# CSCI 8450 Assignment: Chapter 3a

*Mohan Sai Ambati*

1. **(b)**

   The regular expression [A-Z][a-z]* denotes “all words which has alphabets from a to z and first letter should always starts with upper case and from second letter there can be zero or infinite alphabets only in lower case”.

   The words that match the regular expression in given SimpleText is surrounded by curly braces shown below:

   {One} day, his horse ran away. {The} neighbors came to express their concern: "{Oh}, that's too bad. {How} are you going to work the fields now?" {The} farmer replied: "{Good} thing, {Bad} thing, {Who} knows?" {In} a few days, his horse came back and brought another horse with her. {Now}, the neighbors were glad: "{Oh}, how lucky! {Now} you can do twice as much work as before!" {The} farmer replied: "{Good} thing, {Bad} thing, {Who} knows?"

   **(c)**

   The regular expression p[aeiou]{,2}t denotes “ words that start with p as 1st letter and any alphabet among (a, e, i, o, u) as second letter and followed by {,2}t string”.

   If in place of {,2} if there is {2,} it means that there can be 2 or more letters from set (a, e, i, o, u) but {,2} doesn’t have any predefined meaning in regular expression syntax.

   The string that match above regular expression is:

   pa{,2}t, pi{,2}t….. (No such words are present in SimpleText.)

   **(f)**

   The regular expression \w+|[^\\w\s]+ denotes “ words that has at least one alphanumeric or _ character in it and leave the space after that word”.

   Every word in the Simple Text matches with this regular expression

   {One} {day}{,} {his} {horse} {ran} {away}{.} {The} {neighbors} {came} {to} {express} {their} {concern}{:} {"}{Oh}{,} {that}{'}{s} {too} {bad}{.} {How} {are} {you} {going} {to} {work} {the} {fields} {now}{?"} {The} {farmer} {replied}{:} {"}{Good} {thing}{,} {Bad} {thing}{,} {Who} {knows}{?"} {In} {a} {few} {days}{,} {his} {horse} {came} {back} {and} {brought} {another} {horse} {with} {her}{.} {Now}{,} {the} {neighbors} {were} {glad}{:} {"}{Oh}{,} {how} {lucky}{!} {Now} {you} {can} {do} {twice} {as} {much} {work} {as} {before}{!"} {The} {farmer} {replied}{:} {"}{Good} {thing}{,} {Bad} {thing}{,} {Who} {knows}{?"}

2. Given single determiners are (a, an, the)

   These can occur in start of the sentence or in the rest of the sentence. So, the regular expression is as follows:

   \b[Aa][n]?\b|\b[Tt][h][e]\b

   The words that match above regular expression in SimpleText is surrounded by curly braces:

   One day, his horse ran away. {The} neighbors came to express their concern: "Oh, that's too bad. How are you going to work {the} fields now?" {The} farmer replied: "Good thing, Bad thing, Who knows?" In {a} few days, his horse came back and brought another horse with her. Now, {the} neighbors were glad: "Oh, how lucky! Now you can do twice as much work as before!" {The} farmer replied: "Good thing, Bad thing, Who knows?"

3. In this exercise I have taken the words which contains at least one lower case letter in it from the provided html file.

**Difference between result of re.findall() and word_tokenize():** The main differences between the result of regular expression and word_tokenize() method is listed below:

- word_tokenize() have upper case words but regular expression have mixed and lower case words.
- word_tokenize() have any alphanumeric characters in it whereas regular expression result has only English words.
- Combination of words such as ‘fashion-dependent’ are considered as two sperate words in regular expression, but it is considered as single word in wor_tokenize().

**Words in re.findall() and not in word_tokenize()** : {'multimedia', 'rd', 'professor', 'll', 'large', 'visual', 'wide', 'm', 'high', 't', 'don', 'industrial', 's', 'audio', 'Statesman', 'American', 'dependent', 'post', 'ill', 'communication', 'haven', 'inundated', 'fashion', 'tenure', 'directed', 'tech', 'cannot'}

**Words in word_tokenize() and not in re.findall() :** {'?', 'fashion-dependent', '1996', ';', ')', "'ll", '8', 'ill-directed', 'world-wide', 'post-tenure', 'H.', 'BC', '25', '1575', "n't", "nation's", 'I.', 'DDR', 'audio-visual', '1574', 'I', 'UT', '212', 'TV', '3rd', '$', '90s', 'world-at-large', ':', '27', 'academic/industrial', "professor's", '415', '.', 'information-inundated', 'multimedia/communication', '...', 'WW', 'so-called', '(', '""', 'A', '``', '80s', '399', '--', '34', 'knowledge-based', '80', 'American-Statesman', 'II', '70s', 'high-tech', "'s", ',', 'AD', "'m", 'well-kept'}

**Using nltk.WordNetLemmatizer() :**By default WordNetLemmatizer() attempts to find the closest “noun” (stem) for any word. Lemmatize function finds the stem of every word, as a result if we have words having similar stem set() function removes the duplicates and overall word count will be reduced. As a result, the unknown words reduced from 205 in regular expression to 139 after lemmatizing.

Unknown words (unknown_lemmetized_words, unknown_refindall()_words) : (139, 205)

**4.** English to Pig Latin has 3 basic rules:

- For words that begin with consonant cluster, remove consonant cluster before the initial vowel and append it to end of word and append ‘ay’.
- For words that start with vowel, append ‘ay’ to its end.
- Few words even though they start with consonant cluster they have a vowel sound in it. Identify the vowel sounds and remove letters before it and append to its end with ‘ay’. Ex: style – here letter ‘y’ has ‘i’ sound

First and second steps are straight forward. To implement third step, I have taken a list of consonant clusters that can possibly have vowel sound in it and removed it from start and appended to last with ‘ay’.

**SimpleText in Pig Latin:**

Oneay ayday , ishay orsehay anray awayay . eThay eighborsnay amecay otay expressay eirthay oncerncay : `` Ohay , atthay 'say ootay adbay . owHay areay ouyay oinggay otay orkway ethay ieldsfay ownay ? " eThay armerfay epliedray : `` oodGay ingthay , adBay ingthay , oWhay owsknay ? " Inay aay ewfay aysday , ishay orsehay amecay ackbay anday oughtbray anotheray orsehay ithway erhay . owNay , ethay eighborsnay ereway adglay : `` Ohay , owhay uckylay ! owNay ouyay ancay oday icetway asay uchmay orkway asay eforebay ! " eThay armerfay epliedray : `` oodGay ingthay , adBay ingthay , oWhay owsknay ? "