Basic Regex patterns

Notebook:	Managed Services		
Created:	12/20/2017 11:01 AM	**Updated:**	12/20/2017 2:24 PM
Author:	tuan.hoang@episerver.com		
Tags:	Regex		
URL:	https://regex101.com/		

Basic Regex patterns

- Regex online-testing tool: https://regex101.com/

- References:

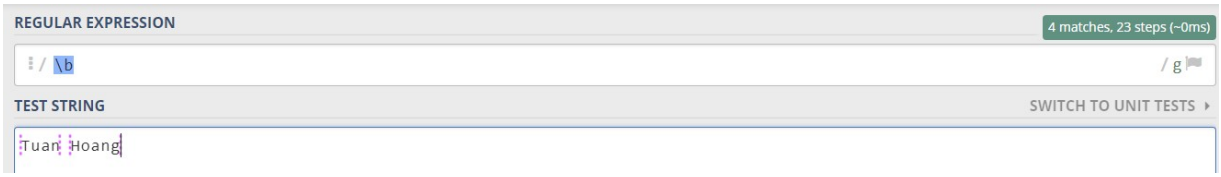
http://www.rexegg.com/
http://www.rexegg.com/regex-quickstart.html
http://www.zytrax.com/tech/web/regex.htm
https://www.cheatography.com/davechild/cheat-sheets/regular-expressions/

Note: Some regex patterns may not work in all environments

A. Basic Patterns

+) Word boundary match

Pattern: \b



+) Non-word boundary match

Pattern: \B

+) Word character match

Pattern: \w

+) Non-word character match => matches any non-word character (equal to [^a-zA-Z0-9_])

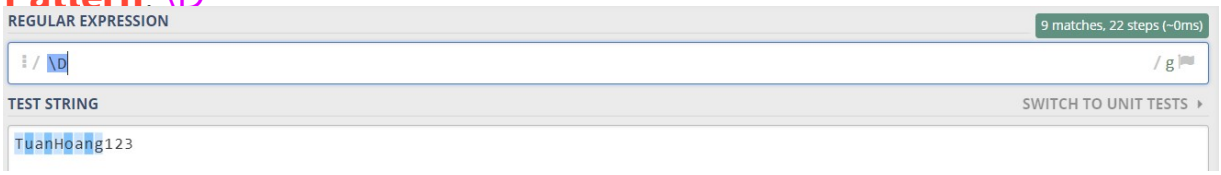
Pattern: \W

+) Digit match

Pattern: \d

+) Non-digit match

Pattern: \D

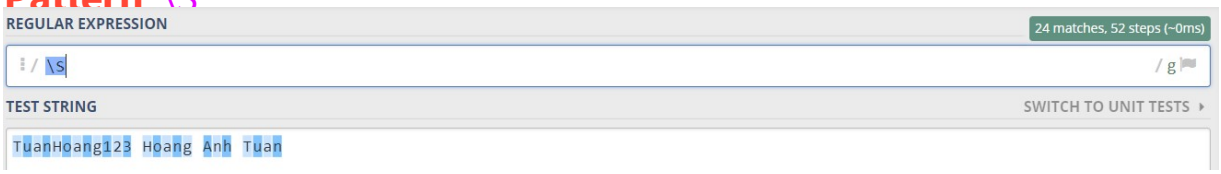


+) White-space match

Pattern: \s

+) Non-whitespace match

Pattern: \S



+) Quantifiers

Quantifiers			
*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

+) Groups and ranges

Groups and Ranges	
.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"
Ranges are inclusive.	

+) Anchors

Anchors	
^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

+) Modifiers

Pattern Modifiers	
g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern
* PCRE modifier	

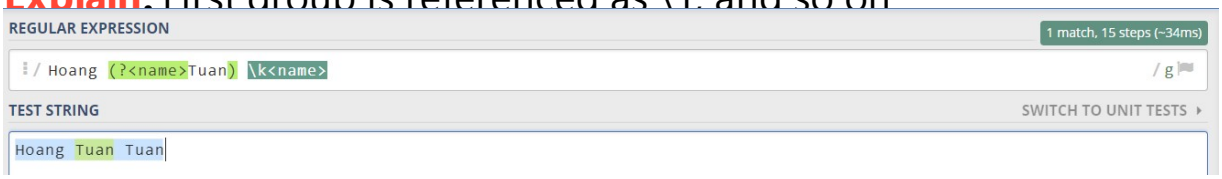
B. Advanced Patterns

1. Named capturing group

Pattern: Hoang (?<name>Tuan) \k<name>
Matched string: Hoang Tuan Tuan
Explain: This pattern will capture the string inside (?<name>...) and hold it in the "name" for back-reference. Later, we can reference to it by \k<name>

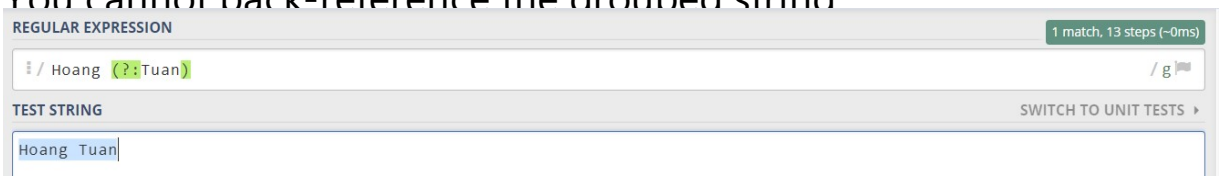
2. No-name capturing group

Pattern: (Hoang) (Tuan) \2 \1
Matched string: Hoang Tuan Tuan Hoang
Explain: First group is referenced as \1, and so on



3. Non-capturing group

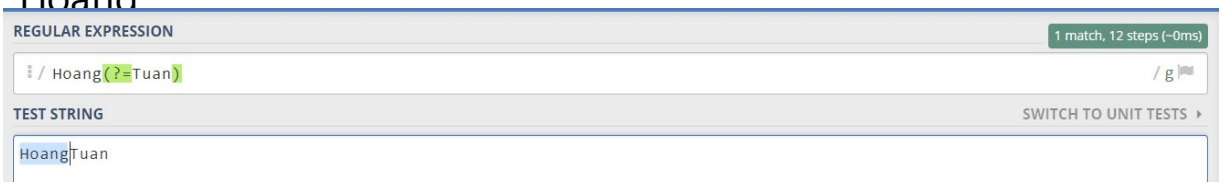
Pattern: Hoang (?:Tuan)
Matched string: Hoang Tuan
You cannot back-reference the grouped string



4. Positive Lookahead

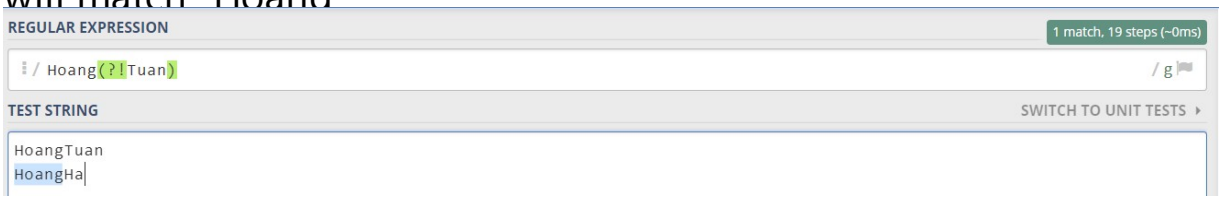
Pattern: Hoang(?=Tuan)

This pattern will match the string "Hoang" in "HoangTuan" but does not match the string "Hoang" in "HoangHa" & "TuanHoang"
Explain: When found "Hoang", it will look ahead to see if the string "Tuan" exist or not. If "Tuan" exists ahead of "Hoang", it will match "Hoang"



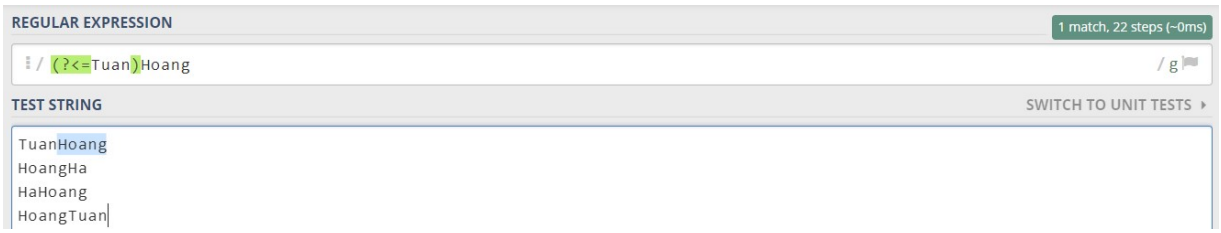
5. Negative Lookahead

Pattern: Hoang(?!Tuan)
This pattern will match the string "Hoang" in "HoangHa" but does not match the string "Hoang" in "HoangTuan" & "TuanHoang"
Explain: When found "Hoang", it will look ahead to see if the string "Tuan" exist or not. If "Tuan" DOES NOT exist ahead of "Hoang", it will match "Hoang"



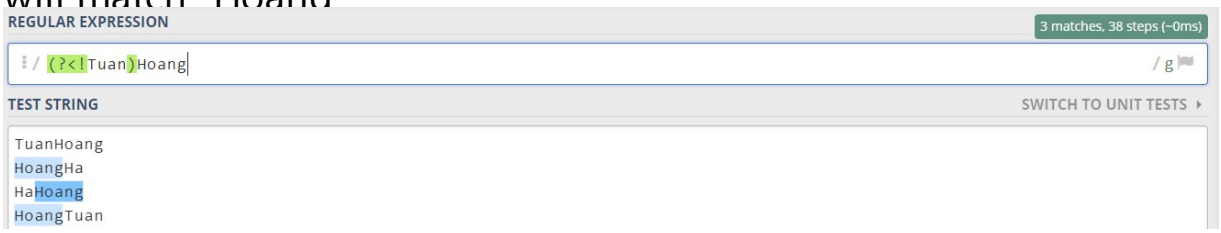
6. Positive Lookbehind

Pattern: (?<=Tuan)Hoang
This pattern will match the string "Hoang" in "TuanHoang" but does not match the string "Hoang" in "HaHoang" & "HoangTuan"
Explain: When found "Hoang", it will look behind to see if the string "Tuan" exist or not. If "Tuan" exists behind of "Hoang", it will match "Hoang"



7. Positive Lookbehind

Pattern: (?<!Tuan)Hoang
This pattern will match the string "Hoang" in "HaHoang" but does not match the string "Hoang" in "TuanHoang" & "HoangTuan"
Explain: When found "Hoang", it will look behind to see if the string "Tuan" exist or not. If "Tuan" DOES NOT exist behind of "Hoang", it will match "Hoang"



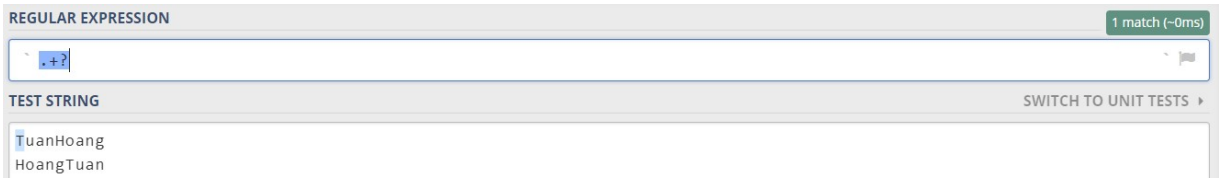
8. Greedy & Lazy quantifiers

By default, all quantifiers are greedy, they will try to match as much as possible

To make a quantifier to be lazy (i.e match as few as possible), put ? next to the quantifier

Example:
Greedy version (.+)

Lazy version (.+)


REGULAR EXPRESSION
1 match (~0ms)
` .+?
TEST STRING
SWITCH TO UNIT TESTS
TuanHoang
HoangTuan