Regular Expressions

BC-COMS2710 Computational Text Analysis
Summer A

Motivation

- Searching for text strings is clearly a vital task in text analysis
- What if we want to search for a pattern or variations of a string, rather than a single specific string?
    - misspelled words, plural/singular words, capitalized/uncapitalized, British/American spellings etc.
- Ex: variations on “hello”: “Hello”, “hellooo”, “hellooooooo”
- Regular Expressions, or RegEx are flexible patterns which allow us to specify all desired variations of a string in a single line

Without Regex

- To find all matches of “hello” with possible additional “o”’s in a string, need a for loop, multiple conditional statements

```
for i in range(len(string)):
    If ….
```

- Lot of work for a relatively simple, frequently used task

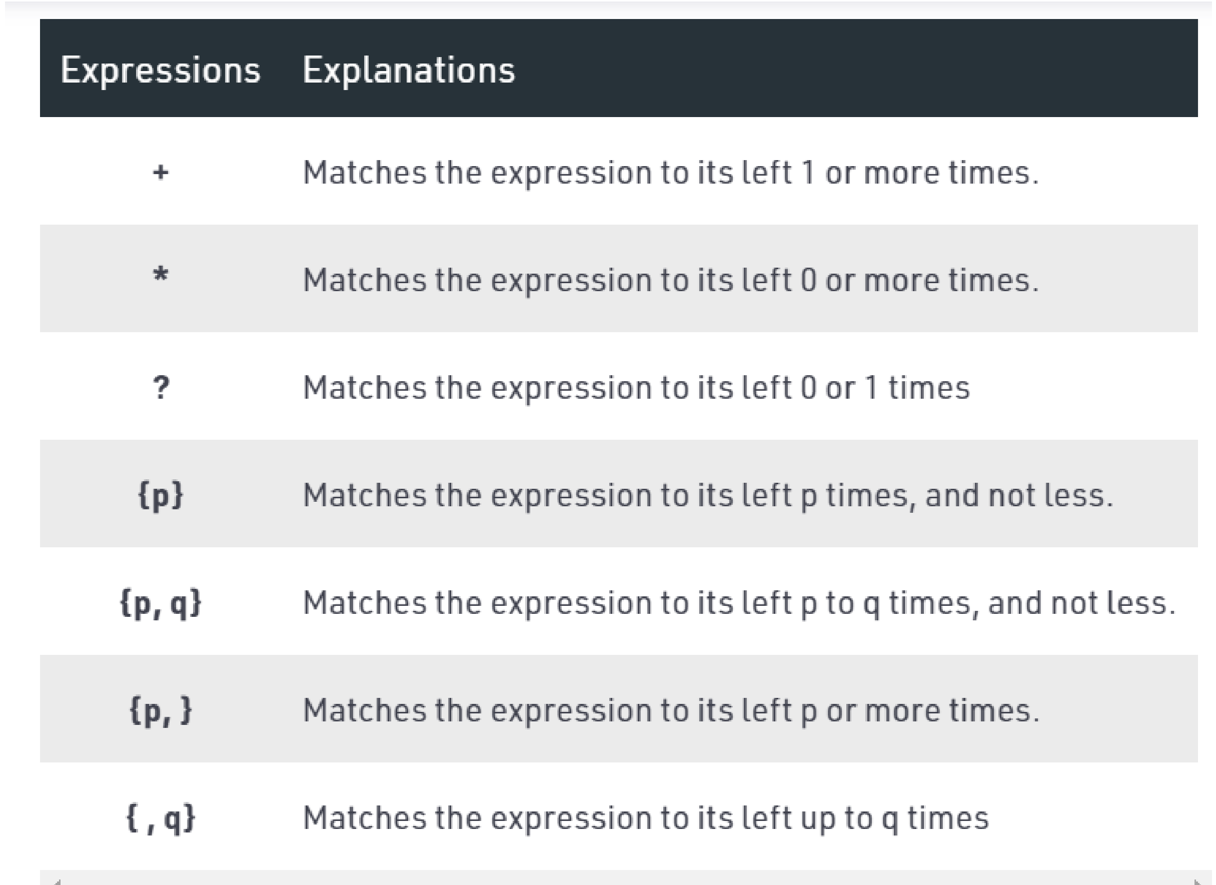
With Regex (in Python)

```
import re

txt = "hello hellloooo hellooooo helloo"

x = re.findall(r"hello+", txt)

#x will be a list of strings containing matches of the pattern in the text
```

RegEx Examples (Quantifiers)

- hello+
    - Matches “hello”, “helloo”, “hellooo” etc.
- cooo*l
    - Matches “cool”, “coool”, “cooool”, etc.

Quantifiers



Expressions	Explanations
+	Matches the expression to its left 1 or more times.
*	Matches the expression to its left 0 or more times.
?	Matches the expression to its left 0 or 1 times
{p}	Matches the expression to its left p times, and not less.
{p, q}	Matches the expression to its left p to q times, and not less.
{p, }	Matches the expression to its left p or more times.
{ , q}	Matches the expression to its left up to q times

Exercises + Cheatsheet

Cheatsheet: https://www.debuggex.com/cheatsheet/regex/python

Kahoot: www.kahoot.it

1. Quantifiers Exercise

Which regex would NOT match all words beginning with ‘a’ and are followed by at least one ‘n’ ? (Ex. “annual”, “antique”)

A. an+
B. an*
C. an{1,}

1. Quantifiers Exercise Solution

Which regex would NOT match all words beginning with ‘a’ and are followed by at least one ‘n’ ? (Ex. “annual”, “antique”)

A. an+
B. an*
C. an{1,}

Explanation: The answer is B since the regex an* would also match words beginning with ‘a’ but having 0 ‘n’s following it. Answer C is equivalent to answer A.

Character Classes (Sets)

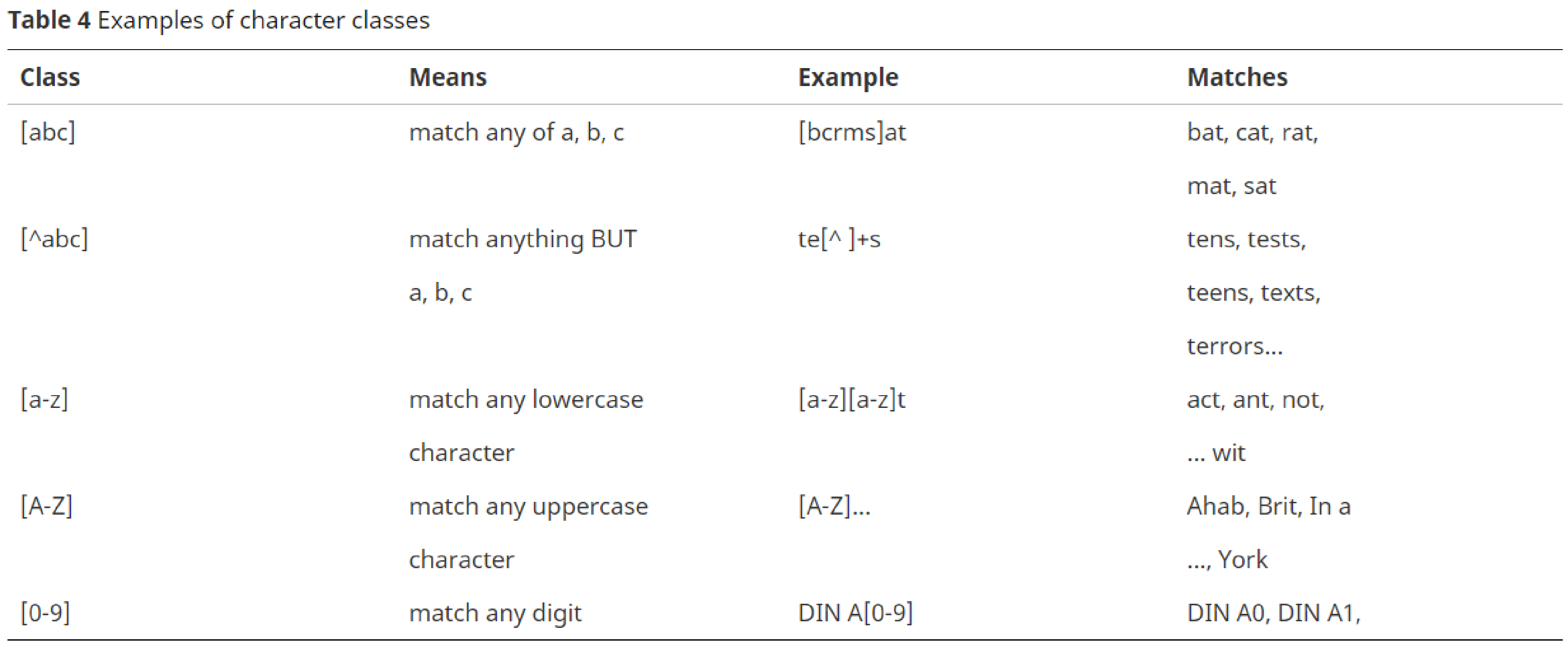


Table 4 Examples of character classes

Class	Means	Example	Matches
[abc]	match any of a, b, c	[bcrms]at	bat, cat, rat, mat, sat
[^abc]	match anything BUT a, b, c	te[^]+s	tens, tests, teens, texts, terrors...
[a-z]	match any lowercase character	[a-z][a-z]t	act, ant, not, ... wit
[A-Z]	match any uppercase character	[A-Z]...	Ahab, Brit, In a ..., York
[0-9]	match any digit	DIN A[0-9]	DIN A0, DIN A1,

Character Classes-

- Can make custom ranges using subsets of alphanumeric characters
- Ex. [a-m], [0-5]
- Ex. [ab-e] == [abcde]

2. Characters Exercise

Which regex would match all words that rhyme with “mouse”?

A. [a-z]ouse
B. [^m]ouse
C. [a-z]+ouse

2. Characters Exercise Solution

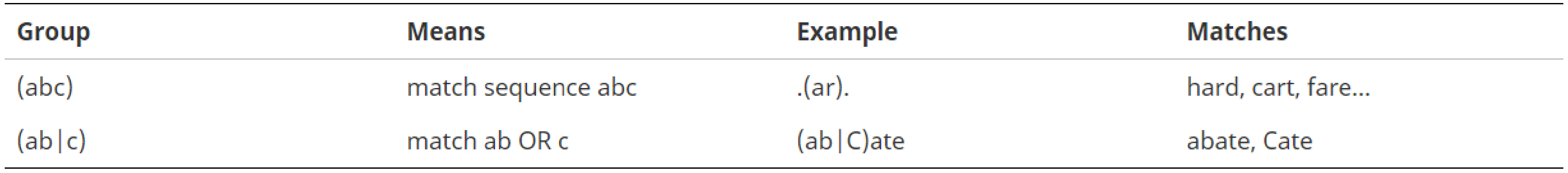
Which regex would match all words that rhyme with “mouse”?

A. [a-z]ouse
B. [^m]ouse
C. [a-z]+ouse

Explanation: The answer is C since there may be more than one preceding character to “ouse”, which the ‘+’ accounts for. B is incorrect because there are non alphabetical characters which it would match.

Groups

Table 5 Examples of groups



Group	Means	Example	Matches
(abc)	match sequence abc	.(ar).	hard, cart, fare...
(ab|c)	match ab OR c	(ab|C)ate	abate, Cate

3. Groups Exercise

Which regex would match with all words rhyming with mouse, but not including mouse?

A. [a-l | n-p]+ouse
B. [^m]ouse
C. [^m](ouse)

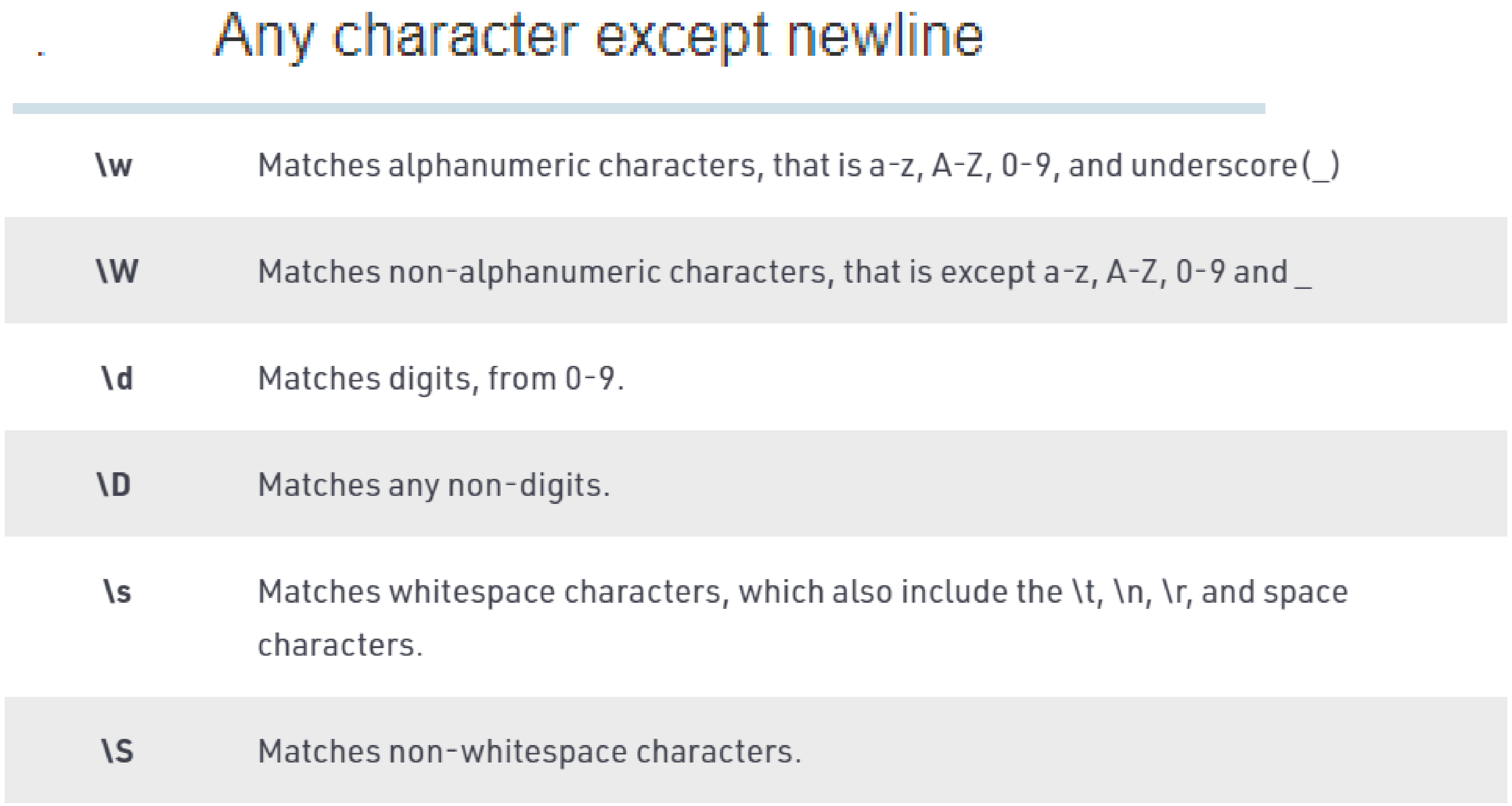
3. Groups Exercise Solution

Which regex would match all words rhyming with mouse, but not including mouse?

A. [a-l | n-p]+ouse
B. [^m]ouse
C. [^m](ouse)

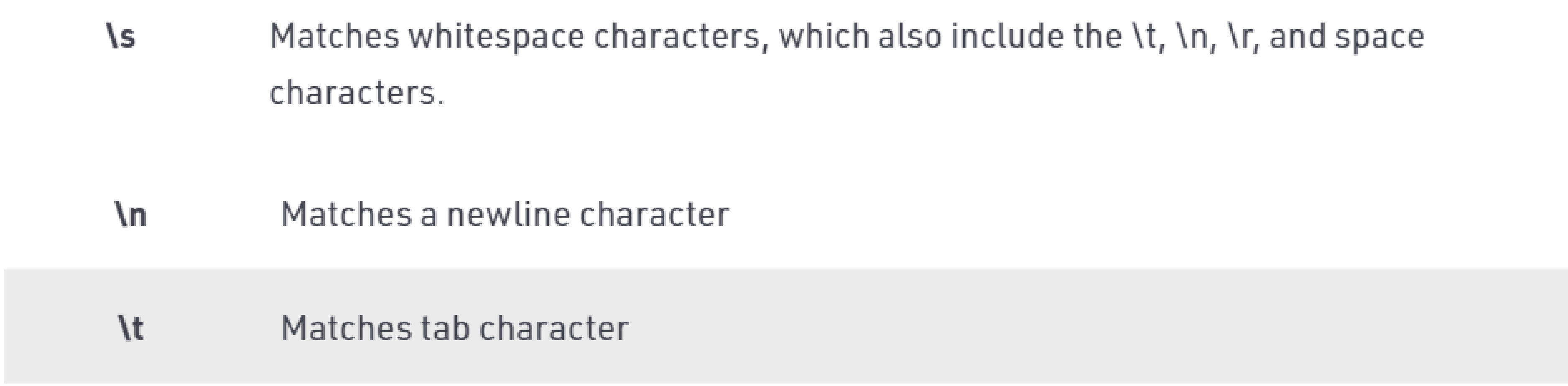
Explanation: The answer is technically A, since B and C could match non alphabetic first characters. B and C are equivalent.

Special Characters: Shorthand Character Classes



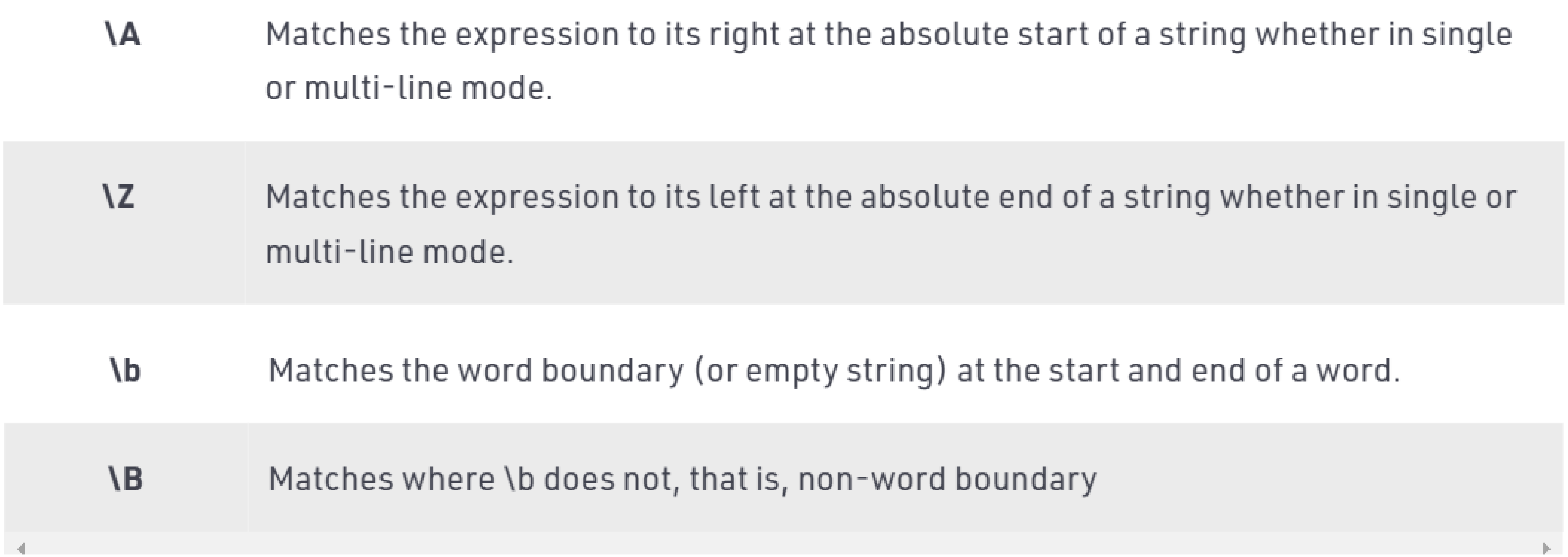
.	Any character except newline
\w	Matches alphanumeric characters, that is a-z, A-Z, 0-9, and underscore(_)
\W	Matches non-alphanumeric characters, that is except a-z, A-Z, 0-9 and _
\d	Matches digits, from 0-9.
\D	Matches any non-digits.
\s	Matches whitespace characters, which also include the \t, \n, \r, and space characters.
\S	Matches non-whitespace characters.

Special Characters: Whitespace



\s	Matches whitespace characters, which also include the \t, \n, \r, and space characters.
\n	Matches a newline character
\t	Matches tab character

Special Characters: “Empty” Strings



\A	Matches the expression to its right at the absolute start of a string whether in single or multi-line mode.
\Z	Matches the expression to its left at the absolute end of a string whether in single or multi-line mode.
\b	Matches the word boundary (or empty string) at the start and end of a word.
\B	Matches where \b does not, that is, non-word boundary

4. Special Characters Exercise

Which regex would match only distinct words rhyming with mouse (including mouse)?

A. [a-z]+ouse
B. [a-z]+ouse\b
C. [.]+ouse\b

4. Special Characters Exercise Solution

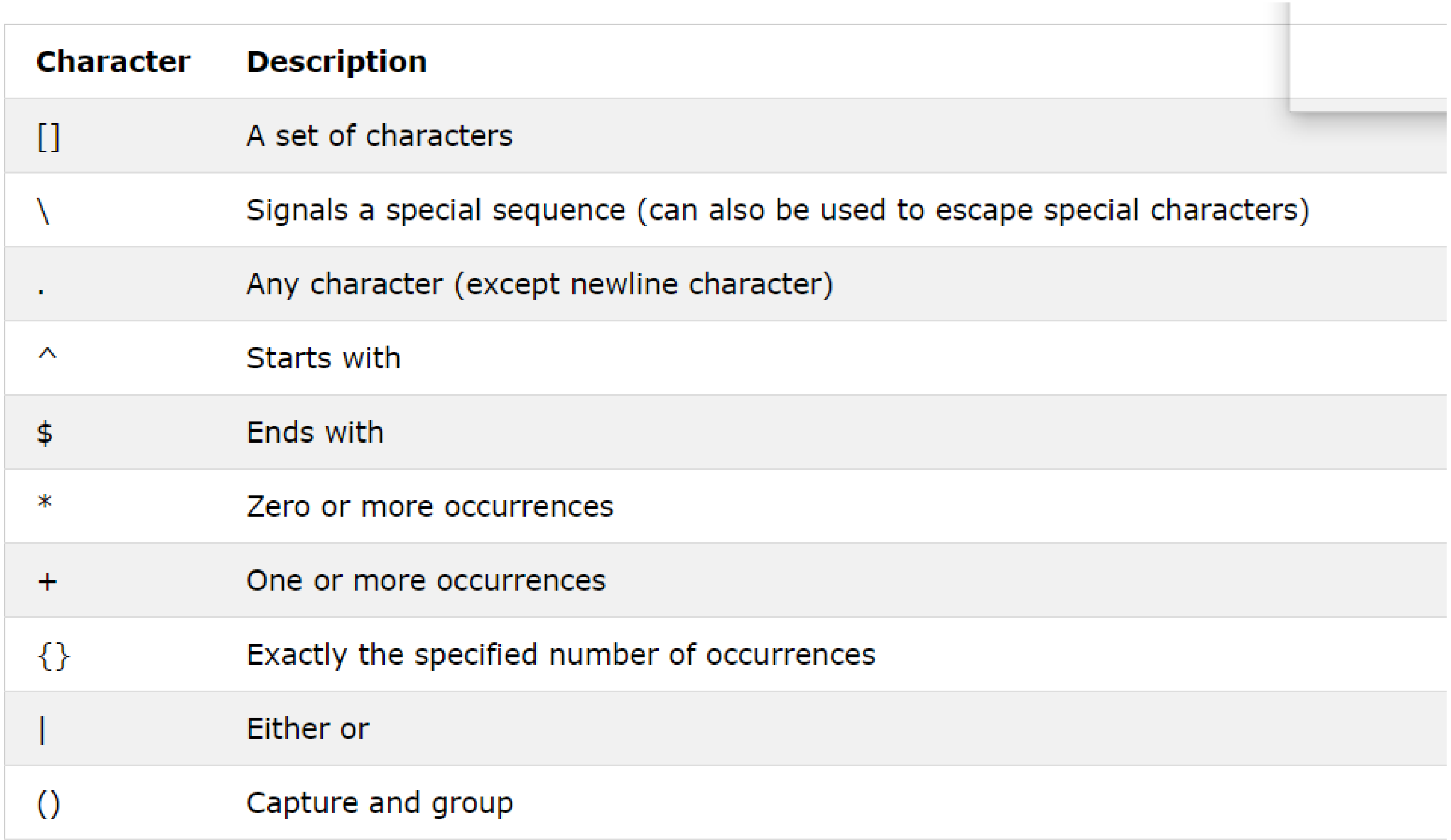
(Same question as before)

Which regex would match only distinct words rhyming with mouse, (including mouse)?

A. [a-z]+ouse
B. [a-z]+ouse\b
C. [.]+ouse\b

Explanation: B is correct because the word boundary ‘\b’ ensures that the word ends in “ouse” and [a-z] ensures that the preceding characters are alphabetical.

Metacharacters



Character	Description
[]	A set of characters
\	Signals a special sequence (can also be used to escape special characters)
.	Any character (except newline character)
^	Starts with
$	Ends with
*	Zero or more occurrences
+	One or more occurrences
{}	Exactly the specified number of occurrences
|	Either or
()	Capture and group

Escape Character

- The escape character is “\“
- Must be used for specifying metacharacters
- Example:
  - to match “^”, regex is “\^”
  - To match “\t” (but not tab character), regex is “\\t”
  - And to match “ (quotation mark), regex is “\””
- Exception: inside a set [] or a group (), metacharacters are literals
  - In regex “(+*?)”, no need for escape characters to specify + * ?

5. Metacharacters Exercise

Which regex would match all sentences ending with the word ‘farewell’ (assuming all sentences end in a period) ?

A. farewell.
B. \sfarewell.
C. \bfarewell\.

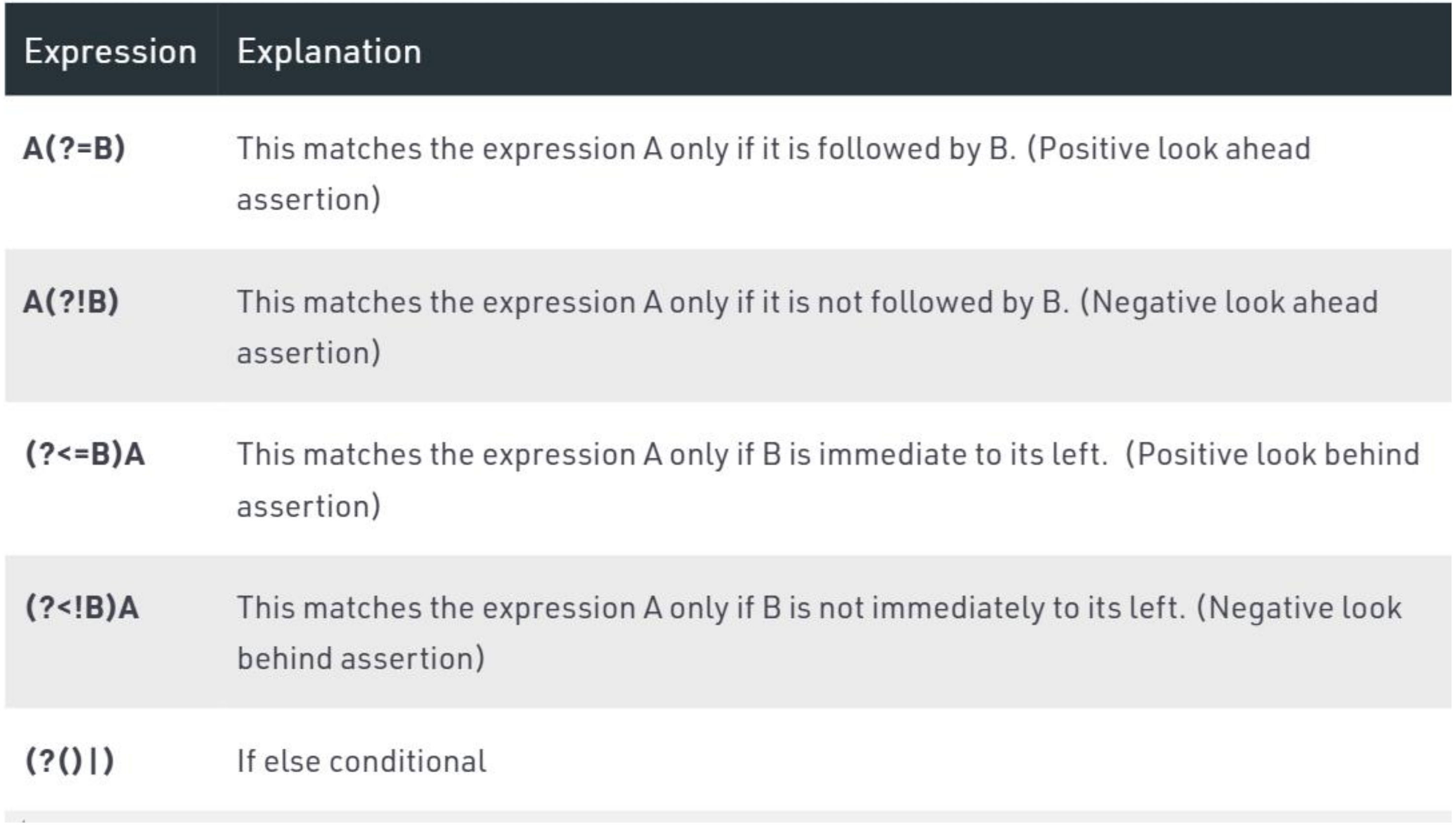
5. Metacharacters Exercise Solution

Which regex would match all sentences ending with the word “farewell” (assuming all sentences end in a period) ?

A. farewell.
B. \sfarewell.
C. \bfarewell\.

Explanation: C is correct because the empty space ‘\s’ ensures that it is in fact the last word, and the escape character before the period ensures that it is read as a period in the regex.

Lookarounds (Assertions)



Expression	Explanation
A(?=B)	This matches the expression A only if it is followed by B. (Positive look ahead assertion)
A(?!B)	This matches the expression A only if it is not followed by B. (Negative look ahead assertion)
(?<=B)A	This matches the expression A only if B is immediate to its left. (Positive look behind assertion)
(?<!B)A	This matches the expression A only if B is not immediately to its left. (Negative look behind assertion)
(?()|)	If else conditional

6. Lookarounds Exercise

Which regex would match all names with the title “Dr. ” ? e.g. “Dr. Livingston”

A. (?=Dr.).+\b\s
B. (?<=Dr\.).+\b\s
C. (?=Dr\.).+\b\s

6. Lookarounds Exercise Solution

Which regex would match all names (with titles) where the title is “Dr. ” ? e.g. “Dr. Livingstone”

A. (?=Dr.).+\b\s
B. (?<=Dr\.).+\b\s
C. (?=Dr\.).+\b\s

Explanation: The (?<=) expression will return matches only for where the immediately preceding characters are “Dr\. “

B is therefore incorrect because it will only get the name, not the title.

A is incorrect because it does not include the escape character for the period.

More Cheatsheets/References

https://learnbyexample.github.io/python-regex-cheatsheet/

https://pythex.org/ --Regex checker

https://www.geeksforgeeks.org/python-regex-cheat-sheet/

Limitations of Regex in Application

- You have to already know what you’re looking for
  - be familiar with the text prior to using regex
- Easy to get false positives or false negatives of desired result
- Can get complicated and lose readability
- Can get computationally expensive
  - Every language/library is different in terms of optimization
- Best practice:
  - use for simple patterns (more than just a substring)
  - test expected results, desired positives and negatives before hand
  - Thoroughly validate results

Sources

- Computational Text Analysis in Python Ch. 3
- Jurafsky slides
- https://www.geeksforgeeks.org/python-regex-cheat-sheet/
- https://learnbyexample.github.io/python-regex-cheatsheet/