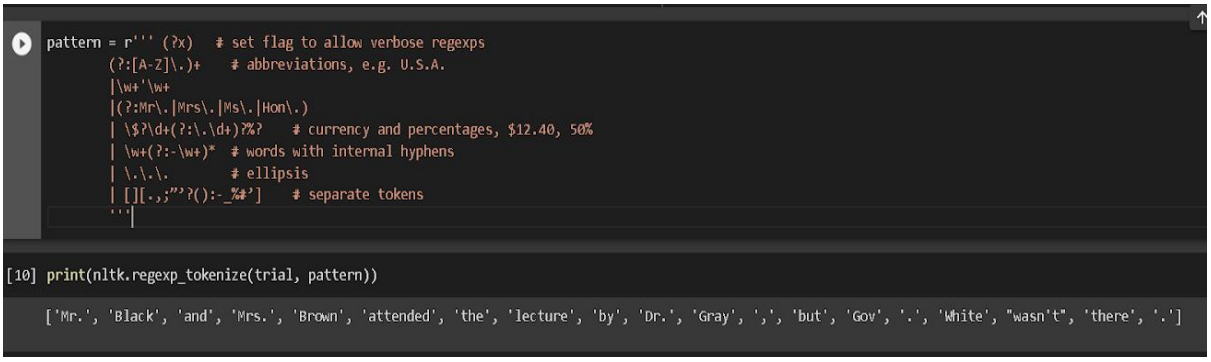
CIS 668 MOO2 SPRING21 NATURAL LANGUAGE PROCESSING LAB 3:

Result screenshots:

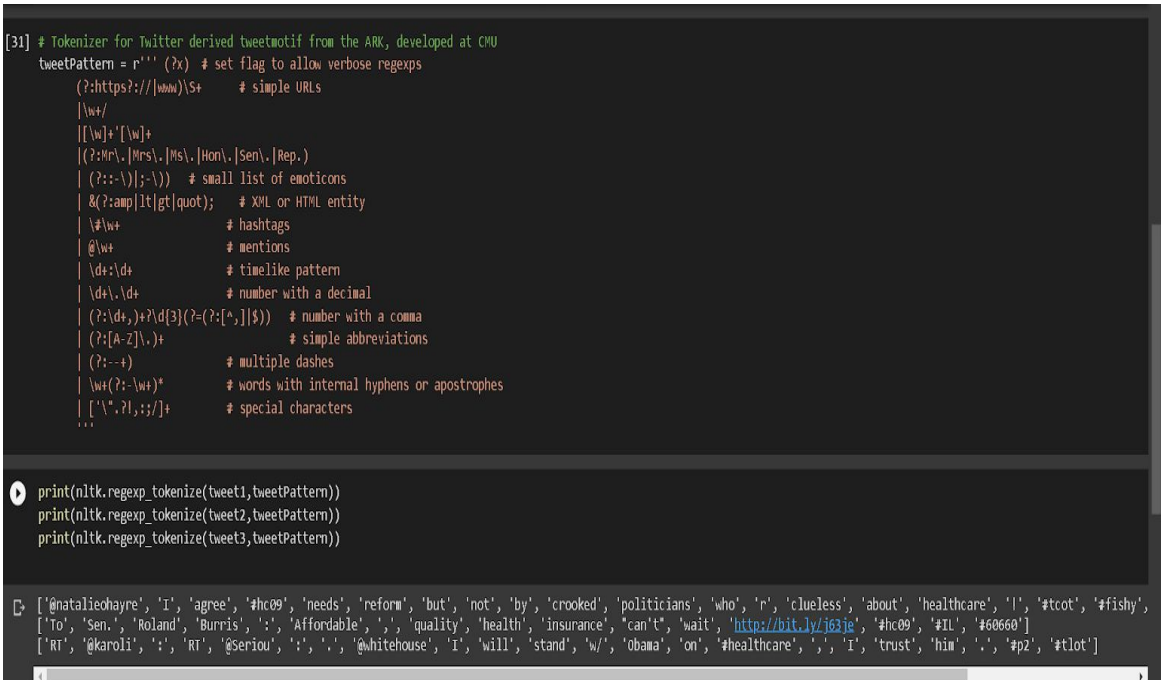


```
pattern = r''' (?x)   # set flag to allow verbose regexps
        (?:[A-Z]\.)+    # abbreviations, e.g. U.S.A.
        |\w+'\w+
        |(?:Mr\.|Mrs\.|Ms\.|Hon\.)
        | \$?\d+(?:\.\d+)?%?    # currency and percentages, $12.40, 50%
        | \w+(?:-\w+)*  # words with internal hyphens
        | \.\.\.        # ellipsis
        | [][.,;"'?():-_%#']    # separate tokens
        '''

[10] print(nltk.regexp_tokenize(trial, pattern))

    ['Mr.', 'Black', 'and', 'Mrs.', 'Brown', 'attended', 'the', 'lecture', 'by', 'Dr.', 'Gray', ',', 'but', 'Gov', '.', 'White', "wasn't", 'there', '.']
```

As seen in the above image 'Mr.' is taken as a single token and dot is included ,' wasn't ' was also taken as a single token.



```
[31] # Tokenizer for Twitter derived tweetmotif from the ARK, developed at CMU
    tweetPattern = r''' (?x)  # set flag to allow verbose regexps
        (?:https?://|www)\S+     # simple URLs
        |\w+/
        |[\w]+'[\w]+
        |(?:Mr\.|Mrs\.|Ms\.|Hon\.|Sen\.|Rep.)
        | (?::-\)|;-\))  # small list of emoticons
        | &(?:amp|lt|gt|quot);    # XML or HTML entity
        | \#\w+                # hashtags
        | @\w+                 # mentions
        | \d+:\d+              # timelike pattern
        | \d+\.\d+             # number with a decimal
        | (?:\d+,)+?\d{3}(?=(?:[^,]|$))  # number with a comma
        | (?:[A-Z]\.)+                   # simple abbreviations
        | (?:--+)              # multiple dashes
        | \w+(?:-\w+)*         # words with internal hyphens or apostrophes
        | ['\".?!,:;/]+        # special characters
        '''

print(nltk.regexp_tokenize(tweet1,tweetPattern))
print(nltk.regexp_tokenize(tweet2,tweetPattern))
print(nltk.regexp_tokenize(tweet3,tweetPattern))

['@natalieohayre', 'I', 'agree', '#hc09', 'needs', 'reform', 'but', 'not', 'by', 'crooked', 'politicians', 'who', 'r', 'clueless', 'about', 'healthcare', '!', '#tcot', '#fishy',
['To', 'Sen.', 'Roland', 'Burris', ':', 'Affordable', ',', 'quality', 'health', 'insurance', "can't", 'wait', 'http://bit.ly/j63je', '#hc09', '#IL', '#60660']
['RT', '@karoli', ':', 'RT', '@seriou', ':', '.', '@whitehouse', 'I', 'will', 'stand', 'w/', 'Obama', 'on', '#healthcare', ',', 'I', 'trust', 'him', '.', '#p2', '#tlot']
```

As seen in the above image from tweet 2 'Sen.' is taken as a single token ,from tweet 3 'w/' is recognized as a token ,from tweet 2 ' can't ' is considered as a single token.

OBSERVATIONS:

1. **"\w+"** is a regular expression which matches a letter or a group of letters, in this case we need to find the words which have an apostrophe in them so we take an expression **(\w+'\w+)**
2. **(?:Mr\.|Mrs\.|Ms\.|Hon\.)** in this regular expression we directly match the words with an escape character dot
3. As we require the words which ends with a **‘ / ’** character and we have no requirement for the token or words in which the **‘ / ’** appears in the middle, so we don't use regular expression \w+/ which matches one or more words
4. We can mould and fabricate Regular expressions as per our problem requirement and same regular expressions can be used throughout our solution

Lesson Learned:

1. Pattern matching requires the key structure of regular expressions
2. As per the problem statement requirement regular expression can be put to use to create a custom tokenizer
3. According to my observation, the nltk function regexp_tokeinze uses the regular expression on text by individually applying each regular expression on text so as to find any matches to the token
4. In regular tokenizer internal special characters are recognized as different characters but one can make a customized tokenizer to take them as one single token