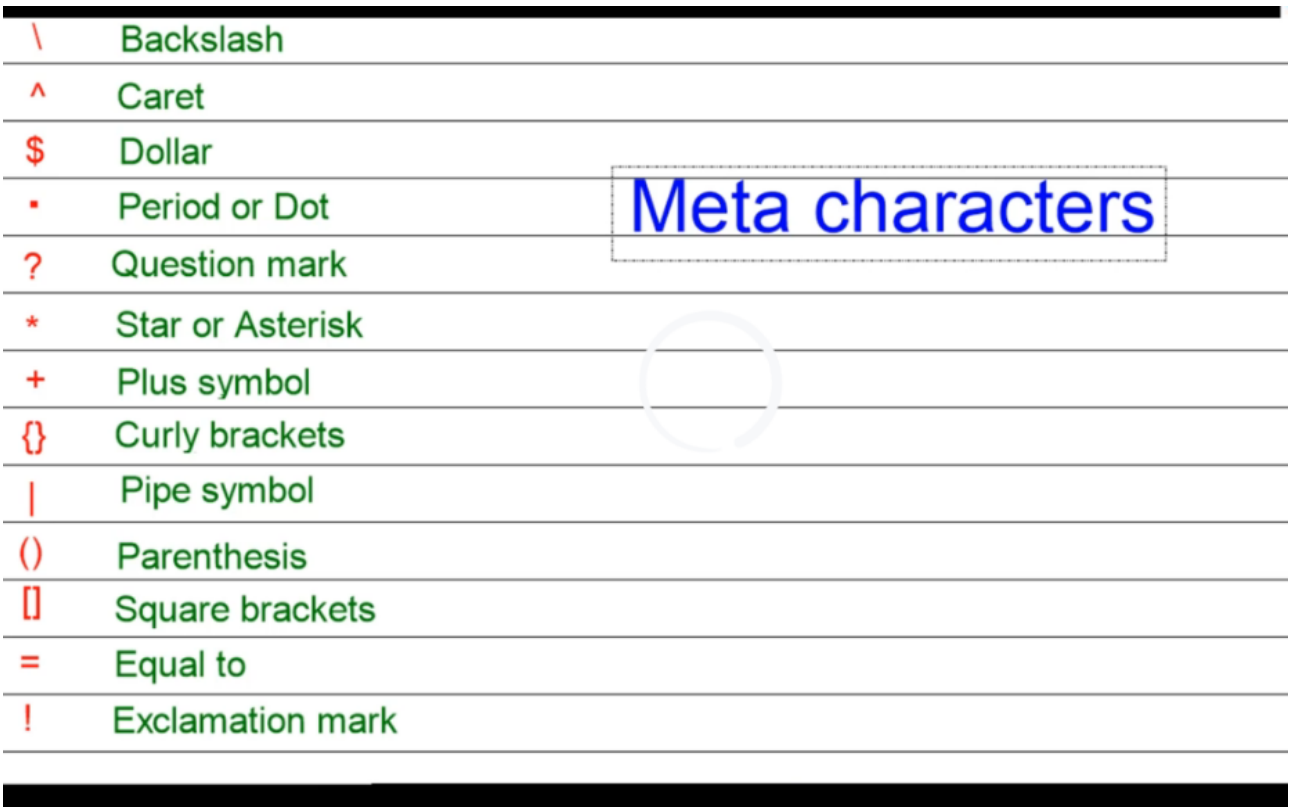


1.3 Meta Characters

1.3.1 What are Meta Characters

In regex each character is described as normal character or meta character.

Special character that have special meaning.



\ used to preceed a meta char or predefine character class.

^ negation character and used in anchors

$ Is used for matching a boundary at the end of the line

. used to match any character aprart from new line

? quantifiers

* quantifiers

+ quantifiers

{} quantifiers

| Alternation

() Grouping

[] Ranges

= Assertions (look ahead, look behind)

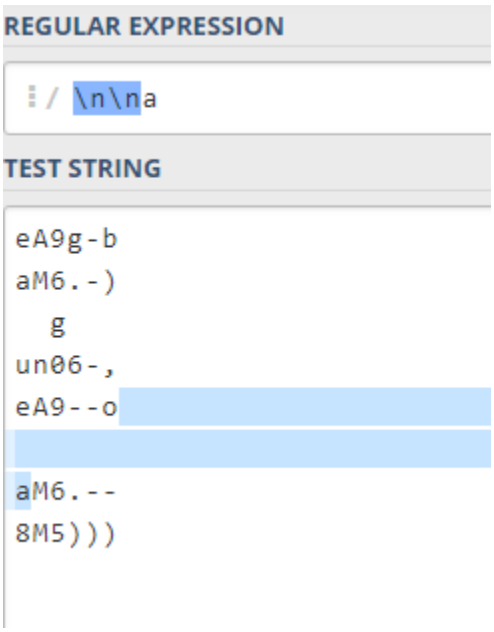
! Assertions (look ahead, look behind)

1.3.2 Wild Card Meta Characters Part 1

. meta character is also called wild card metacharacter

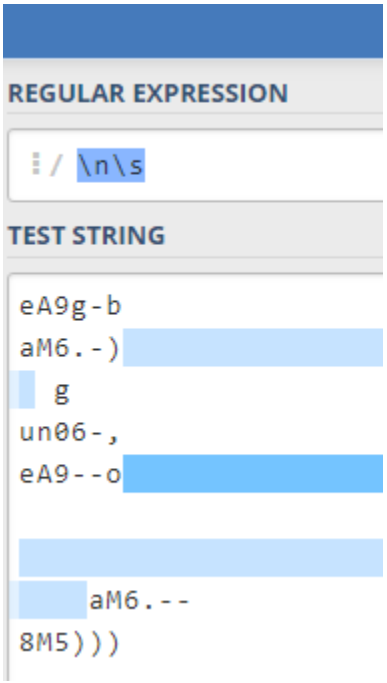
To match only one character

New line char is represented by \n. Line break.

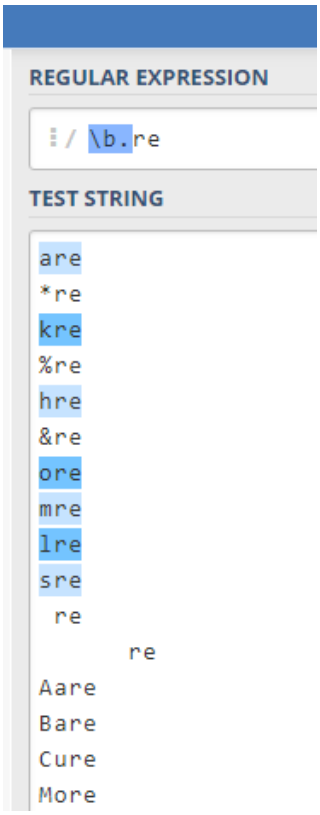


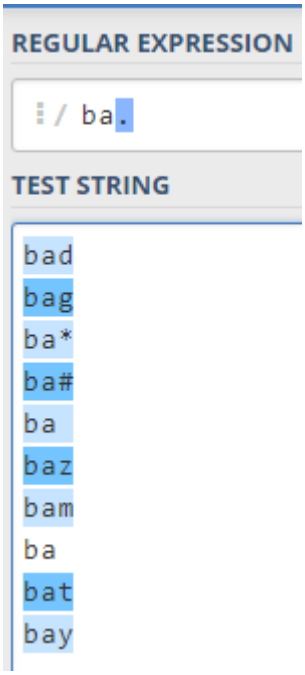
\s whitespace

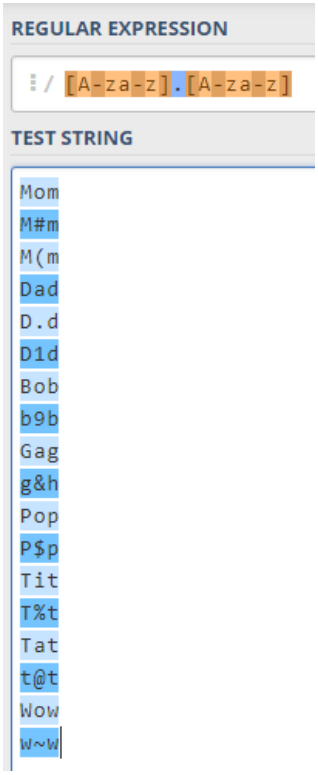
Matches new line followed by whitespace

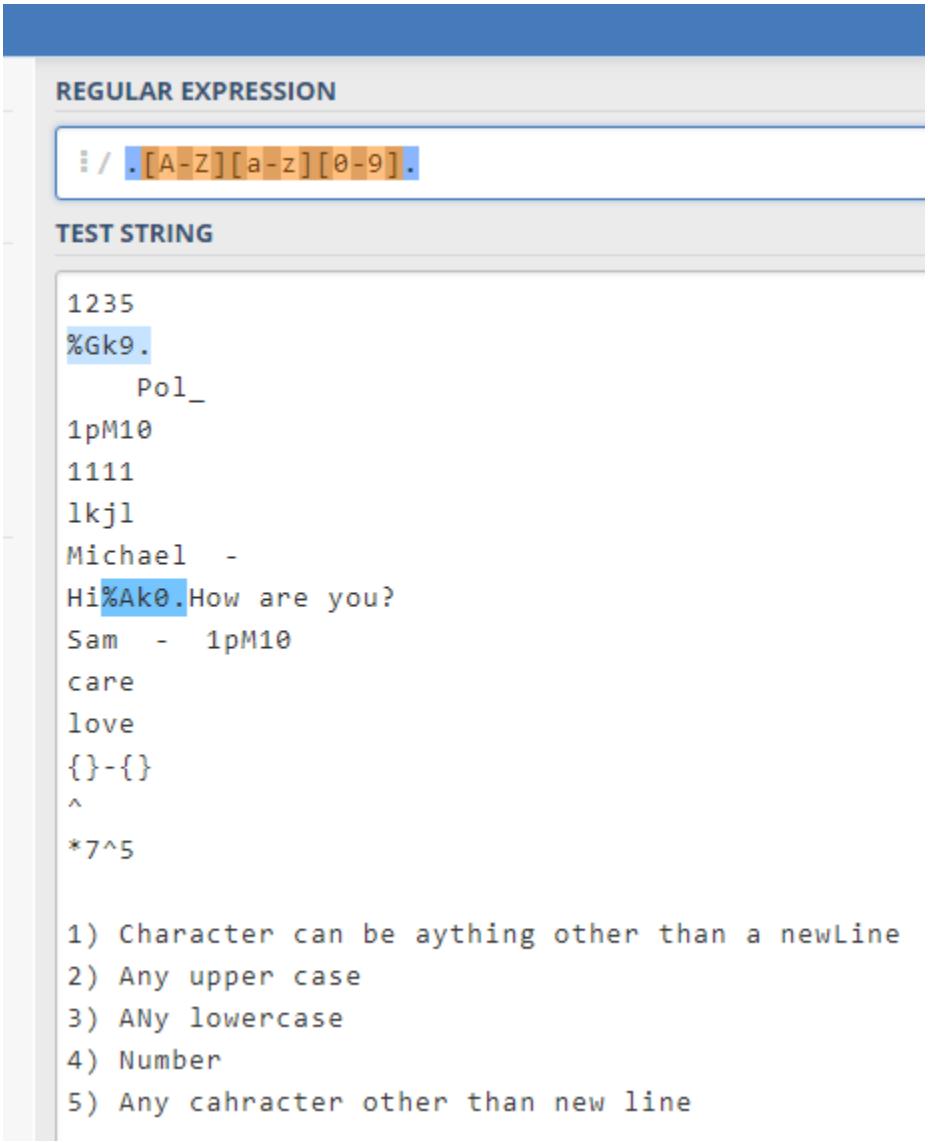


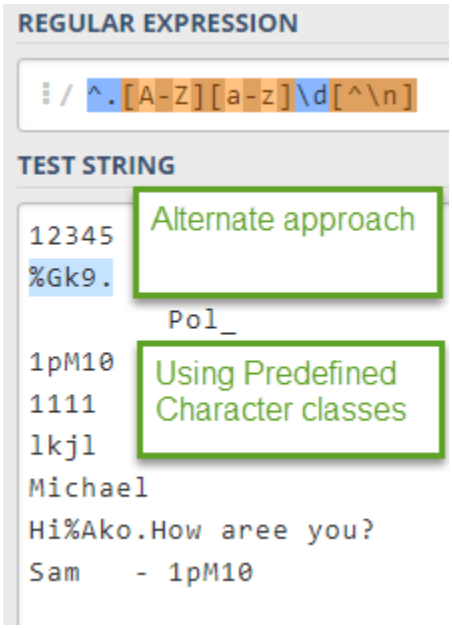
\b is a boundary between a word and a non word character





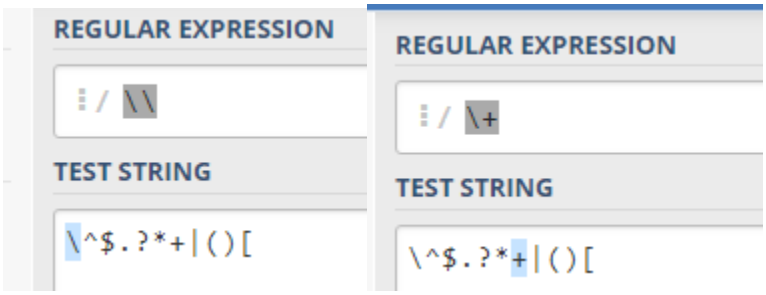



REGULAR EXPRESSION
/ .[A-Z][a-z][0-9].
TEST STRING
1235
%Gk9.
Pol_
1pM10
1111
lkjl
Michael -
Hi%Ak0.How are you?
Sam - 1pM10
care
love
{}-{}
^
*7^5
1) Character can be aything other than a newLine
2) Any upper case
3) ANy lowercase
4) Number
5) Any cahracter other than new line

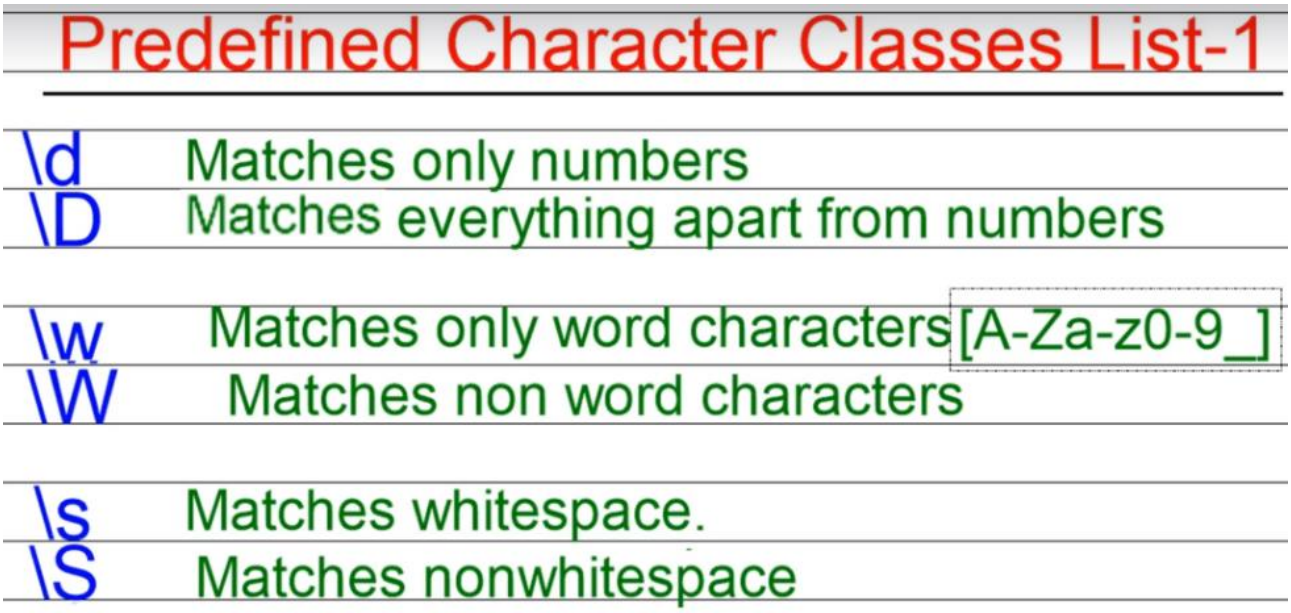


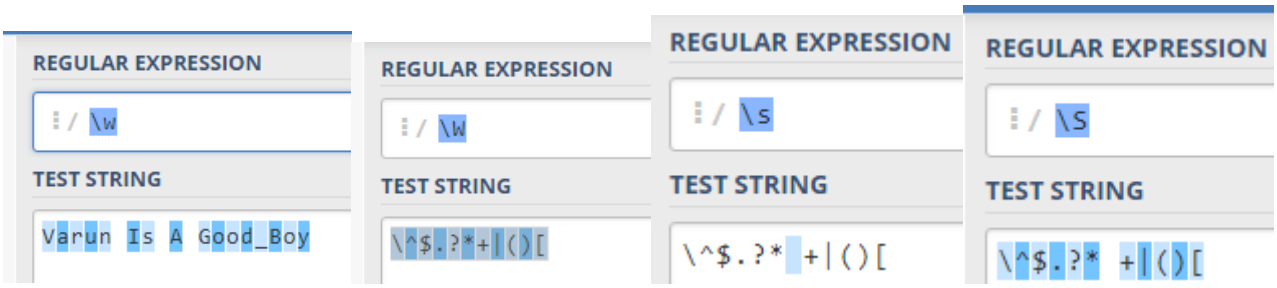
1.3.3 Escaping Meta Characters

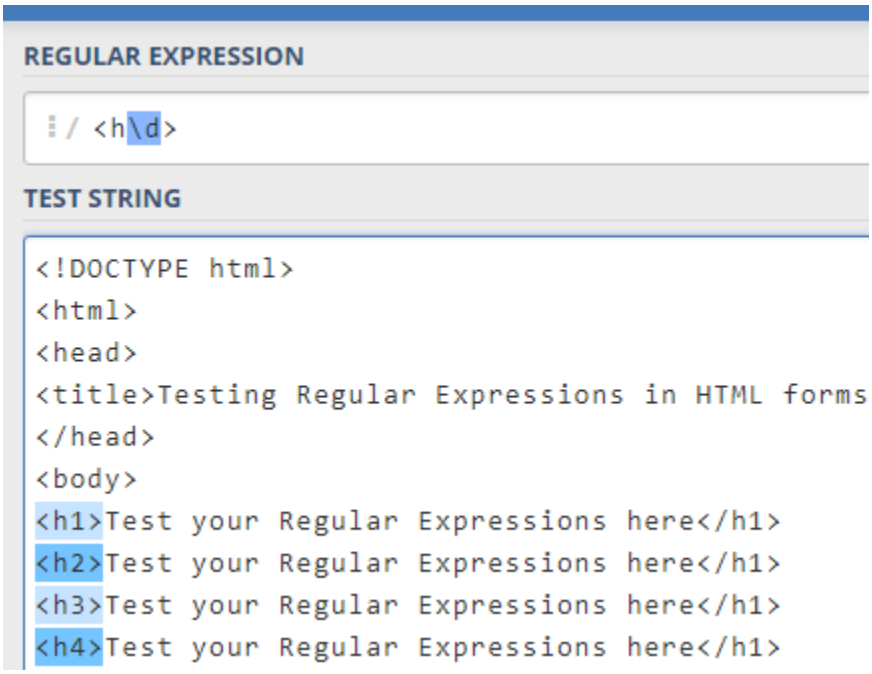
Meta characters are escaped by \



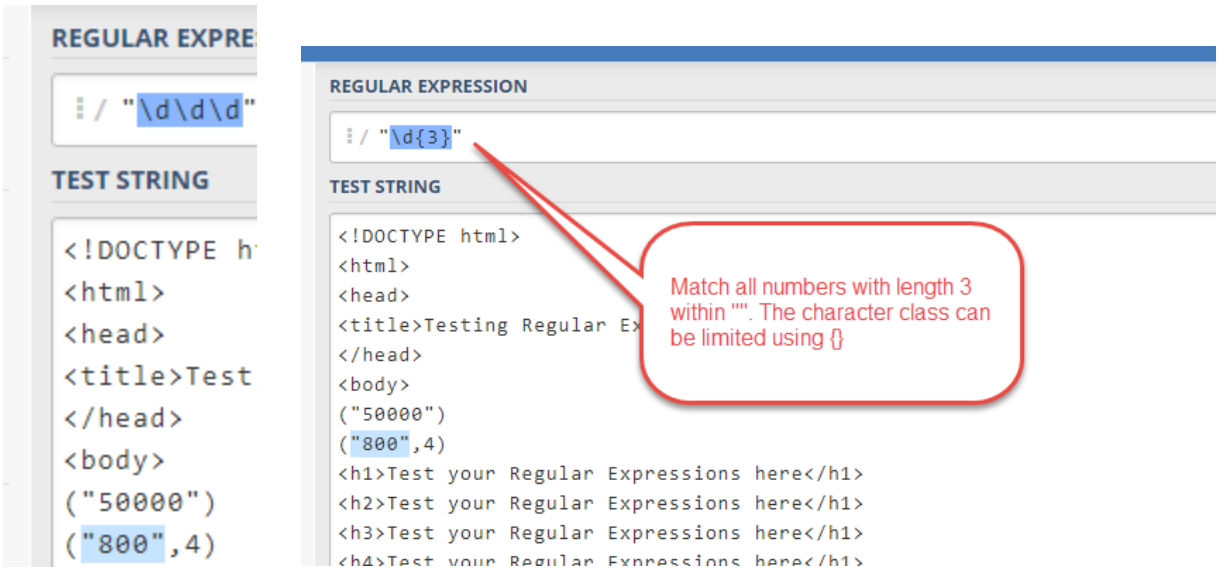
1.3.4 Predifined Character Classes







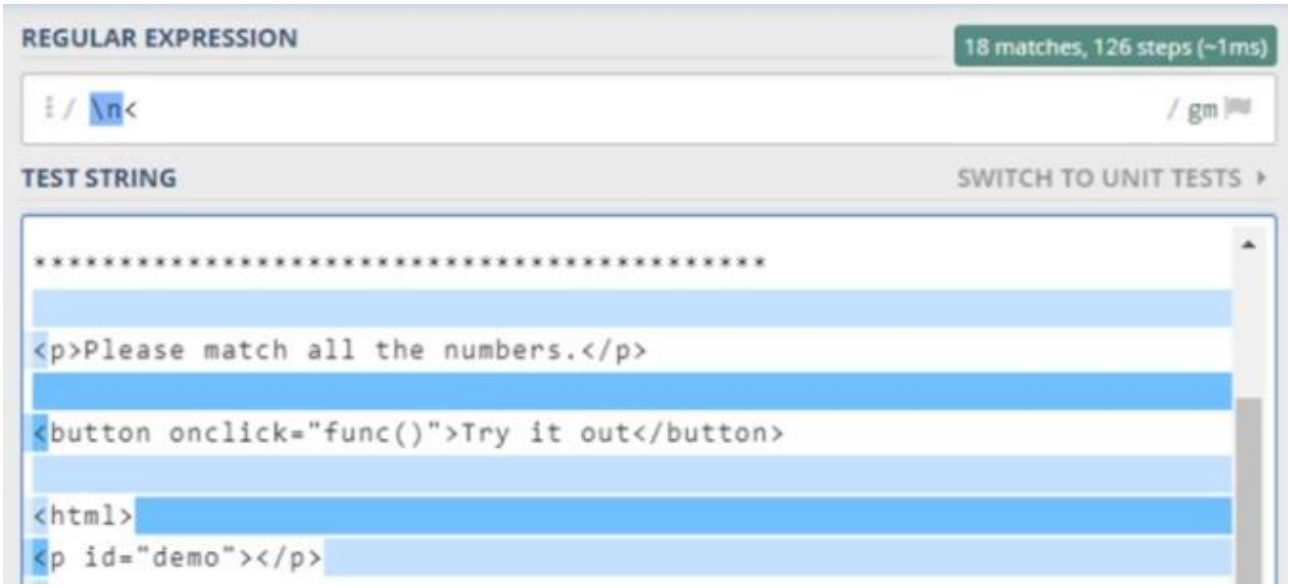
To capture a 3 digit number within “”



Predefined Character Classes List-2

\n	Line break or New line character
\t	Tab character

non- printable characters
\r is carriage return
\f is form feed

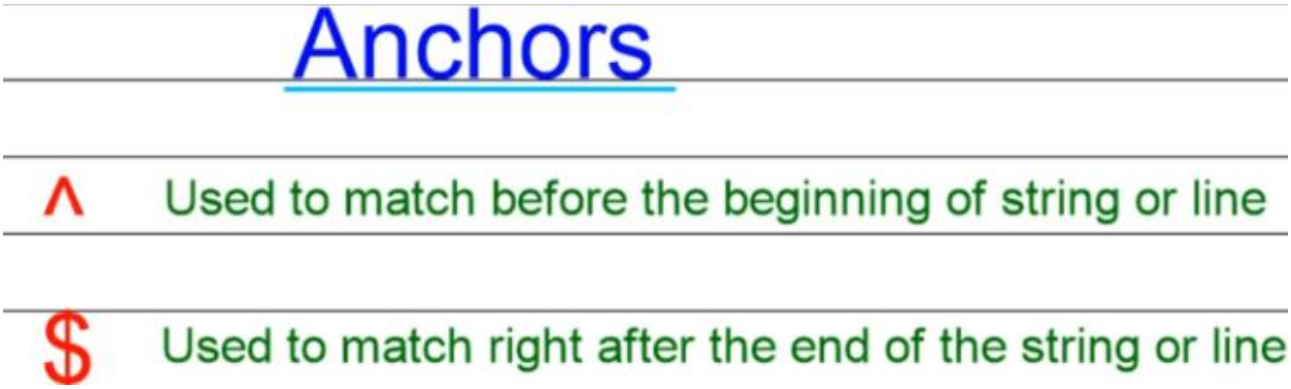


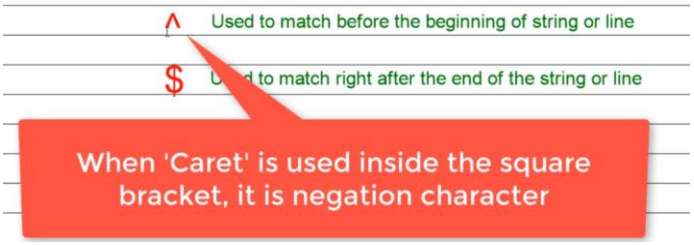
1.4 Anchors And Word Boundary

1.4.1 Anchors

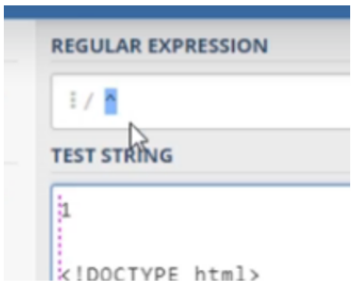
Are used the match before the beginning or end of

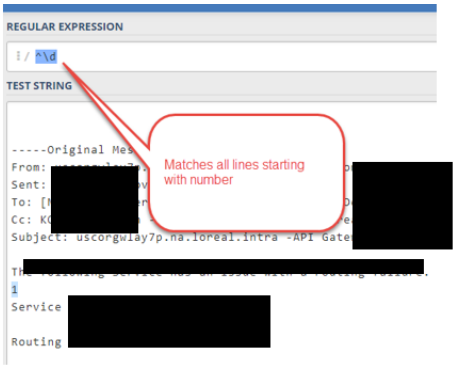
1) String
2) Line

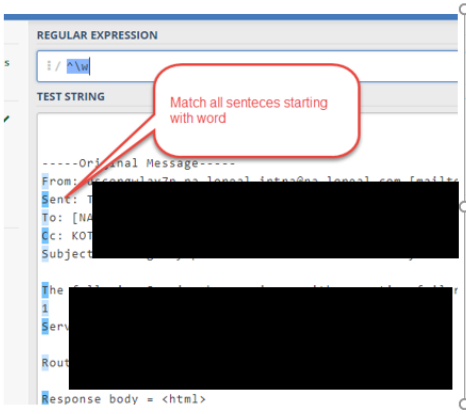


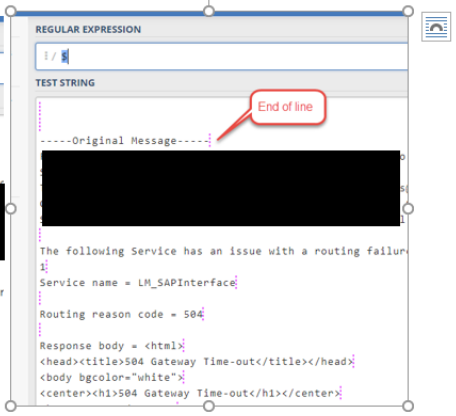

^ Used to match before the beginning of string or line
$ Used to match right after the end of the string or line
When 'Caret' is used inside the square bracket, it is negation character

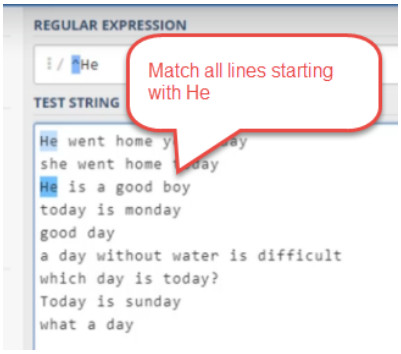

REGULAR EXPRESSION
TEST STRING
EXPLANATION
REGEX FLAGS
global
multi line
insensitive
extended
eXtra
Matches start and end of each line
-----Original Message-----

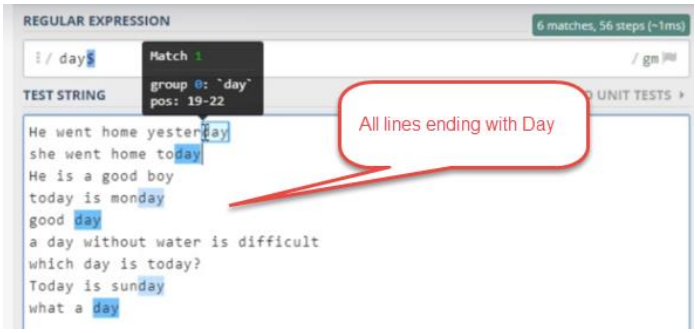

REGULAR EXPRESSION
TEST STRING
<!DOCTYPE html>

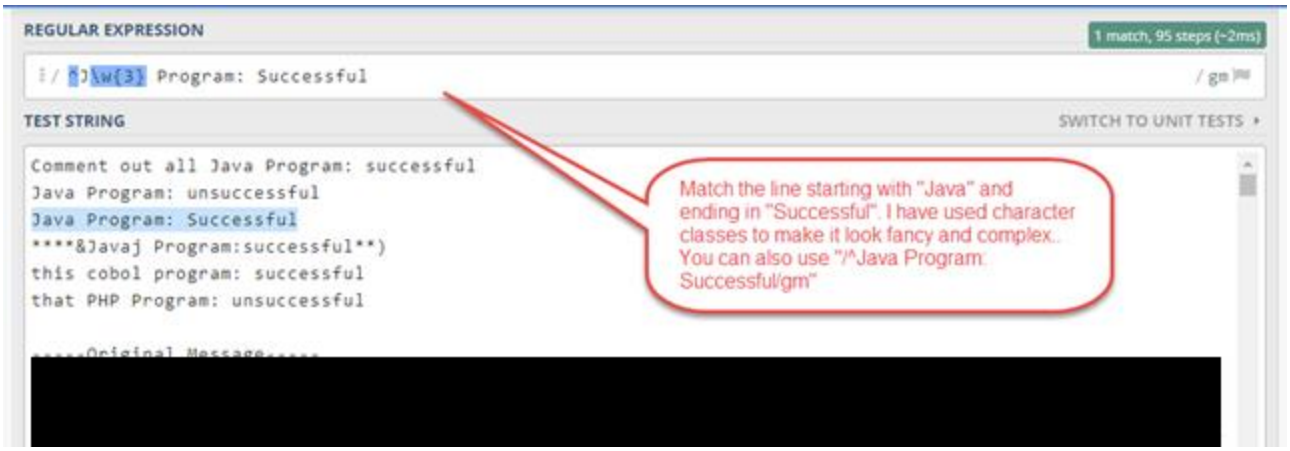

REGULAR EXPRESSION
TEST STRING
Matches all lines starting with number
-----Original Message-----


REGULAR EXPRESSION
TEST STRING
Match all senteces starting with word
-----Original Message-----
Response body = <html>


REGULAR EXPRESSION
TEST STRING
End of line
-----Original Message-----
The following Service has an issue with a routing failur
Service name = LM_SAPInterface
Routing reason code = 504
Response body = <html>
<head><title>504 Gateway Time-out</title></head>
<body bgcolor="white">
<h1>504 Gateway Time-out</h1>

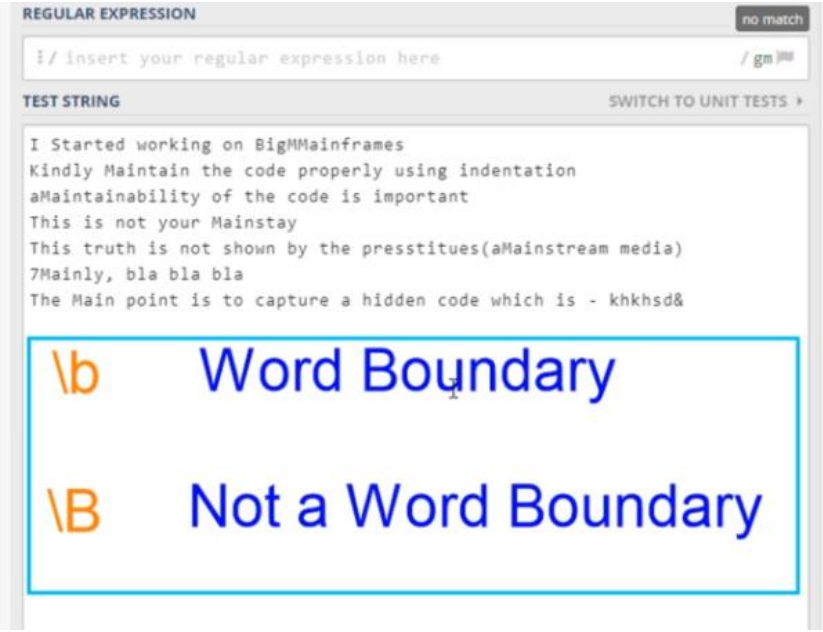


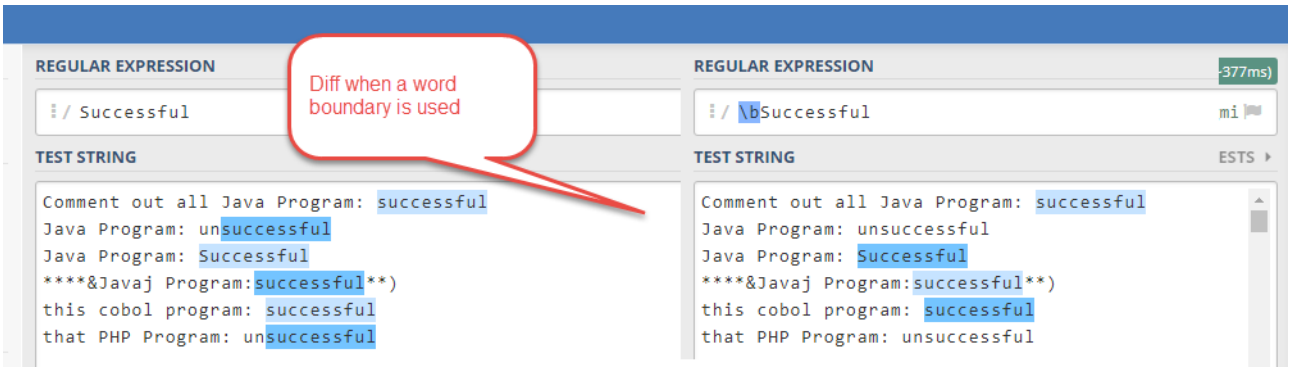


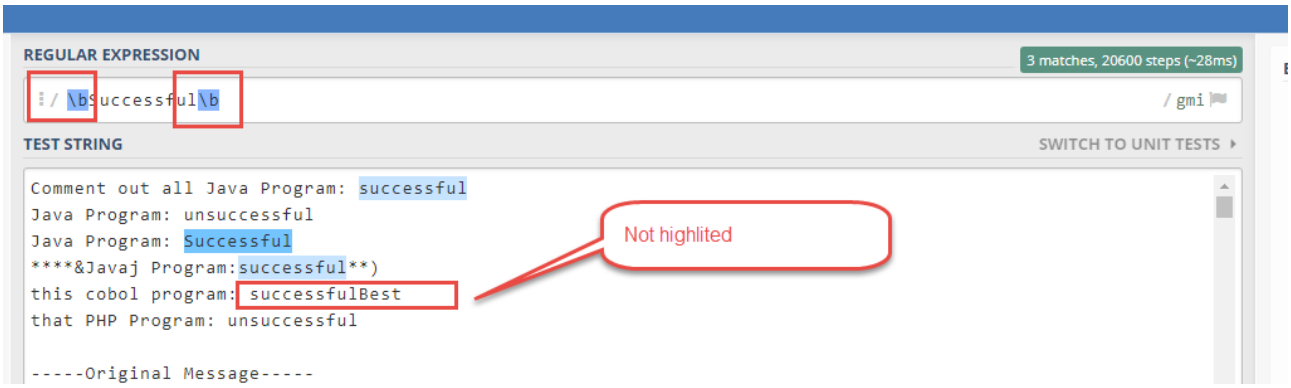


1.4.2 Word Boundary

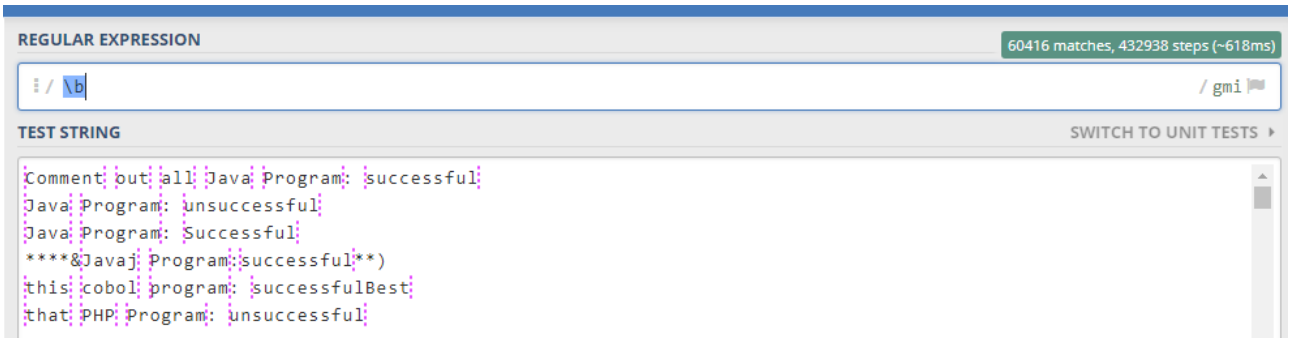
Is used to match the boundary between word and non-word character.



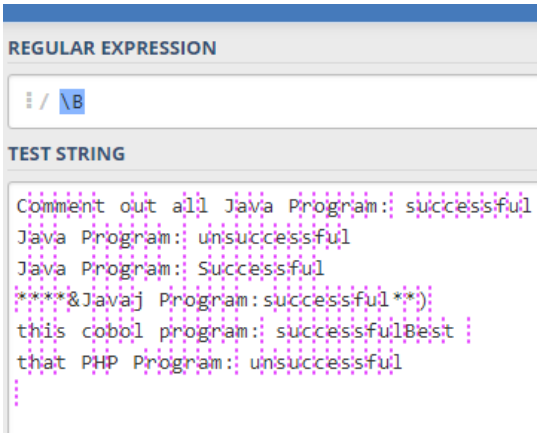


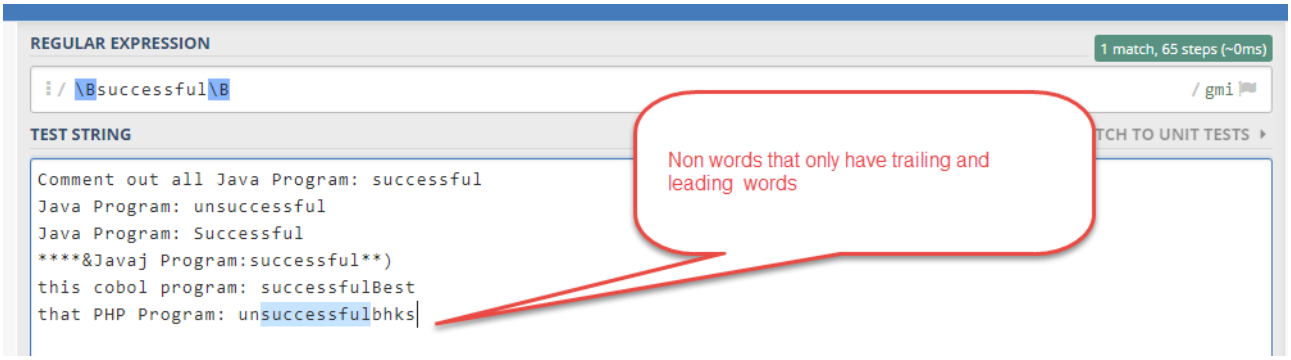


Captures all word boundaries



All non-word boundary characters.

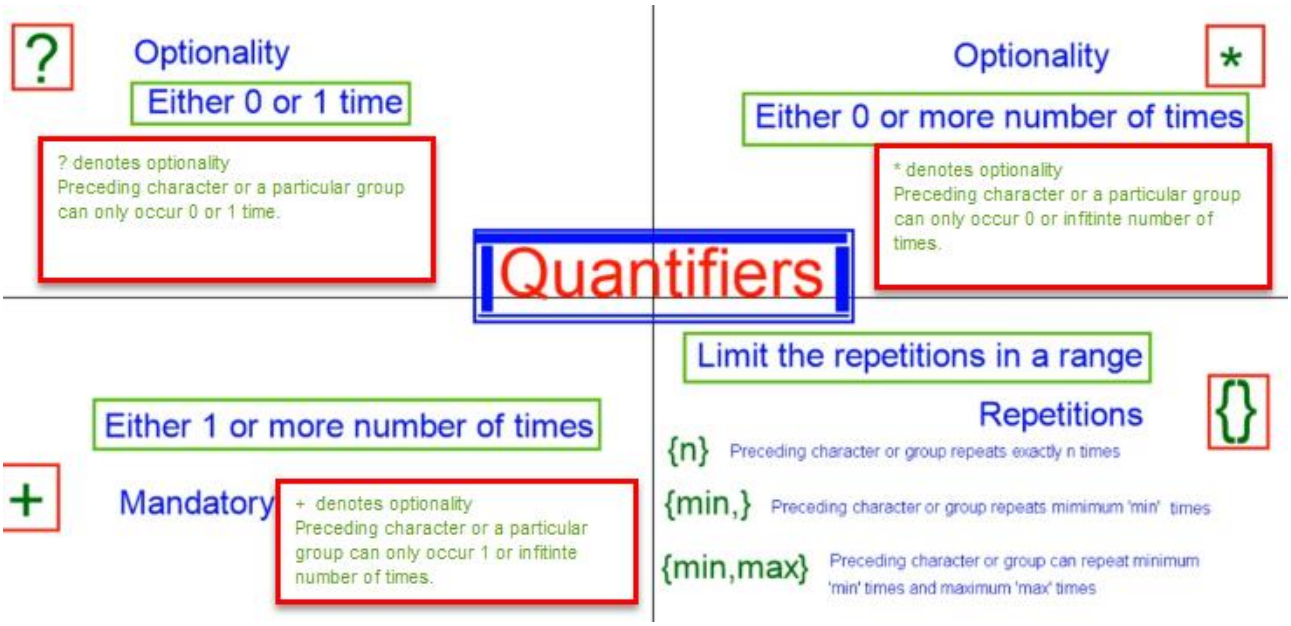




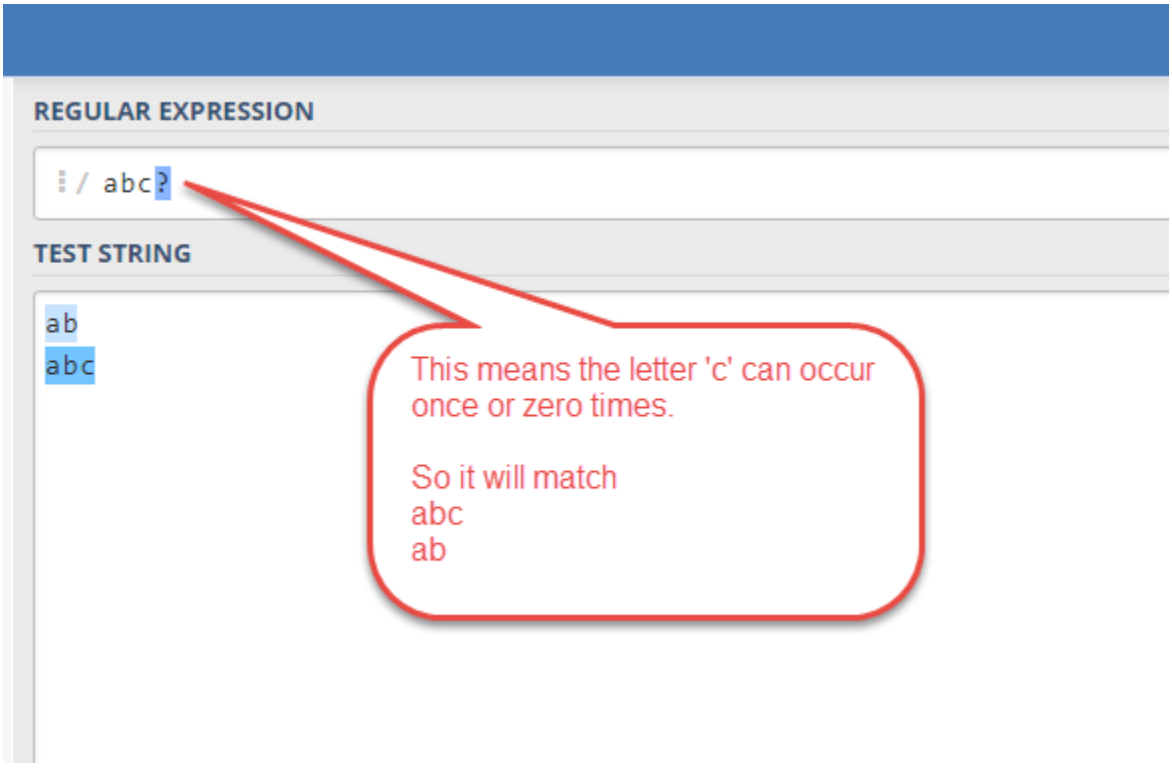
1.5 Quantifiers

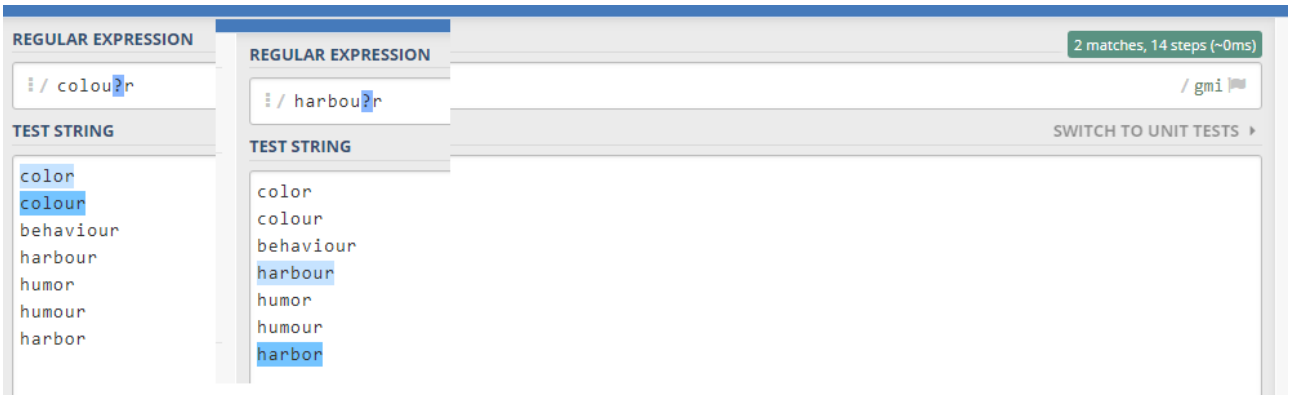
In regx there are 4 types of Quantifiers.

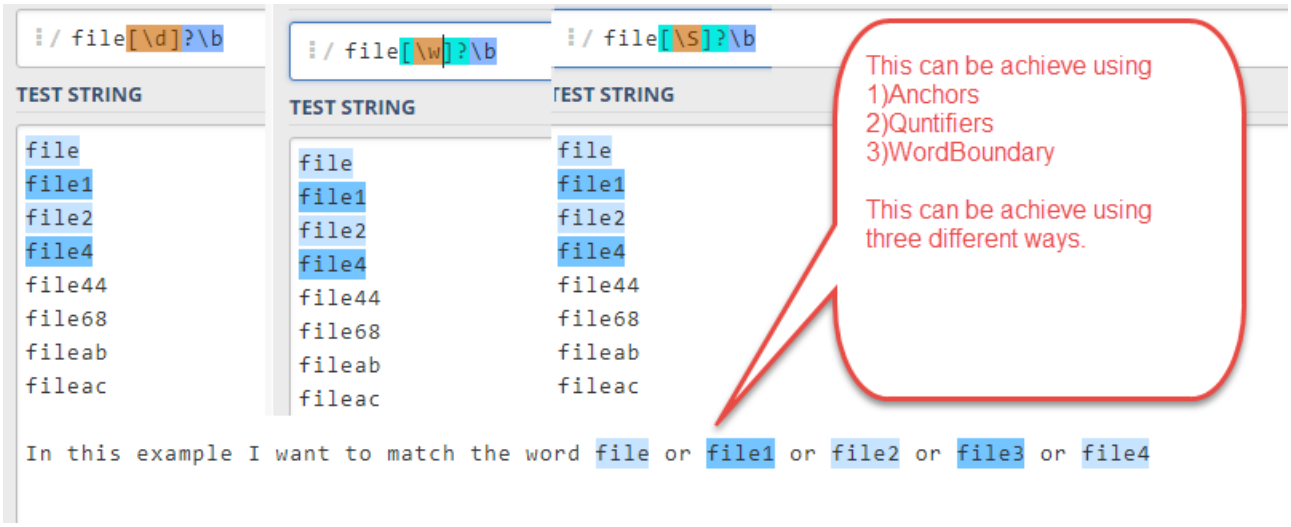
A quantifier is used after a charcter or group and decides how the character or group before the quantifier will occur.



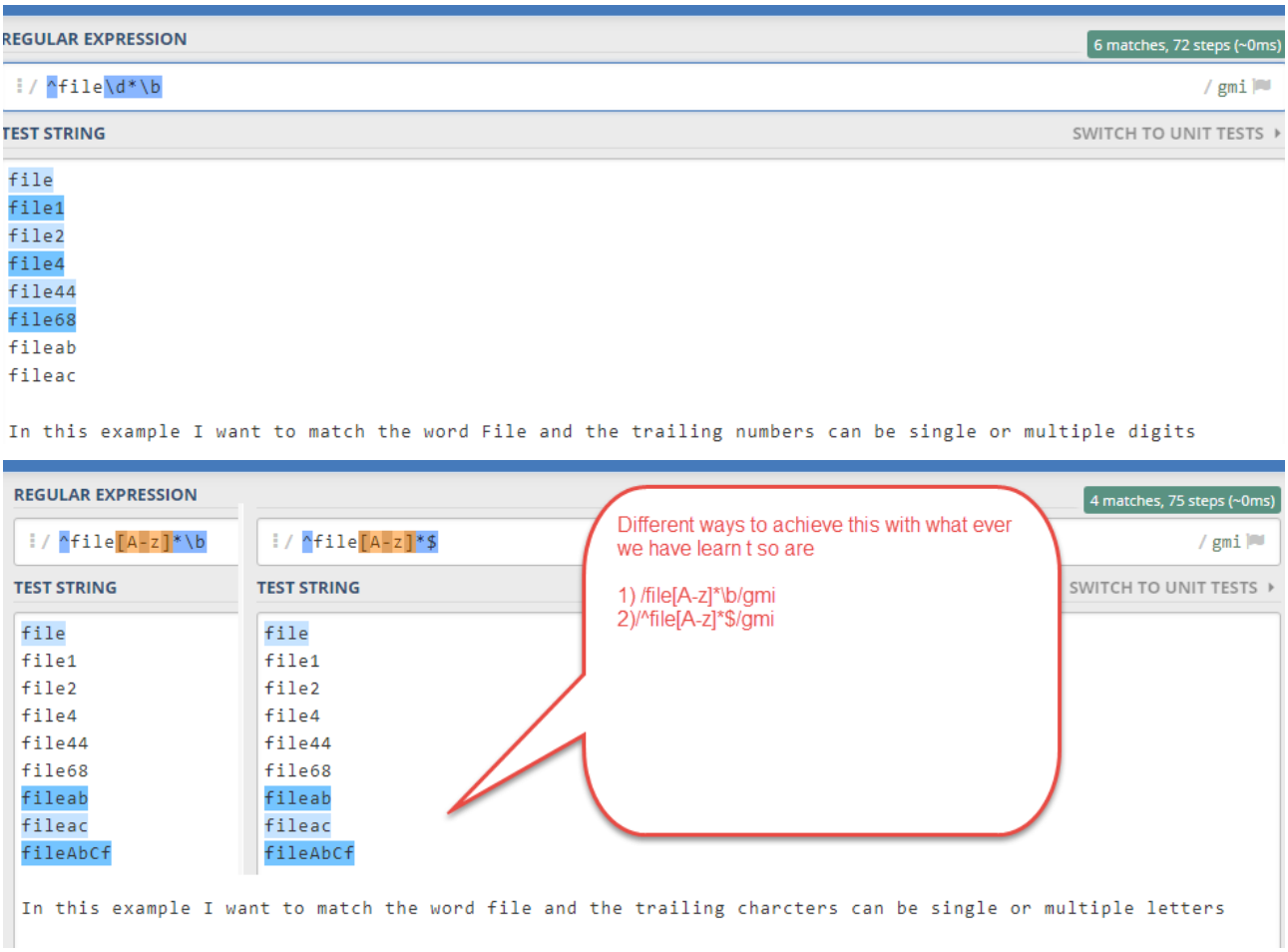
1.5.1 ? quantifier


REGULAR EXPRESSION
/ abc?
TEST STRING
ab
abc
This means the letter 'c' can occur once or zero times.
So it will match
abc
ab


REGULAR EXPRESSION
/ colou?r
TEST STRING
color
colour
behaviour
harbour
humor
humour
harbor
REGULAR EXPRESSION
2 matches, 14 steps (~0ms)
/ harbou?r
/ gmi
SWITCH TO UNIT TESTS
TEST STRING
color
colour
behaviour
harbour
humor
humour
harbor



1.5.2 * Quantifier



1.5.3 Matching Example

Q.1) Create a Regular Expression to match a pattern like below -

User has to provide a valid Name of company in an on line Form and it can have any alphanumeric value.

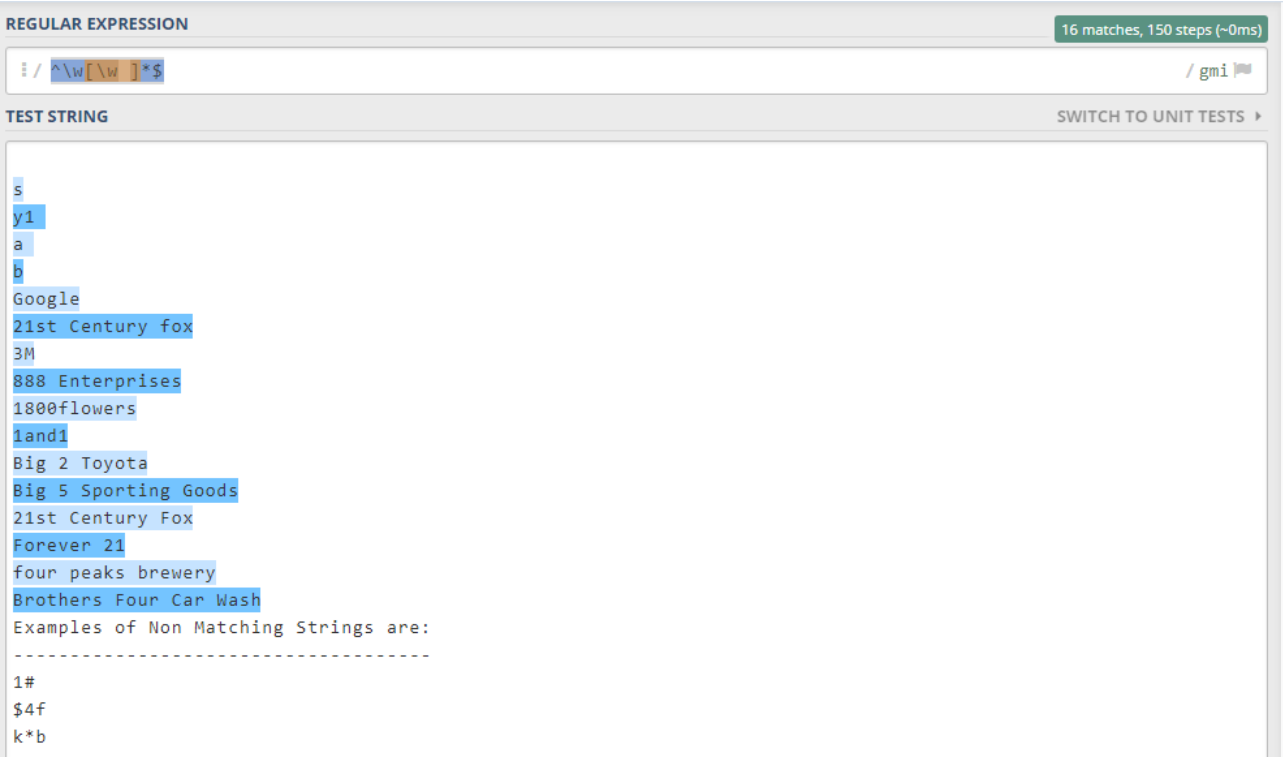
Hint:

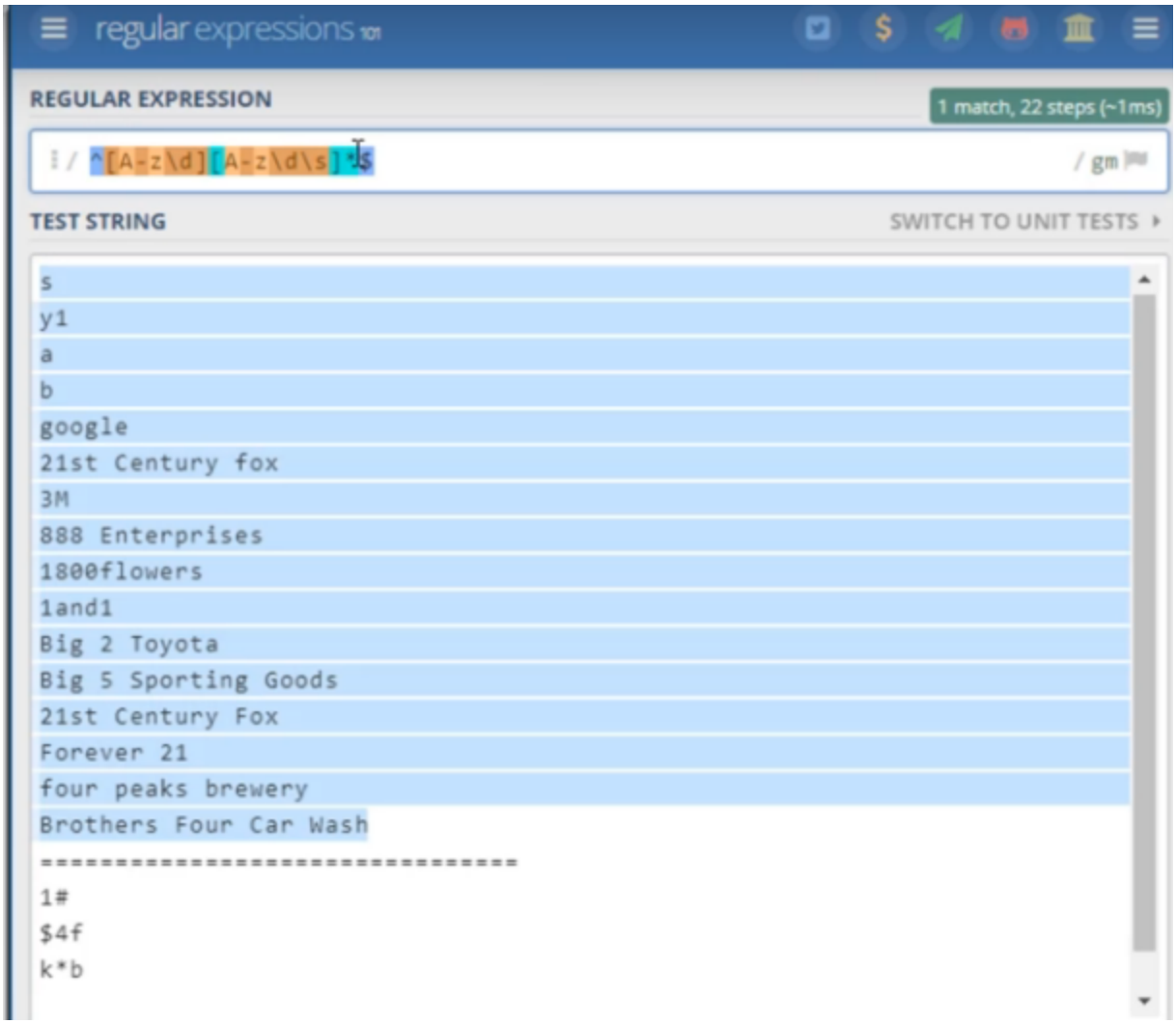
Name of a company can contain alphanumeric characters

Assumption:

Here, we assume that the names of the company does not contain any other special characters. If your requirement is to match a company name with special characters as well, then the RegeX will change accordingly.

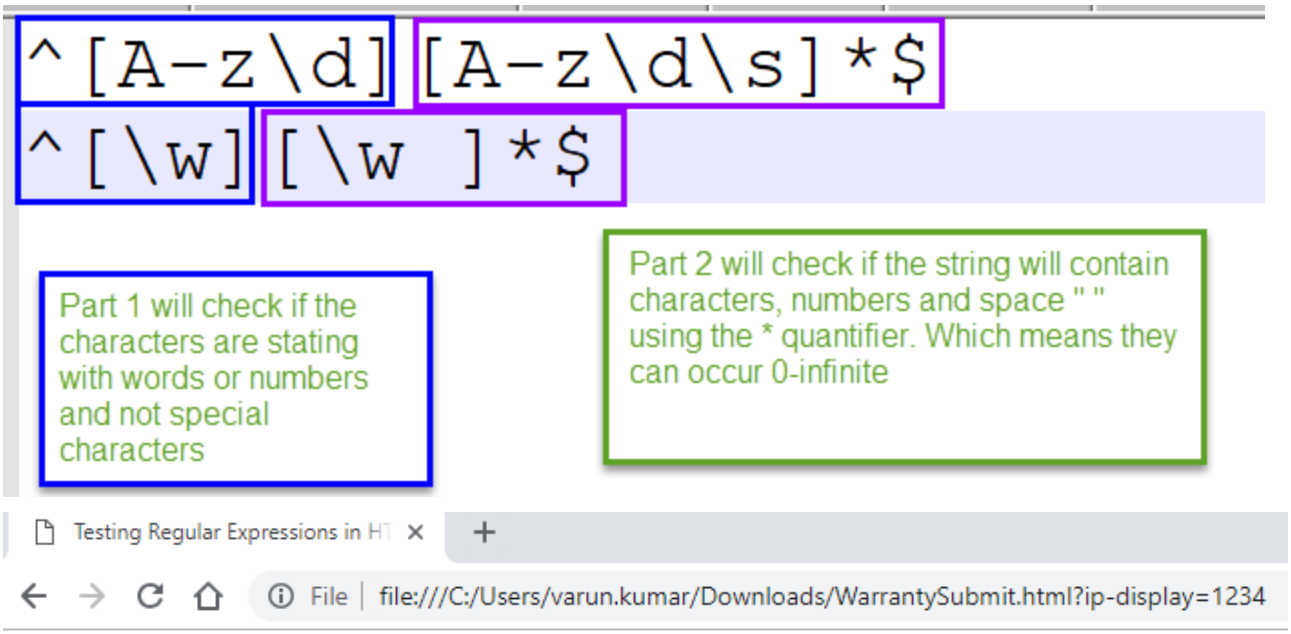
Different waves to achieve it.



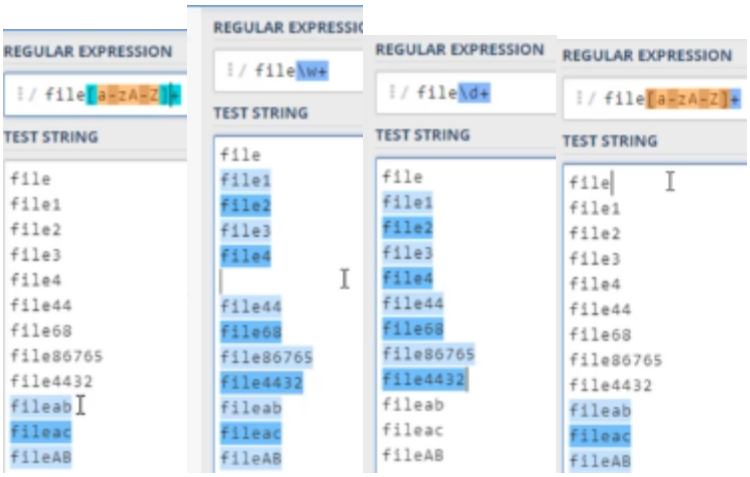

regular expressions 101
REGULAR EXPRESSION
1 match, 22 steps (~1ms)
/ ^[A-z\d][A-z\d\s]*$ / gm
TEST STRING
SWITCH TO UNIT TESTS
s
y1
a
b
google
21st Century fox
3M
888 Enterprises
1800flowers
1and1
Big 2 Toyota
Big 5 Sporting Goods
21st Century Fox
Forever 21
four peaks brewery
Brothers Four Car Wash
==========
1#
$4f
k*b


```
<!DOCTYPE html>
<html>
<head>
<title>Testing Regular Expressions in HTML forms</title>
</head>
<body>
<h1>Test your Regular Expressions here</h1>
<fieldset>
<legend>Write your Regular expressions</legend>
<form>
<div>
<label for="display-name"> Enter a company name here:
<span class="warning">*(Enter an alpha numeric character.)</span>
</label>
<input type="text" id="display-name" name="ip-display"
pattern="^[0-z][A-z\d\s]*$"
title="Special Characters Not Allowed"/>
</div>
<div>
<input type="submit" class="submit" value="Submit" />
</div>
</form>
</fieldset>
</body>
</html>
```

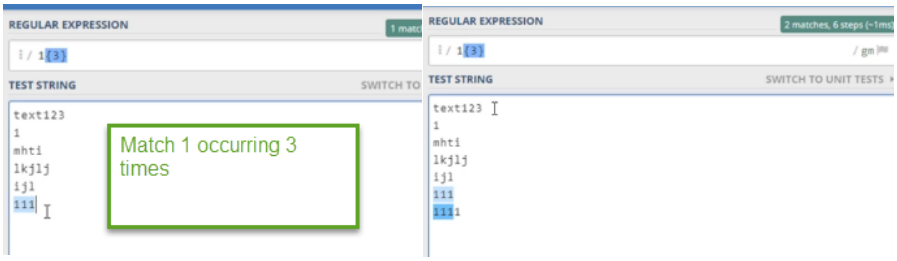
Command explanation.

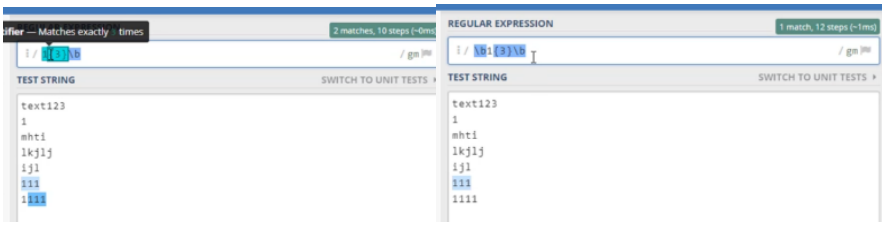


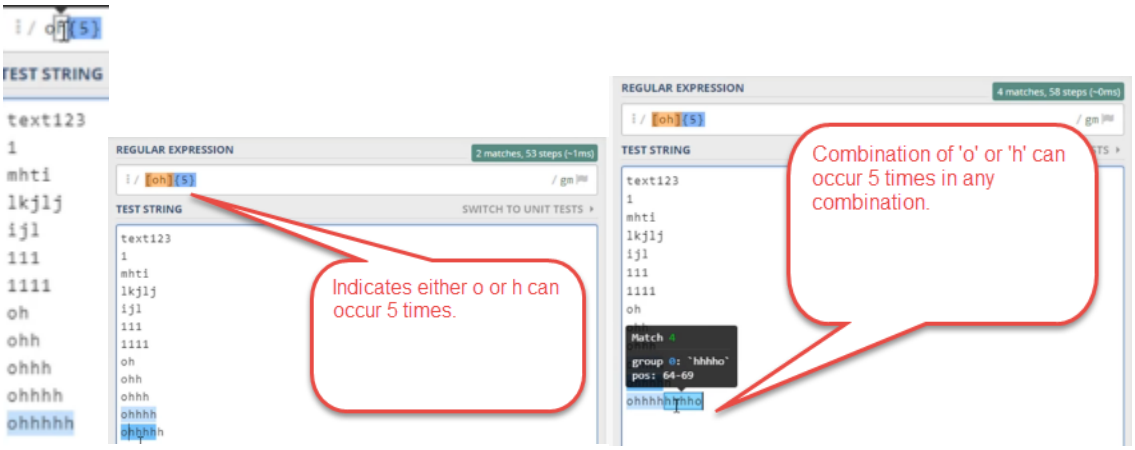
1.5.4 + Quantifier



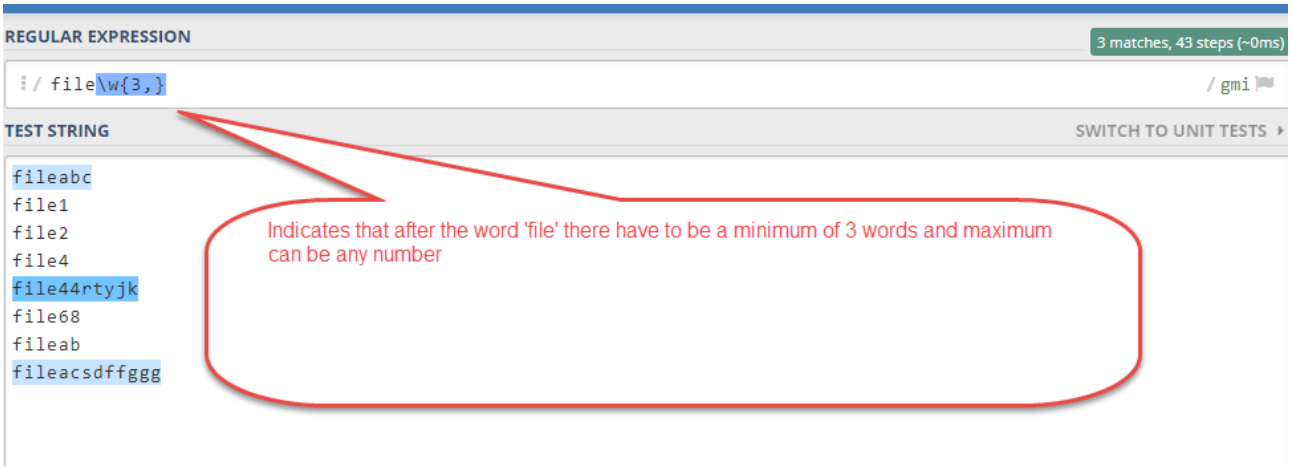
1.5.5 {} Quantifiers To Limit Range



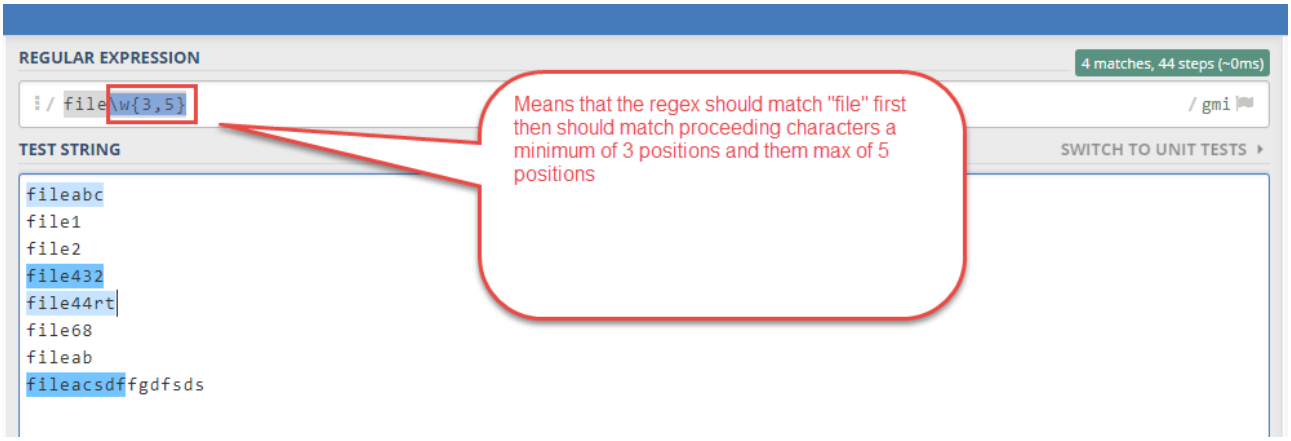


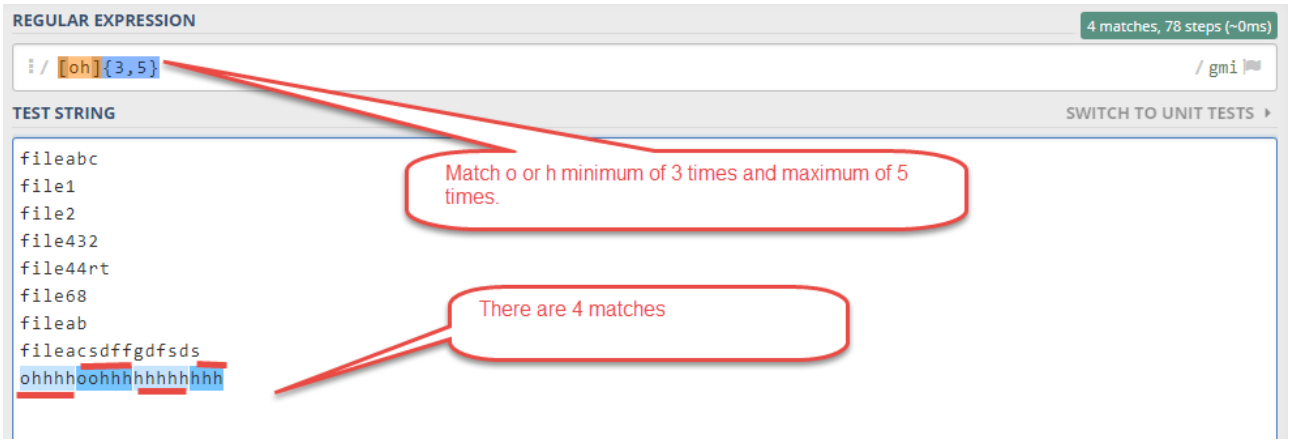


1.5.6 {min,} Quantifiers

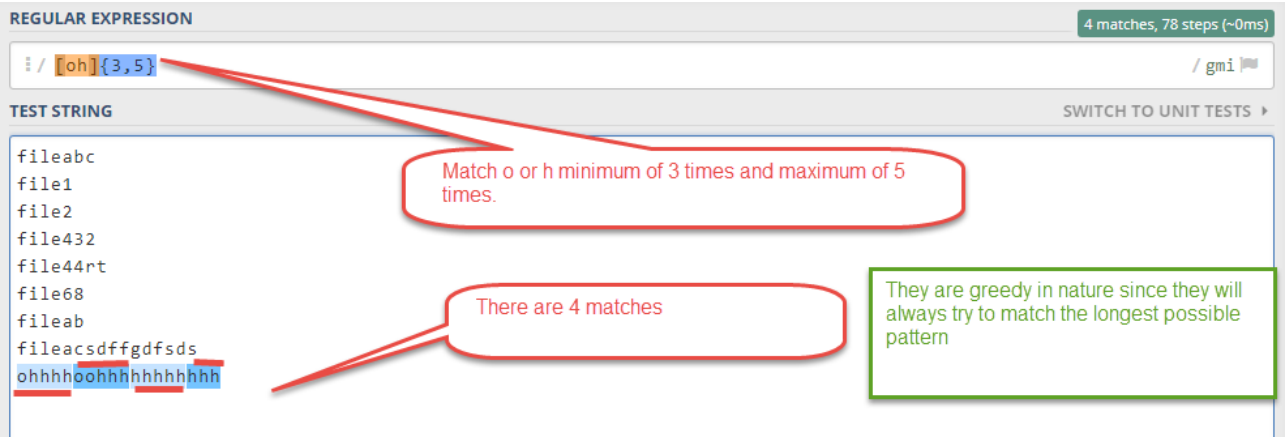


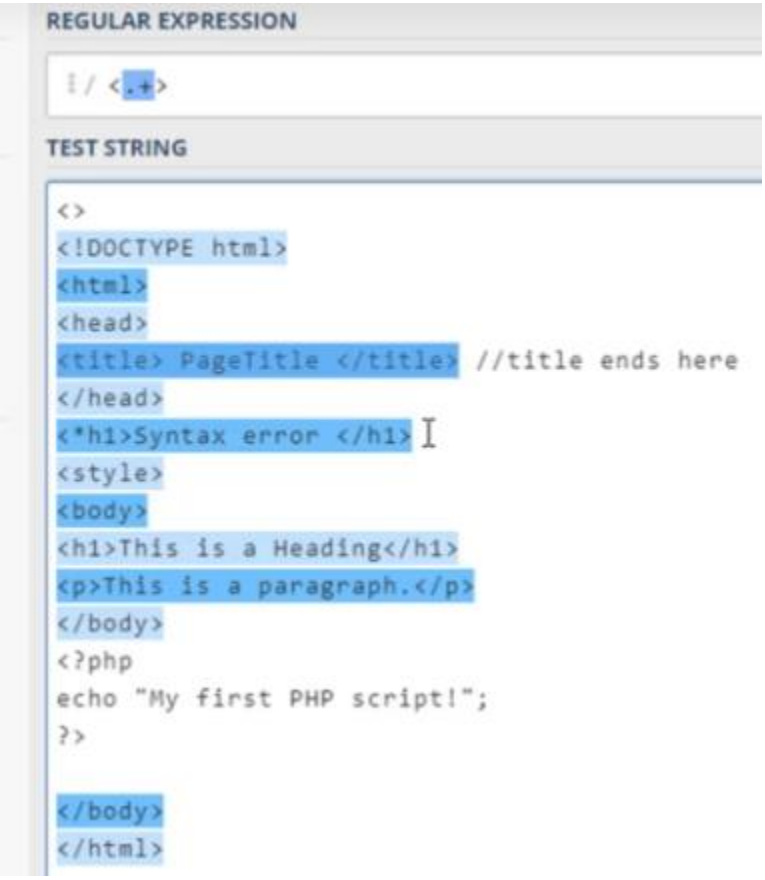
1.5.7 {min,max} Quantifiers





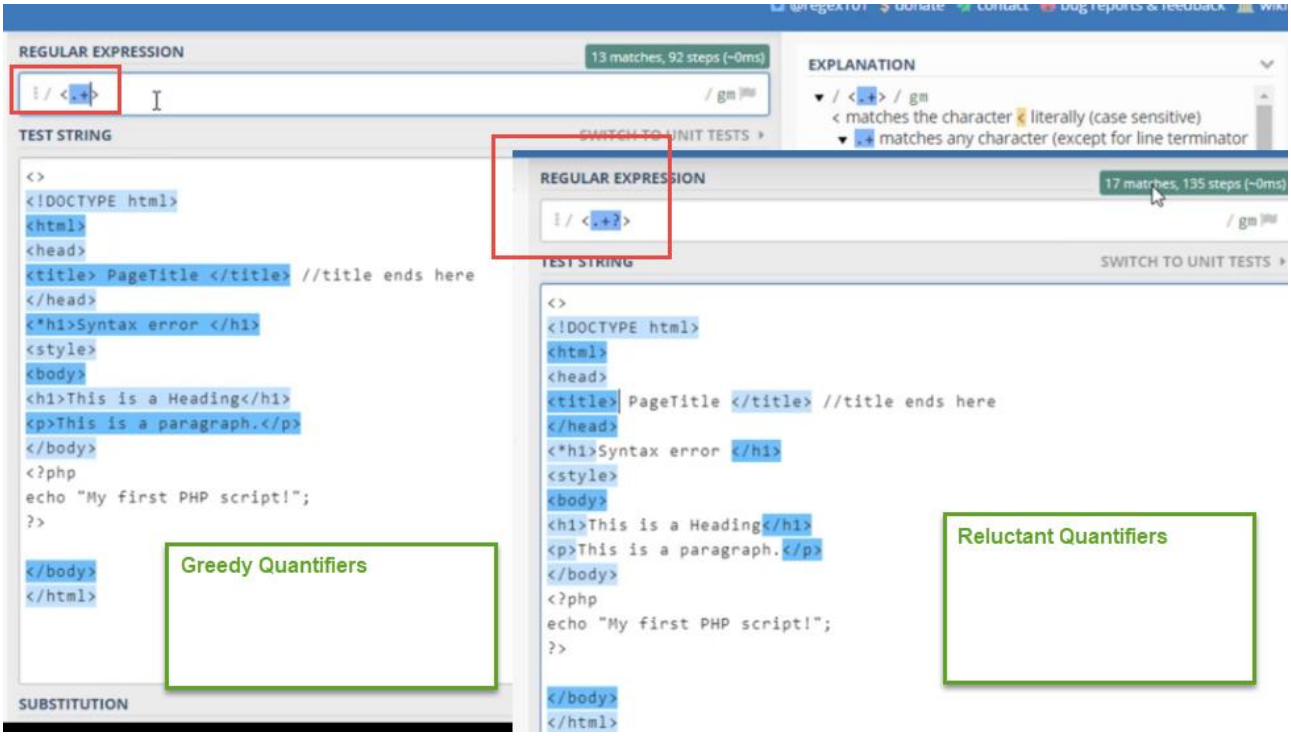
1.5.8 Greedy Quantifiers

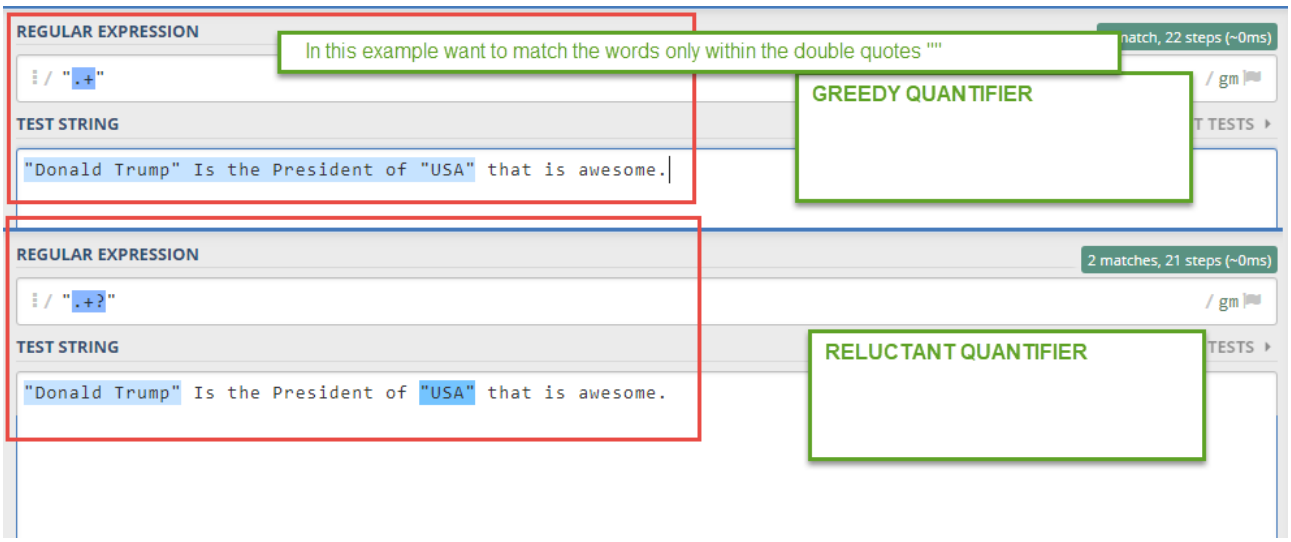




There is a fix for the greedy natures of the quantifiers by making the quantifiers “Lazy” or “Reluctant” quantifiers.

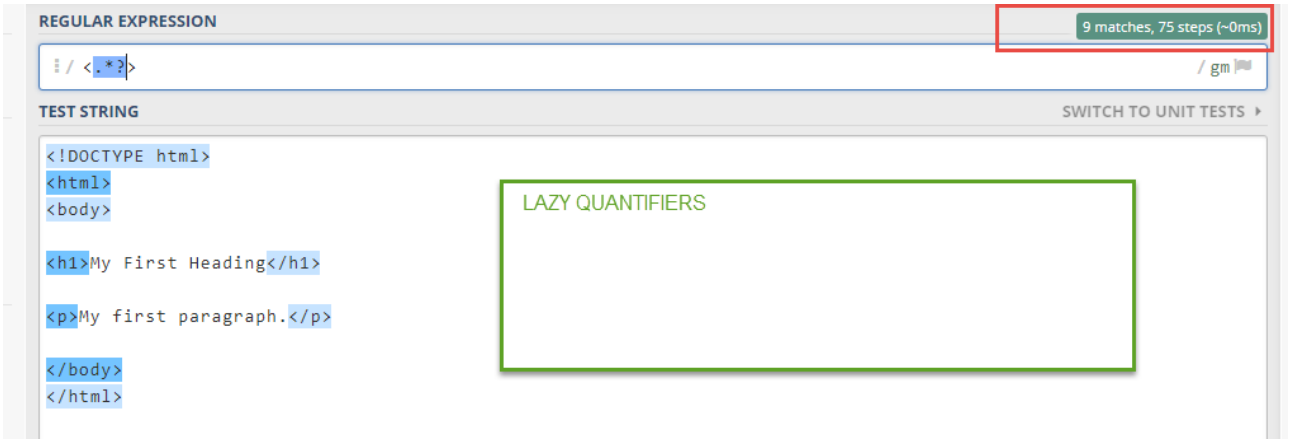
1.5.9 Lazy Or Reluctant Quantifiers

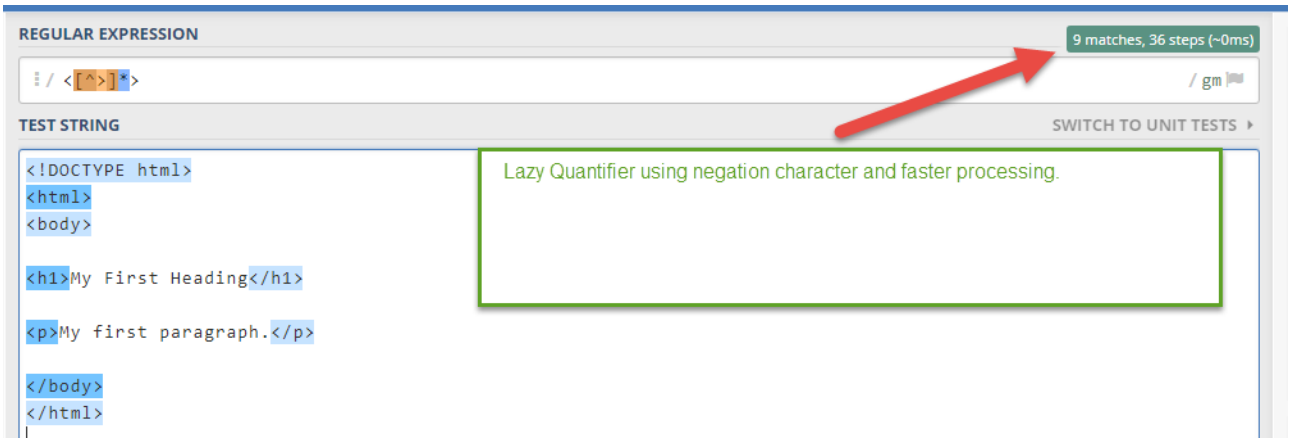


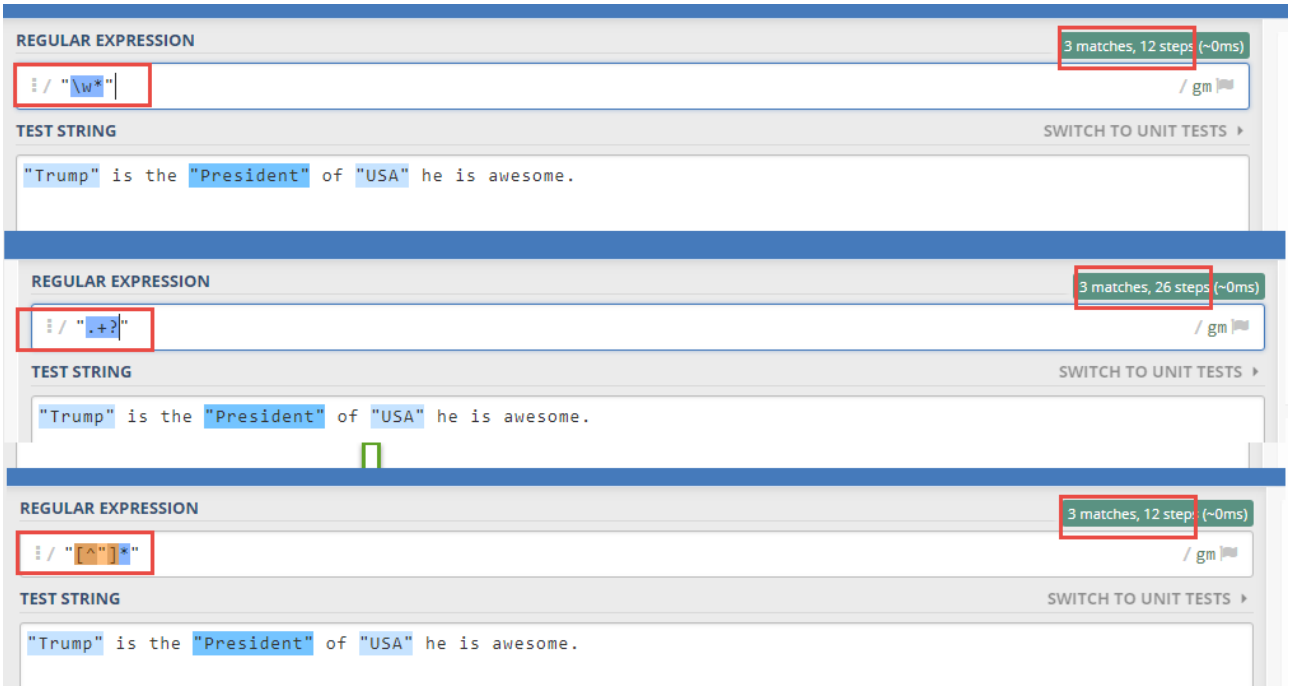


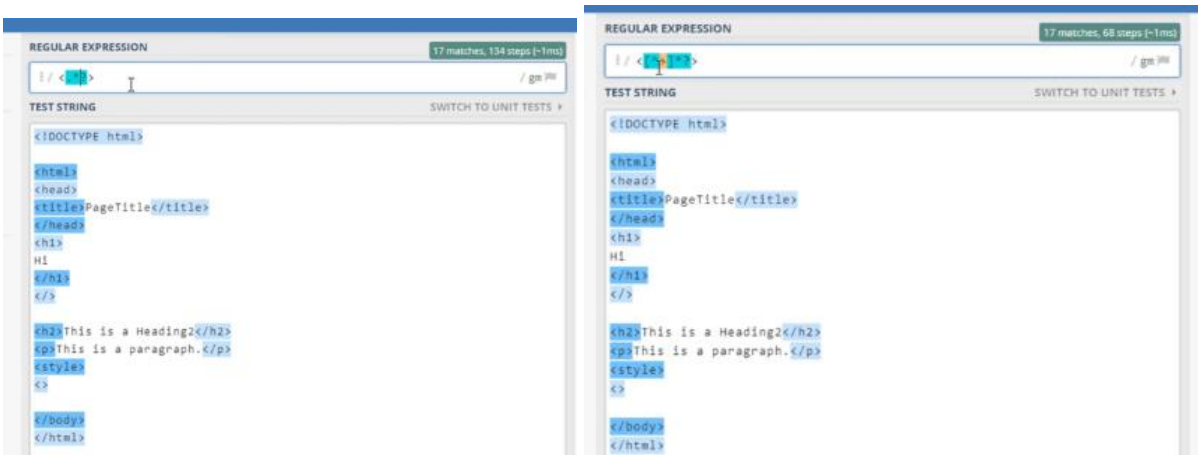
1.5.10 Alternative For Lazy Quantifiers

When making a quantifier Lazy this will result in increased processing time and this when used in applications(bulding live applications) will cause lags. This can be over come by using negation characters.

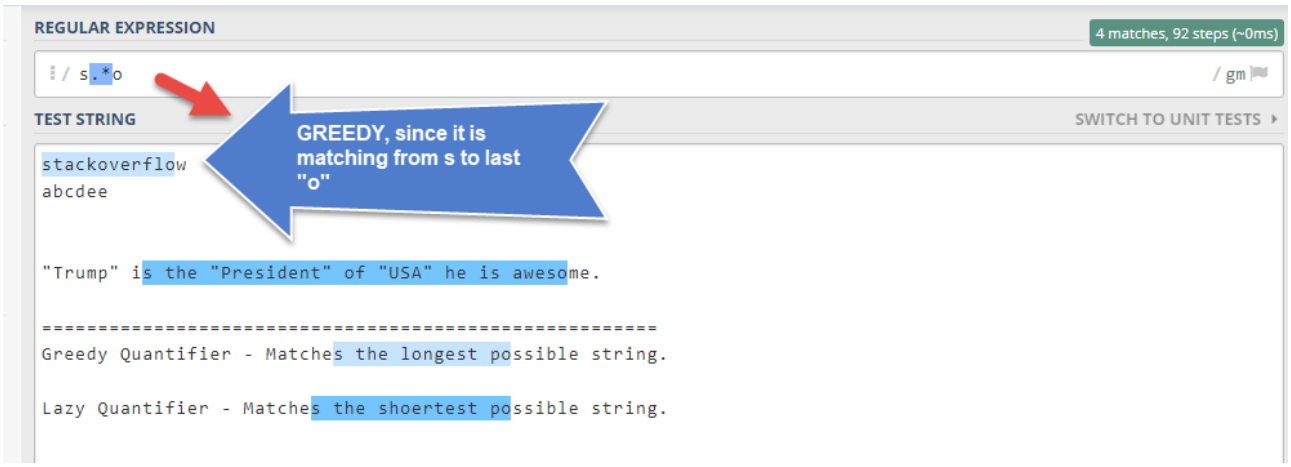

REGULAR EXPRESSION
9 matches, 75 steps (~0ms)
/ <.*?>
/ gm
TEST STRING
SWITCH TO UNIT TESTS
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
My first paragraph.
</body>
</html>
LAZY QUANTIFIERS

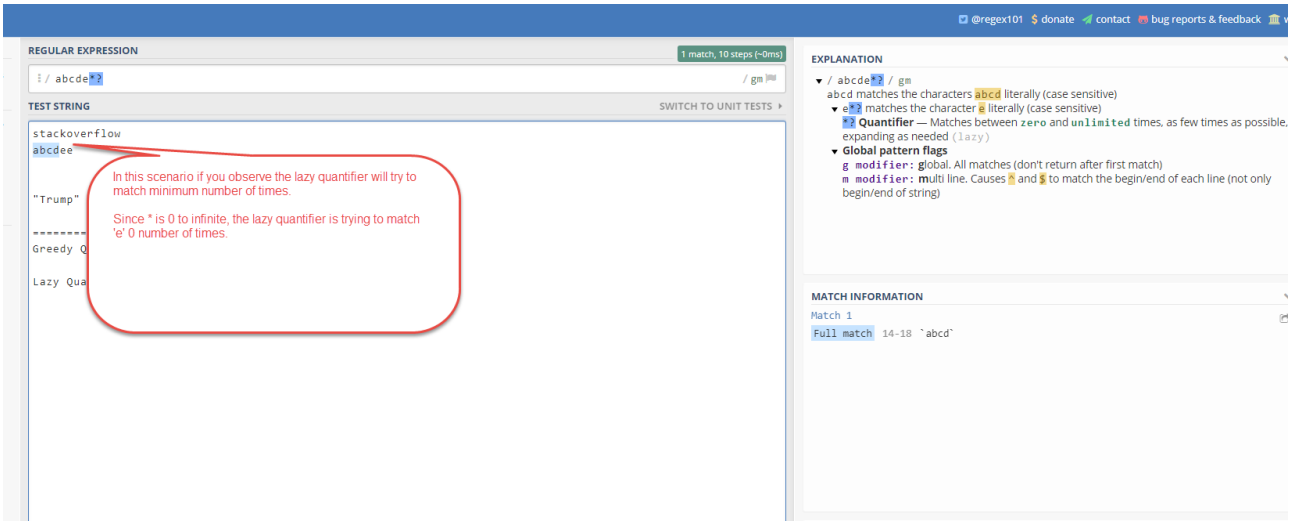

REGULAR EXPRESSION
9 matches, 36 steps (~0ms)
/ <[^>]*>
/ gm
TEST STRING
SWITCH TO UNIT TESTS
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
My first paragraph.
</body>
</html>
Lazy Quantifier using negation character and faster processing.

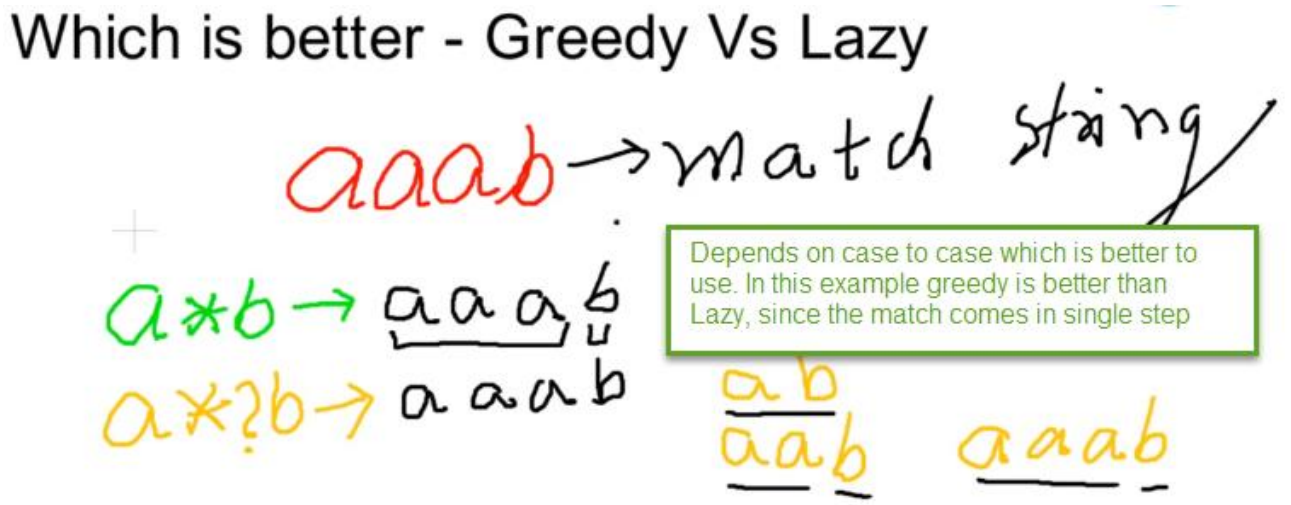

REGULAR EXPRESSION
3 matches, 12 steps (~0ms)
/ "\w*"
/ gm
TEST STRING
SWITCH TO UNIT TESTS
"Trump" is the "President" of "USA" he is awesome.
REGULAR EXPRESSION
3 matches, 26 steps (~0ms)
/ ".+?"
/ gm
TEST STRING
SWITCH TO UNIT TESTS
"Trump" is the "President" of "USA" he is awesome.
REGULAR EXPRESSION
3 matches, 12 steps (~0ms)
/ "[^"]*"
/ gm
TEST STRING
SWITCH TO UNIT TESTS
"Trump" is the "President" of "USA" he is awesome.

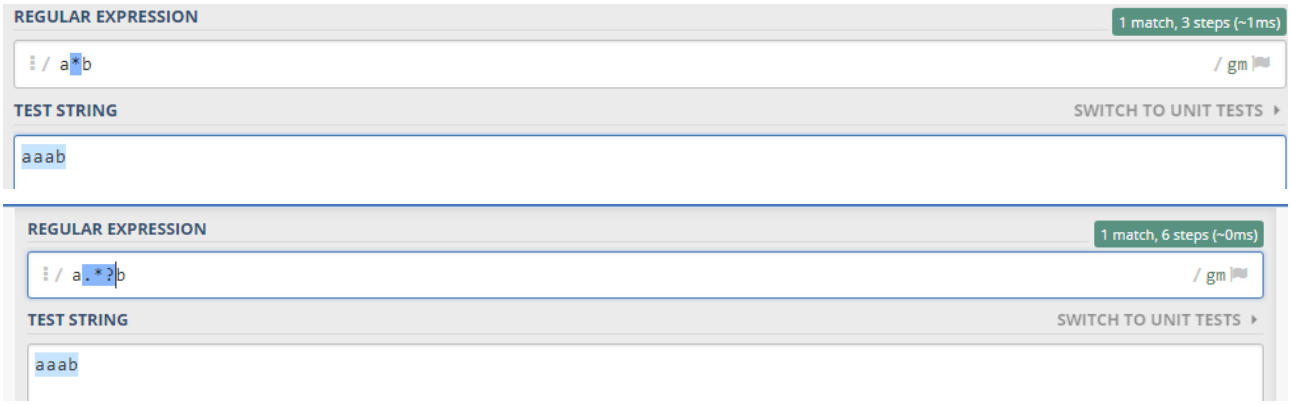


1.5.11 Greedy Quantifiers vs Lazy Quantifiers









In the pattern match of xml tags we have seen earlier the Lazy quantifiers are much better than greedy quantifiers.

1.5.12 Increasing the performance of the RegeX

The normal behavior of a quantifier is always Greedy and in some cases, Greedy quantifiers can lead to performance issues and in some cases, Lazy Quantifier can also lead to performance issues.

Note:

If we find that Greedy Quantifiers have performance issue in your search pattern, then try changing it to Lazy Quantifier and see if it improves the performance. On the other hand, if you have a lazy Quantifier which has a performance issue, try to change it into Greedy Quantifier to see if the performance increases. We can also try to use the Negation character to improve the performance.

The key to performance is to always remember -

**** Greedy matches the longest possible string**
**** Lazy matches the smallest possible string**
So, remember this, if we try to make Greedy quantifier as Lazy then the meaning will change like this -

{min,max}? - Repeat minimum 'min' times and maximum 'max' times, but as few times as possible(lowest is 'min' times)
{min,}? - Repeat minimum 'min' times and maximum any times, but as few times as possible(lowest is 'min' times)
***?** - Repeat any number of times, but as few times as possible(lowest is 0 time)
+? - Repeat any number of times, but as few times as possible(lowest is 1 time)
?? - Repeat either 0 time or 1 time, but as few times as possible(lowest is 0 time)

1.6 Groups In Regex

To Be Updated

1.7 Assertions

To Be Updated

1.8 Real Life Scenarios

To Be Updated