Part I

Problem 21

Write a function `unknown()` that takes a URL as its argument, and returns a list of unknown words that occur on that webpage. In order to do this, extract all substrings consisting of lowercase letters (using `re.findall()`) and remove any items from this set that occur in the Words Corpus (`nltk.corpus.words`). Try to categorize these words manually and discuss your findings.

Code:

```
def calc_unknown(url):
   ssl._create_default_https_context =
      ssl._create_unverified_context
   html = request.urlopen(url).read().decode('utf8')
   raw = BeautifulSoup(html).get_text()
   tokens = [w for (_,w) in re.findall(r'\b[a-z]+\b', raw)]
   known_words = [w for w in nltk.corpus.words.words('en') if
      w.islower()]
   unknown = list(set(tokens).difference(set(known_words)))

   print("Unknown words: %s" % unknown)
   print("Length of raw response: %d" % len(raw))
   print("Number of unknown words in of raw response: %d" %
      len(set(unknown)))
```

Output:

```
$ python3 unknown.py https://www.nltk.org/book/ch03.html
[ 'occurrences',
    'doyouseethekittyseethedoggydoyoulikethekittylikethedoggy',
    'interfaces', 'systems', 'terytorium', 'eql',
    'lolcatbible', 'messages', 'webpage', 'fractions',
    'metals', 'kjv', 'upenn', 'selecting', 'quotes',
    'colgroup', 'ed', 'duplicated', 'rules', 'lists',
    'describing', 'znane', 'involves', 'amp', 'abstracts',
    'infuriating', 'exercising', 'placeholder', 'ig', 'has',
    'frequencies', 'vowels', 'netscape', 'problems', 'lines',
    'failed', 'cortisol', 'fonts', 'ceremoni', 'hmn',
    'distributing', 'alphanumeric', 'cvs', 'regexp', 'thead',
    'email', 'forms', 'specifies', 'filename', 'whch',
    'constructed', 'represented', 'changes', 'kultury',
    'ffffff', 'acts', 'measures', 'discussions', 'mapped',
    'others', 'elements', 'treebank', 'oriented',
    'determining', 'whitespace', 'blog', 'su', 'scores',
    'doyouliketh', 'distributions', 'ones', 'hyphens',
    'variables', 'suggested', 'seethekit', 'representations',
    'borrowed', 'nwell', 'things', 'lengths', 'haaa', 'doy',
    'dates', 'occurring', 'seetheki', 'spam', 'causes',
    'phrases', 'souls', 'thedogg', 'headwords', 'closures',
    'miiiiiinnnnnnnnnnneeeeeeee', 'methods', 'spaces', 'frdm',
    'et', 'ui', 'ekittylike', 'lol', 'points', 'tries',
    'boosted', 'mapping', 'suggests', 'splitlines',
    'realized', 'settings', 'dawne', 'fdist', 'pieces',
    'toplevel', 'amounts', 'overflows', 'ahhh', 'makes',
    'engines', 'placed', 'exercises', 'gt', 'distinctions',
    'nd', 'consists', 'numeric', 'iterations', 'tokenizes',
    'published', 'notations', 'loops', 'sztuki', 'tabs',
    'oauaio', 'liquids',
    'mmmmmmmmiiiiiiiiinnnnnnnnnneeeeeeee', 'kapokapora',
    'aaaaaaaaaaaaaaaa', 'bngs', 'languagelog', 'linebreak',
    'possibilities', 'earlier', 'integers', 'demonstrated',
    'collocations', 'normalizing', 'includes', 'bigram',
    'acronyms', 'lmao', 'processe', 'facilitates',
    'eoldrnnnna', 'nazw', 'ou', 'stm', 'helps', 'hahaha',
    'trains', 'techniques', 'unicode', 'applies', 'items',
    'suffers', 'programming', 'marks', 'twisters',
    'titlecased', 'keeps', 'thursday', 'constraints',
    'ahhhh', 'ancestors', 'substitutions', 'kaapo', 'online',
    'installed', 'findall', 'inline', 'inputting',
    'apostrophes', 'ch', 'nom', 'software', 'wider', 'max',
    'nc', 'ident', 'couldn', 'doesn', 'lowercase',
    'tokenizers', 'hypernym', 'sighan', 'chooses',
    'reformatting', 'questions', 'requires', 'lexeme',
    'words', ...]
Length of raw response: 127362
Number of unknown words in of raw response: 951 ...]
```

Observations: the unknown words in Chapter 3 of the NLTK text book appear to be words related to python, html, javascript or coding in general.

A few words (specifically 'liquids', 'suggests', and other instances of plural words) surprised me when they were categorized as 'unknown', because they seem common enough to appear in the words corpus.

Problem 22

Examine the results of processing the URL http://news.bbc.co.uk/ using the regular expressions suggested above. You will see that there is still a fair amount of non-textual data there, particularly Javascript commands. You may also find that sentence breaks have not been properly preserved. Define further regular expressions that improve the extraction of text from this web page.

Results:

```
$ python3 unknown.py http://news.bbc.co.uk/
'polyfill', 'minutes', 'films', 'ensuring', 'webmodule',
    'occurred', 'cy', 'keywords', 'imrworldwide',
    'stylesheet', 'debug', 'notifications', 'noscript',
    'jquery', 'crwdcntrl', 'options', 'var', 'uid', 'items',
    'suffers', 'deploys', 'nations', 'locations',
    'morebutton', 'odnoklassniki', 'cymru', 'cf', 'alsos',
    'struggles', 'plurk', 'jssignals', 'trafalgar',
    'makeitdigital', 'gb', 'faces', 'png', 'arts',
    'statusbar', 'investigates', 'livejournal', 'stamps',
    'templates', 'beats', 'onload', 'killings', 'gsurl',
    'features', 'lotame', 'promo', 'facebook', 'charset',
    'const', 'unescape', 'dcdcdc', 'says', 'dotcom',
    'perspectives', 'bbcpage', 'fff', 'years', 'gpt',
    'scotland', 'callback', 'paths', 'waits', 'newsdotcom',
    'ltr', 'str', 'pe', 'css', 'calc', 'tags', 'styles',
    'ssc', 'keyframes', 'svg', 'gt', 'miterlimit', 'couldn',
    'arguments', 'ejected', 'footerbutton', 'eslint',
    'labels', 'menuitem', 'ul', 'northernireland', 'max',
    'usingthebbc', 'accuses', 'modules', 'bundles',
    'versions', 'html', 'cmd', 'sharedmodules',
    'bbcworldwide', 'cta', 'pinterest', 'accounts',
    'homepage', 'hopes', 'meneame', 'preferences', 'len',
    'declarations', 'lazyload', 'shows', 'flipboard', 'lx',
     'envelopes', 'src', 'hostname', 'xs', 'isn', 'substring',
     'moz', 'reactid', 'details', 'became', 'mozart', 'info',
     'determining', 'webmodules', 'cdn', 'overachiever',
     'googleplus', 'bbcprivacy', 'envs', 'subnav', 'ms',
     'warnings', 'https', 'crashed', 'adding', 'utf', 'waf',
     'istats', 'youtube', 'bbc', 'async', 'expressions',
     'weibo', 'feeds', 'edr', 'cookies', 'prefetch',
     'bbcredirection', 'placeholder']
Length of raw response: 246615
Number of unknown words in of raw response: 369
```

Observations: Most uknown words seem related to javascript libraries named by unique, coding specific words, but we still see the plurals being categorized as unknown.

I used the stemming exercises form Ch 3 to find only token stems, and compare those to the corpus of known words. There are still many unknown words related to coding, but significantly fewer unknown words overall.

New Code:

```
def stem(word):
   for suffix in ['ing', 'ly', 'ed', 'ious', 'ies', 'ive',
       'es', 's', 'ment']:
      if word.endswith(suffix):
         return word[:-len(suffix)]
   return word

def calc_unknown(url):
   ssl._create_default_https_context =
       ssl._create_unverified_context
   html = request.urlopen(url).read().decode('utf8')
   raw = BeautifulSoup(html).get_text()
   tokens = re.findall(r'\b[a-z]+\b', raw)
   known_words = [w for w in nltk.corpus.words.words('en') if
       w.islower()]
   unknown = list(set(tokens).difference(set(known_words)))
   token_stems = [stem(w) for w in tokens]
   unknown_stems =
       list(set(token_stems).difference(set(known_words)))
```

```
print("Unknown word stems: %s" % unknown_stems)
print("Length of raw response: %d" % len(raw))
print("Number of unknown words in of raw response: %d" %
    len(set(unknown)))
print("Number of unknown word stems in of raw response:
    %d" % len(set(unknown_stems)))
```

```
Unknown word stems: ['', 'pinterest', 'balatarin',
    'preferenc', 'dep', 'stor', 'var', 'png', 'advertis',
    'localnew', 'became', 'earthdotcom', 'headerbutton',
    'stat', 'cbbc', 'google', 'odnoklassniki',
    'bbcworldwide', 'explod', 'shar', 'charset', 'invok',
    'driv', 'lazyload', 'webkit', 'enhanc', 'ensur',
    'navpromo', 'blog', 'lib', 'async', 'styl', 'actres',
    'closeable', 'btn', 'dcdcdc', 'moz', 'webmodul', 'ltr',
    'synchronou', 'miterlimit', ... 'substr', 'bbcpage',
    'modul', 'docu', 'placeholder', 'meneame', 'bitesize',
    'ssc', 'ellipsi', 'gt', 'giv', 'makeitdigital', 'comp',
    'choic', 'overachiever', 'focu', 'plu', 'xxl', 'bos',
    'investigat', 'utf', 'svg', 'dialog', 'unescape',
    'usingthebbc', 'bbcuser', 'debug', 'transmitt',
    'gscontxt', 'progid', 'radiu', 'unus', 'bramstein',
    'thi', 'lx', 'const', 'nav', 'googletagservic',
    'echoclient', 'bbcprivacy', 'idcta', 'cy',
    'imrworldwide', 'cookie', 'com', 'autocorrect', 'gb',
    'cach', 'http', 'provid', 'ap', 'renren', 'hatena',
    'padd', 'disabl', 'isn', 'dai', 'pathname']
Length of raw response: 246645
Number of unknown words in of raw response: 370
Number of unknown word stems in of raw response: 305
```

Problem 29

Readability measures are used to score the reading difficulty of a text, for the purposes of selecting texts of appropriate difficulty for language learners. Let us define `w` to be the average number of letters per word, and `s` to be the average number of words per sentence, in a given text. The Automated Readability Index (ARI) of the text is defined to be: 4.71 w + 0.5 s - 21.43. Compute the ARI score for various sections of the Brown Corpus, including section f (lore) and j (learned). Make use of the fact that `nltk.corpus.brown.words()` produces a sequence of words, while `nltk.corpus.brown.sents()` produces a sequence of sentences.

Code:

```
def readability(input):
   letters = brown.raw(categories=input)
   words = brown.words(categories=input)
   sentences = brown.sents(categories=input)

   letters_per_word = len(letters) / len(words)
   words_per_sentence = len(words) / len(sentences)

   ari_score = (4.71 * float(letters_per_word)) + (0.5 *
      float(words_per_sentence)) - 21.43
   print("Letters per word: %s" % letters_per_word)
   print("Words per sentence: %s" % words_per_sentence)
   print("ARI Score: %s" % ari_score)
```

Output:

```
$ python3 readability.py lore
Letters per word: 8.60259839164453
Words per sentence: 22.59762343782012
ARI Score: 30.387050143555797
```

```
$ python3 readability.py learned
Letters per word: 8.828718771991555
Words per sentence: 23.51797258856995
ARI Score: 31.912251710365197
```

Part II

After downloading the tei-reader tool, I used their example code on one of the example texts from teibyexample.org to generate the following:

Code:

```
from tei_reader import TeiReader
reader = TeiReader()
corpora = reader.read_file('tei_example.xml') # or read_string
print(corpora.text)

# show element attributes before the actual element text
print(corpora.tostring(lambda x, text: str(list(a.key + '=' +
    a.text for a in x.attributes)) + text))
```

Output:

```
$ python3 tei.py
Review
Die Leiden des jungen Werther is an exceptionally good example
    of a book full of Weltschmerz.
[]['fileDesc::titleStmt::title=Review: an electronic
    transcription', 'fileDesc::publicationStmt=Published as an
    example for the Introduction module of TBE.',
    'fileDesc::sourceDesc=No source: born
    digital.']['tei-tag=text']['tei-tag=body']['tei-tag=head'][][]Review
['tei-tag=p', 'note= by Goethe'][]['tei-tag=title', 'note= by
    Goethe'][][]Die Leiden des jungen Werther[] is an
    ['tei-tag=emph'][][]exceptionally[] good example of a book
    full of ['tei-tag=term'][][]Weltschmerz[].
```

Part III

Identify a possible available textual source that you might use for a project and identify the possible challenges that you will face in preprocessing the text.

I think the The New Testament, The Torah, and The Quran would be interesting textual sources for one of the class projects. There are versions is freely available online in xml formats, and would not require any Copyright or License-based hoops to jump through. However, I envision the source needing some processing regarding chapters, verses, sections, books, preserving natural breaks within the physical sourcse that would allow for interesting comparisons and visualizations, much like the "Readability visualization" we saw in class two weeks ago.