REGULAR EXPRESSION

1. Introduce

* History:

The concept of regular expressions began in the 1950s when the American mathematician [Stephen Cole Kleene](https://en.wikipedia.org/wiki/Stephen_Cole_Kleene) formalized the concept of a [regular language](https://en.wikipedia.org/wiki/Regular_language). They came into common use with [Unix](https://en.wikipedia.org/wiki/Unix) text-processing utilities. Different [syntaxes](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) for writing regular expressions have existed since the 1980s, one being the [POSIX](https://en.wikipedia.org/wiki/POSIX) standard and another, widely used, being the [Perl](https://en.wikipedia.org/wiki/Perl) syntax.

<https://www.youtube.com/watch?v=528Jc3q86F8> (Chú ý 5 phút đầu video khi làm slide)

* Definition:

Formally, a regular expression is an algebraic notation for characterizing a set of strings. They are particularly useful for searching in texts when we have a **pattern** to search for and a **corpus** of  
texts to search through

NOTE: A search can be designed to return every match on a line if there are more than  
one, or just the first match

What is a pattern, corpus?

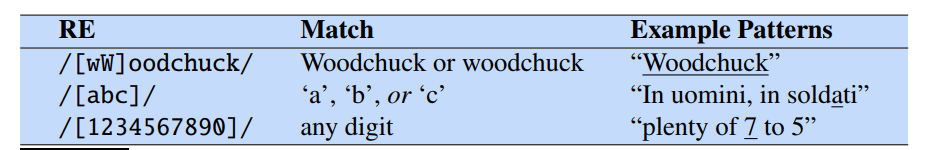
* How does regular expression work: https://www.youtube.com/watch?v=YBTvrkRg0FA

1. Basic regular expression

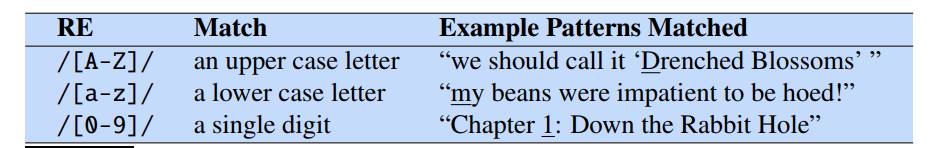
* 2.1 Regular expressions are **case sensitive**; lower case /s/ is distinct from upper  
  case /S/ (/s/ matches a lowercase *s* but not an uppercase *S*). This means that  
  the pattern /woodchucks/ will not match the string *Woodchucks*

| RE | Match | Example pattern matched |
| --- | --- | --- |
| /woodchucks/ | Any corpus has ‘woodchucks’ | ‘interesting links to woodchucks and lemus |
| /a/ | Any corpus has ‘a’ | ‘Mary Ann stopped by Mona’s |

We can solve this problem with the use of the square braces [ and ].

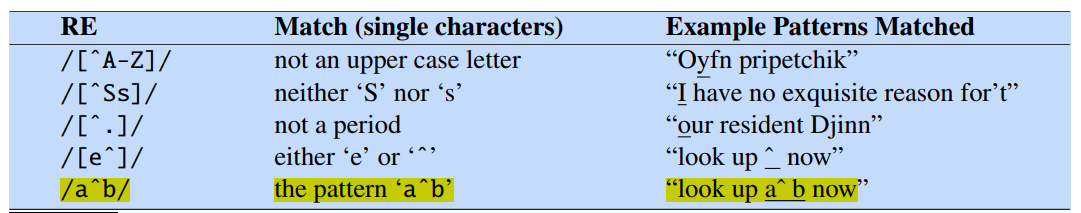


* 2.2 The regular expression /[1234567890]/ specifies any single digit. While such  
  classes of characters as digits or letters are important building blocks in expressions,  
  they can get awkward (e.g., it’s inconvenient to specify  
  /[ABCDEFGHIJKLMNOPQRSTUVWXYZ]/  
  to mean “any capital letter”). In cases where there is a well-defined sequence associated with a set of characters, the brackets can be used with the dash (-) to specify any one character in a **range**.

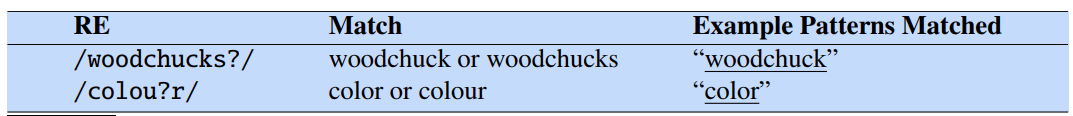


* 2.3 The square braces can also be used to specify what a single character *cannot* be, by use of the caret ^

NOTE: If it occurs anywhere else, it usually stands for a caret.



* 2.4 The question mark as meaning “zero or one instances of the previous character



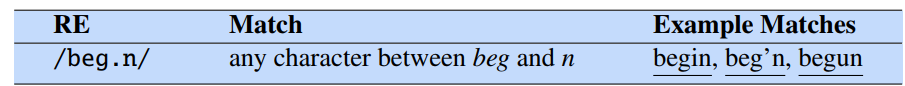
* 2.5 The Kleene star means “zero or more occurrences of the immediately previous character or regular expression

| RE | Match | Example pattern matched |
| --- | --- | --- |
| /a\*/ | Any string of zero or more ‘a’s | a or aaaa or XYZ because XYZ has zero ‘a’s |
| /[ab]\*/ | “zero or more *a*’s or *b*’s” (not “zero or more right square braces”) | *aaaa* or *ababab* or *bbbb*  or XYZ |

* 2.6 The **Kleene +**, which means “one or more occurrences of the immediately preceding  
  character or regular expression”.

| RE | Match | Example pattern matched |
| --- | --- | --- |
| /[0-9]+/ | A sequence of digits | 1 or 12 or 123456789 |

* 2.7 A **wildcard** expression that matches any single character (*except* a carriage return)



| /aardvark.\*aardvark/ | *aardvark*, appears twice | ‘*aardvark and aardvark’ or ‘aardvark abcdef aardvark’* |
| --- | --- | --- |

* 2.8 **Anchors** are special characters that anchor regular expressions to particular places in a string

| RE | Match | Example pattern matched |
| --- | --- | --- |
| ^ -> /^The/ | The only at the start of line | The boy is reading this document |
| /$ -> /^The dog \.$/ | A line that contains only the phrase The dog. | *The dog.* |
| \b -> /\bthe\b/ | \* | *Word ‘ the ’ not word ‘other’* |
| \B -> /\BX\B/ | Any X in word | *Abcxyz not ‘ xyz’* |

NOTE:

+ Word boundary: is not a digit, underscore, or letter. Ex: space, !, +, -

+ The caret ^ has three uses:

* To match the start of a line (2.8)
* To indicate a negation inside of square brackets (2.3)
* Just to mean a caret.

1. **Disjunction, Grouping, and Precedence**

* 3.1 Disjunction

Since we can’t use the square brackets to search for “cat or dog” (whycan’t we say /[catdog]/?) (The PIPE symbol)

| RE | Match | Example pattern matched |
| --- | --- | --- |
| | -> /cat|dog/ | Either string cat or dog | Cat and dog or ‘cat’ or ‘dog’ |

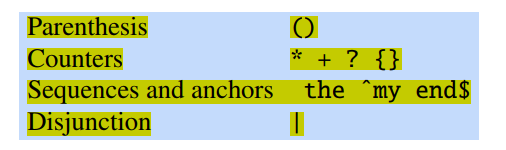
* 3.2 Precedence

How can I specify both *guppy* and *guppies*? We cannot simply say /guppy|ies/, because that would match only the strings *guppy* and *ies*.

So operators ( and ) is enclosing a pattern in parentheses makes it act like a single character for the purposes of neighboring operators like the pipe | and the Kleene\*

| RE | Match | Example pattern matched |
| --- | --- | --- |
| ( and ) -> /gupp(y|ies) / | Any string has guppy or guppies | How many live [***guppies***](https://tr-ex.me/d%E1%BB%8Bch/ti%E1%BA%BFng+anh-ti%E1%BA%BFng+vi%E1%BB%87t/guppies) in the aquarium |
| /(Column [0-9]+ \*)\*/ not /Column [0-9]+ \*/ | Match the word *Column*, followed by a number and optional spaces, the whole pattern repeated zero or more times | *Column 1 Column 2 Column 3 or*  *Column 1 Column 2 Column 3* |

* 3.3 The following table gives the order of RE operator precedence, from highest precedence to lowest precedence.



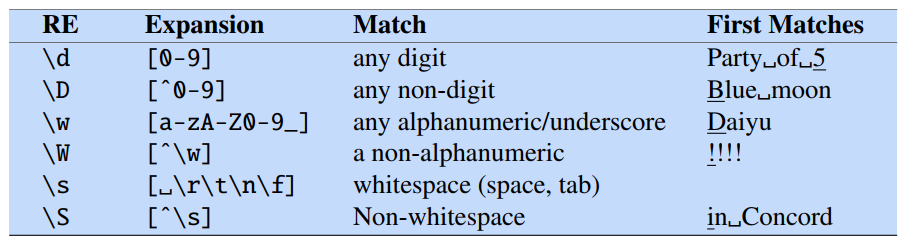
| counters have a higher precedence than sequences | /the\*/ matches *theeeee* but not *thethe* |
| --- | --- |
| sequences have a higher precedence than disjunction | /the|any/ matches *the* or *any* but not *thany* or *theny* |
|  |  |

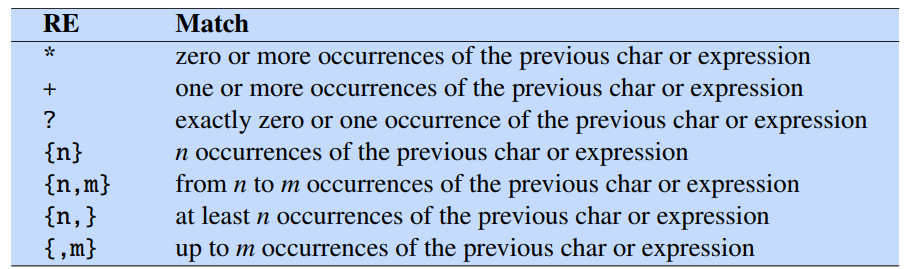
Simple example:

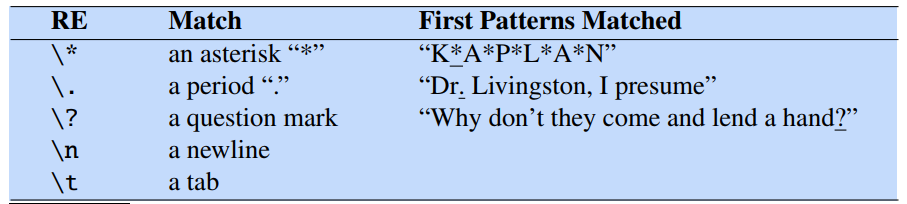
Suppose we wanted to write a RE to find cases of the English article *the*. A simple  
(but incorrect) pattern might be:  
/the/  
One problem is that this pattern will miss the word when it begins a sentence and  
hence is capitalized (i.e., *The*). This might lead us to the following pattern:  
/[tT]he/  
But we will still incorrectly return texts with the embedded in other words (e.g.,  
*other* or *theology*). So we need to specify that we want instances with a word boundary on both sides:  
/\b[tT]he\b/  
Suppose we wanted to do this without the use of /\b/. We might want this since  
/\b/ won’t treat underscores and numbers as word boundaries; but we might want  
to find *the* in some context where it might also have underlines or numbers nearby  
(*the* or *the25*). We need to specify that we want instances in which there are no  
alphabetic letters on either side of the *the*:  
/[^a-zA-Z][tT]he[^a-zA-Z]/  
But there is still one more problem with this pattern: it won’t find the word *the*when it begins a line. This is because the regular expression [^a-zA-Z], which  
we used to avoid embedded instances of *the*, implies that there must be some single  
(although non-alphabetic) character before the *the*. We can avoid this by specifying that before the *the* we require *either* the beginning-of-line or a non-alphabetic  
character, and the same at the end of the line:

/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/

1. More operators







SUMMARY

Character classes:

| . | any character except newline |
| --- | --- |
| \w\d\s | word, digit, whitespace |
| \W\D\S | not word, digit, whitespace |
| [abc] | any of a, b, or c |
| [^abc] | not a, b, or c |
| [a-g] | character between a & g |

Anchors:

| ^abc$ | start / end of the string |
| --- | --- |
| \b \B | word, not-word boundary |

Escaped characters:

| \. \\* \\ | Escaped special characters |
| --- | --- |
| \t \n \r | tab, feed line, carriage return |

Group & Lookaround:

| (abc) | capture group |
| --- | --- |
| \1 | Backreference to group #1 |
| (?:abc) | Non-capturing group |
| (?=abc) | Positive lookahead |
| (?!abc) | Negative lookahead |

Quantifiers & Alternation:

| a\* a+ a? | 0 or more, 1 or more, 0 or 1 |
| --- | --- |
| a{5} a{2,} | Exactly five, two or more |
| a{1,3} | Between one & three |
| a+? a{2,}? | Match as few as possible |
| ab|cd | Match ab or cd |

Khun content thuyết trình:

Parenthesis - Dấu ngoặc đơn

Disjunction - Dấu gạch dọc

What are Regular Expression

--------> Used for matching,searching and replace

What is "Matches"

a text matches a regular expression if it is correctly

described by the regex

"car" matches "car"

"car" also matches the first three letters in "cartoon"

"car" does not match "c\_a\_r"

Case-sensitive (by default): "car" does not match "Car"

Trong basic regular

The wildcard Character

Escaping Metacharacter

Other special characters

Tabs: \t

Line returns: \r (line return), \n (newline), \r\n

Unicode codes: \u00A9

ASCII codes: \x00A9

Character sets

Character ranges

Shorthand character sets

Negative Character sets

Nằm trong phần Advanced

Repetition

Quantified Repetition

Greedy Expressions

Greedy Expressions