Regression Test : NLTK Word Tokenizer

Tokenizing some test strings.

In [3]:
```
import os
import nltk
```

In [4]:
```
from nltk.tokenize import word_tokenize
```

In [5]:
```
s1 = "On a $50,000 mortgage of 30 years at 8 percent, the monthly payment would
s1
```

Out[5]:
```
'On a $50,000 mortgage of 30 years at 8 percent, the monthly payment would be
$366.88.'
```

In [6]:
```
word_tokenize(s1)
```

Out[6]:
```
['On',
 'a',
 '$',
 '50,000',
 'mortgage',
 'of',
 '30',
 'years',
 'at',
 '8',
 'percent',
 ',',
 'the',
 'monthly',
 'payment',
 'would',
 'be',
 '$',
 '366.88',
 '.']
```

In [7]:
```
 s2 = "\"We beat some pretty good teams to get here,\" Slocum said."
s2
```

Out[7]:
```
'"We beat some pretty good teams to get here," Slocum said.'
```

In [8]:
```
word_tokenize(s2)
```

Out[8]:
```
['``',
 'We',
 'beat',
 'some',
 'pretty',
 'good',
 'teams',
 'to',
 'get',
 'here',
 ',',
 "''",
 'Slocum',
 'said',
 '.']
```

In [9]:
```
s3 = "Well, we couldn't have this predictable, cliche-ridden, \"Touched by an A
s3
```

Out[9]:
```
'Well, we couldn\'t have this predictable, cliche-ridden, "Touched by an Angel"
(a show creator John Masius worked on) wanna-be if she didn\'t.'
```

In [10]:
```
word_tokenize(s3)
```

Out[10]:
```
['Well',
 ',',
 'we',
 'could',
 "n't",
 'have',
 'this',
 'predictable',
 ',',
 'cliche-ridden',
 ',',
 '``',
 'Touched',
 'by',
 'an',
 'Angel',
 "''",
 '(',
 'a',
 'show',
 'creator',
 'John',
 'Masius',
 'worked',
 'on',
 ')',
 'wanna-be',
 'if',
 'she',
 'did',
 "n't",
 '.']
```

In [11]:
```
s4 = "I cannot cannot work under these conditions!"
s4
```

Out[11]: 'I cannot cannot work under these conditions!'

In [12]:
```
word_tokenize(s4)
```

Out[12]: ['I', 'can', 'not', 'can', 'not', 'work', 'under', 'these', 'conditions', '!']

In [13]:
```
 s5 = "The company spent $30,000,000 last year."
s5
```

Out[13]: 'The company spent $30,000,000 last year.'

In [14]:
```
word_tokenize(s5)
```

Out[14]: ['The', 'company', 'spent', '$', '30,000,000', 'last', 'year', '.']

In [15]:
```
s6 = "The company spent 40.75% of its income last year."
s6
```

Out[15]: 'The company spent 40.75% of its income last year.'

In [16]:
```
word_tokenize(s6)
```

Out[16]:
```
['The',
 'company',
 'spent',
 '40.75',
 '%',
 'of',
 'its',
 'income',
 'last',
 'year',
 '.']
```

In [17]:
```
s7 = "He arrived at 3:00 pm."
s7
```

Out[17]: 'He arrived at 3:00 pm.'

In [18]:
```
word_tokenize(s7)
```

Out[18]: ['He', 'arrived', 'at', '3:00', 'pm', '.']

In [19]:
```
s8 = "I bought these items: books, pencils, and pens."
s8
```

Out[19]: 'I bought these items: books, pencils, and pens.'

In [20]:
```
word_tokenize(s8)
```

Out[20]:
```
['I',
 'bought',
 'these',
 'items',
 ':',
 'books',
 ',',
 'pencils',
 ',',
 'and',
 'pens',
 '.']
```

In [21]:
```
s9 = "Though there were 150, 100 of them were old."
s9
```

Out[21]:
```
'Though there were 150, 100 of them were old.'
```

In [22]:
```
word_tokenize(s9)
```

Out[22]:
```
['Though',
 'there',
 'were',
 '150',
 ',',
 '100',
 'of',
 'them',
 'were',
 'old',
 '.']
```

In [23]:
```
s10 = "There were 300,000, but that wasn't enough."
s10
```

Out[23]:
```
"There were 300,000, but that wasn't enough."
```

In [24]:
```
word_tokenize(s10)
```

Out[24]:
```
['There', 'were', '300,000', ',', 'but', 'that', 'was', "n't", 'enough', '.']
```

In [25]:
```
s11 = "It's more'n enough."
s11
```

Out[25]:
```
"It's more'n enough."
```

In [26]:
```
word_tokenize(s11)
```

Out[26]:
```
['It', "'s", 'more', "'n", 'enough', '.']
```

Gathering the spans of the tokenized strings.

In [28]:
```
s = '''Good muffins cost $3.88\nin New (York).  Please (buy) me\ntwo of them.\n(
s
```

Out[28]:
```
'Good muffins cost $3.88\nin New (York).  Please (buy) me\ntwo of them.\n(Thanks).'
```

In [29]:
```
expected = [(0, 4), (5, 12), (13, 17), (18, 19), (19, 23),
... (24, 26), (27, 30), (31, 32), (32, 36), (36, 37), (37, 38),
... (40, 46), (47, 48), (48, 51), (51, 52), (53, 55), (56, 59),
... (60, 62), (63, 68), (69, 70), (70, 76), (76, 77), (77, 78)]
expected
```

Out[29]:
```
[(0, 4),
 (5, 12),
 (13, 17),
 (18, 19),
 (19, 23),
 (24, 26),
 (27, 30),
 (31, 32),
 (32, 36),
 (36, 37),
 (37, 38),
 (40, 46),
 (47, 48),
 (48, 51),
 (51, 52),
 (53, 55),
 (56, 59),
 (60, 62),
 (63, 68),
 (69, 70),
 (70, 76),
 (76, 77),
 (77, 78)]
```

Testing improvement made to the TreebankWordTokenizer

In [31]:
```
sx1 = '\xabNow that I can do.\xbb'
expected = ['\xab', 'Now', 'that', 'I', 'can', 'do', '.', '\xbb']
word_tokenize(sx1) == expected
```

Out[31]:
```
True
```

In [32]:
```
 sx2 = 'The unicode 201C and 201D \u201cLEFT(RIGHT) DOUBLE QUOTATION MARK\u201d
sx2
```

Out[32]:
```
'The unicode 201C and 201D “LEFT(RIGHT) DOUBLE QUOTATION MARK” is also OPEN_PUNCT and CLOSE_PUNCT.'
```

In [33]:
```
word_tokenize(sx2)
```

Out[33]:
```
['The',
 'unicode',
 '201C',
 'and',
 '201D',
 '“',
 'LEFT',
 '(',
 'RIGHT',
 ')',
 'DOUBLE',
 'QUOTATION',
 'MARK',
 '”',
 'is',
 'also',
 'OPEN_PUNCT',
 'and',
 'CLOSE_PUNCT',
 '.']
```

In [34]:
```
sx2 = 'The unicode 201C and 201D \u201cLEFT(RIGHT) DOUBLE QUOTATION MARK\u201d
expected = ['The', 'unicode', '201C', 'and', '201D', '\u201c', 'LEFT', '(', 'RI
word_tokenize(sx2) == expected
```

Out[34]:
```
True
```

Testing treebank's detokenizer

In [36]:
```
from nltk.tokenize.treebank import TreebankWordDetokenizer
```

In [37]:
```
detokenizer = TreebankWordDetokenizer()
```

In [38]:
```
>>> s = "On a $50,000 mortgage of 30 years at 8 percent, the monthly payment wou
s
```

Out[38]:
```
'On a $50,000 mortgage of 30 years at 8 percent, the monthly payment would be
$366.88.'
```

In [39]:
```
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[39]:
```
'On a $50,000 mortgage of 30 years at 8 percent, the monthly payment would be
$366.88.'
```

In [40]:
```
>>> s = "\"We beat some pretty good teams to get here,\" Slocum said."
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[40]:
```
'"We beat some pretty good teams to get here," Slocum said.'
```

In [41]:
```
>>> s = "Well, we couldn't have this predictable, cliche-ridden, \"Touched by an
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[41]:
```
'Well, we couldn\'t have this predictable, cliche-ridden, "Touched by an Angel"
(a show creator John Masius worked on) wanna-be if she didn\'t.'
```

In [42]:
```
>>> s = "I cannot cannot work under these conditions!"
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[42]: 'I cannot cannot work under these conditions!'

In [43]:
```
>>> s = "The company spent $30,000,000 last year."
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[43]: 'The company spent $30,000,000 last year.'

In [44]:
```
>>> s = "The company spent 40.75% of its income last year."
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[44]: 'The company spent 40.75% of its income last year.'

In [45]:
```
>>> s = "He arrived at 3:00 pm."
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[45]: 'He arrived at 3:00 pm.'

In [46]:
```
>>> s = "I bought these items: books, pencils, and pens."
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[46]: 'I bought these items: books, pencils, and pens.'

In [47]:
```
>>> s = "Though there were 150, 100 of them were old."
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[47]: 'Though there were 150, 100 of them were old.'

In [48]:
```
>>> s = "There were 300,000, but that wasn't enough."
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[48]: "There were 300,000, but that wasn't enough."

In [49]:
```
>>> s = 'How "are" you?'
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[49]: 'How "are" you?'

In [50]:
```
>>> s = "Hello (world)"
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[50]: 'Hello (world)'

In [51]:
```
>>> s = '<A sentence> with (many) [kinds] of {parentheses}. "Sometimes it\'s ins
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[51]: '<A sentence> with (many) [kinds] of {parentheses}. "Sometimes it\'s inside (qu
otes)". ("Sometimes the otherway around").'

In [52]:
```
>>> s = "Sentence ending with (parentheses)"
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[52]: 'Sentence ending with (parentheses)'

In [53]:
```
>>> s = "(Sentence) starting with parentheses."
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[53]: '(Sentence) starting with parentheses.'

In [54]:
```
>>> s = "I've"
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[54]: "I've"

In [55]:
```
>>> s = "Don't"
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[55]: "Don't"

In [56]:
```
>>> s = "I'd"
>>> detokenizer.detokenize(word_tokenize(s))
```

Out[56]: "I'd"

Sentence tokenization in word_tokenize:

In [58]:
```
>>> s11 = "I called Dr. Jones. I called Dr. Jones."
>>> word_tokenize(s11)
```

Out[58]: ['I', 'called', 'Dr.', 'Jones', '.', 'I', 'called', 'Dr.', 'Jones', '.']

In [59]:
```
>>> s12 = ("Ich muss unbedingt daran denken, Mehl, usw. fur einen "
...        "Kuchen einzukaufen. Ich muss.")
>>> word_tokenize(s12)
```

Out[59]:
```
['Ich',
 'muss',
 'unbedingt',
 'daran',
 'denken',
 ',',
 'Mehl',
 ',',
 'usw',
 '.',
 'fur',
 'einen',
 'Kuchen',
 'einzukaufen',
 '.',
 'Ich',
 'muss',
 '.']
```

In [60]:
```
>>> word_tokenize(s12, 'german')
```

Out[60]:
```
['Ich',
 'muss',
 'unbedingt',
 'daran',
 'denken',
 ',',
 'Mehl',
 ',',
 'usw.',
 'fur',
 'einen',
 'Kuchen',
 'einzukaufen',
 '.',
 'Ich',
 'muss',
 '.']
```

Regression Tests: Regexp Tokenizer

In [62]:
```
from nltk.tokenize import regexp_tokenize
```

Some additional test strings.

In [64]:
```
>>> s2 = ("Alas, it has not rained today. When, do you think, "
...       "will it rain again?")
```

In [65]:
```
>>> regexp_tokenize(s2, r'[,\.\?!"]\s*', gaps=False)
```

Out[65]:
```
[', ', '. ', ', ', ', ', ', '?']
```

In [66]:
```
>>> regexp_tokenize(s2, r'[,\.\?!"]\s*', gaps=True)
```

Out[66]:
```
['Alas',
 'it has not rained today',
 'When',
 'do you think',
 'will it rain again']
```

In [67]:
```
>>> s3 = ("<p>Although this is <b>not</b> the case here, we must "
...       "not relax our vigilance!</p>")
s3
```

Out[67]:
```
'<p>Although this is <b>not</b> the case here, we must not relax our vigilance!
</p>'
```

Take care to avoide using capturing groups:

In [69]:
```
>>> regexp_tokenize(s3, r'</?[bp]>', gaps=False)
```

Out[69]:
```
['<p>', '<b>', '</b>', '</p>']
```

In [70]:
```
>>> regexp_tokenize(s3, r'</?(?:b|p)>', gaps=False)
```

Out[70]:
```
['<p>', '<b>', '</b>', '</p>']
```

In [71]:
```
>>> regexp_tokenize(s3, r'</?(?:b|p)>', gaps=True)
```

Out[71]:
```
['Although this is ',
 'not',
 ' the case here, we must not relax our vigilance!']
```

Named groups are capturing groups, and confuse the tokenizer:

In [73]:
```
>>> regexp_tokenize(s3, r'</?(?P<named>b|p)>', gaps=False)
```

Out[73]:
```
['p', 'b', 'b', 'p']
```

In [74]:
```
>>> regexp_tokenize(s3, r'</?(?P<named>b|p)>', gaps=True)
```

Out[74]:
```
['p',
 'Although this is ',
 'b',
 'not',
 'b',
 ' the case here, we must not relax our vigilance!',
 'p']
```

Make sure that nested groups don't confuse the tokenizer:

In [76]:
```
>>> regexp_tokenize(s2, r'(?:h|r|l)a(?:s|(?:i|n0))', gaps=False)
```

Out[76]:
```
['las', 'has', 'rai', 'rai']
```

In [77]:
```
>>> regexp_tokenize(s2, r'(?:h|r|l)a(?:s|(?:i|n0))', gaps=True)
```

Out[77]:
```
['A', ', it ', ' not ', 'ned today. When, do you think, will it ', 'n again?']
```

Back-references require capturing groups, and these are not supported:

In [79]:
```
>>> regexp_tokenize("aabbbcccc", r'(.)\1')
```

Out[79]:
```
['a', 'b', 'c', 'c']
```

A simple sentence tokenizer '.(s+|$)'

In [81]:
```
>>> regexp_tokenize(s, pattern=r'\.(?:\s+|$)', gaps=True)
```

Out[81]:
```
["I'd"]
```

Regression Tests: TweetTokenizer

In [152…
```
>>> from nltk.tokenize import TweetTokenizer
```

In [154…
```
>>> tknzr = TweetTokenizer()
```

In [156…
```
>>> s0 = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"
```

In [158…
```
>>> tknzr.tokenize(s0)
```

Out[158…
```
['This',
 'is',
 'a',
 'cooool',
 '#dummysmiley',
 ':',
 ':-)',
 ':-P',
 '<3',
 'and',
 'some',
 'arrows',
 '<',
 '>',
 '->',
 '<--']
```

In [160…
```
>>> s1 = "@Joyster2012 @CathStaincliffe Good for you, girl!! Best wishes :-)"
>>> tknzr.tokenize(s1)
```

Out[160…
```
['@Joyster2012',
 '@CathStaincliffe',
 'Good',
 'for',
 'you',
 ',',
 'girl',
 '!',
 '!',
 'Best',
 'wishes',
 ':-)']
```

In [162…
```
>>> s2 = "3Points for #DreamTeam Gooo BAILEY! :) #PBB737Gold @PBBabscbn"
>>> tknzr.tokenize(s2)
```

Out[162…
```
['3Points',
 'for',
 '#DreamTeam',
 'Gooo',
 'BAILEY',
 '!',
 ':)',
 '#PBB737Gold',
 '@PBBabscbn']
```

In [164…
```
>>> s3 = "@Insanomania They do... Their mentality doesn't :("
>>> tknzr.tokenize(s3)
```

Out[164…
```
['@Insanomania', 'They', 'do', '...', 'Their', 'mentality', "doesn't", ':(']
```

In [166…
```
>>> s4 = "RT @facugambande: Ya por arrancar a grabar !!! #TirenTirenTiren vamoo
>>> tknzr.tokenize(s4)
```

Out[166…
```
['RT',
 '@facugambande',
 ':',
 'Ya',
 'por',
 'arrancar',
 'a',
 'grabar',
 '!',
 '!',
 '!',
 '#TirenTirenTiren',
 'vamoo',
 '!',
 '!']
```

In [168…
```
>>> tknzr = TweetTokenizer(reduce_len=True)
>>> s5 = "@crushinghes the summer holidays are great but I'm so bored already :(
>>> tknzr.tokenize(s5)
```

Out[168…
```
['@crushinghes',
 'the',
 'summer',
 'holidays',
 'are',
 'great',
 'but',
 "I'm",
 'so',
 'bored',
 'already',
 ':(']
```

In [170…
```
>>> tknzr = TweetTokenizer(strip_handles=True, reduce_len=True)
>>> s6 = '@remy: This is waaaaayyyy too much for you!!!!!!'
>>> tknzr.tokenize(s6)
```

Out[170…
```
[':', 'This', 'is', 'waaayyy', 'too', 'much', 'for', 'you', '!', '!', '!']
```

In [172…
```
>>> s7 = '@_willy65: No place for @chuck tonight. Sorry.'
>>> tknzr.tokenize(s7)
```

Out[172…
```
[':', 'No', 'place', 'for', 'tonight', '.', 'Sorry', '.']
```

In [174…
```
>>> s8 = '@mar_tin is a great developer. Contact him at mar_tin@email.com.'
>>> tknzr.tokenize(s8)
```

Out[174…
```
['is',
 'a',
 'great',
 'developer',
 '.',
 'Contact',
 'him',
 'at',
 'mar_tin@email.com',
 '.']
```

In [176…
```
>>> tknzr = TweetTokenizer(preserve_case=False)
>>> s9 = "@jrmy: I'm REALLY HAPPYYY about that! NICEEEE :D :P"
>>> tknzr.tokenize(s9)
```

Out[176…
```
['@jrmy',
 ':',
 "i'm",
 'really',
 'happyyy',
 'about',
 'that',
 '!',
 'niceeee',
 ':D',
 ':P']
```

It should not hang on long sequences of the same punctuation character.

In [179…
```
>>> tknzr = TweetTokenizer()
>>> s10 = "Photo: Aujourd'hui sur http://t.co/0gebOFDUzn Projet... http://t.co/b
>>> tknzr.tokenize(s10)
```

Out[179…
```
['Photo',
 ':',
 "Aujourd'hui",
 'sur',
 'http://t.co/0gebOFDUzn',
 'Projet',
 '...',
 'http://t.co/bKfIUbydz2',
 '...',
 'http://fb.me/3b6uXpz0L']
```

Tokenizing multiple sentences at once:

In [182…
```
>>> tknzr = TweetTokenizer()
>>> sentences = [
...     "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--",
...     "@jrmy: I'm REALLY HAPPYYY about that! NICEEEE :D :P",
...     "@_willy65: No place for @chuck tonight. Sorry."
... ]
>>> tknzr.tokenize_sents(sentences)
```

Out[182…
```
[['This',
  'is',
  'a',
  'cooool',
  '#dummysmiley',
  ':',
  ':-)',
  ':-P',
  '<3',
  'and',
  'some',
  'arrows',
  '<',
  '>',
  '->',
  '<--'],
 ['@jrmy',
  ':',
  "I'm",
  'REALLY',
  'HAPPYYY',
  'about',
  'that',
  '!',
  'NICEEEE',
  ':D',
  ':P'],
 ['@_willy65',
  ':',
  'No',
  'place',
  'for',
  '@chuck',
  'tonight',
  '.',
  'Sorry',
  '.']]
```

Regression Tests: PunktSentenceTokenizer

In [189…
```
>>> from nltk.tokenize.punkt import PunktBaseClass, PunktTrainer, PunktSentenceT
>>> from nltk.tokenize.punkt import PunktLanguageVars, PunktParameters
>>> pbc = PunktBaseClass(lang_vars=None, params=None)
>>> type(pbc._params)
```

Out[189…
```
nltk.tokenize.punkt.PunktParameters
```

In [191…
```
>>> type(pbc._lang_vars)
```

Out[191…
```
nltk.tokenize.punkt.PunktLanguageVars
```

In [193…
```
>>> pt = PunktTrainer(lang_vars=None)
>>> type(pt._lang_vars)
```

Out[193…
```
nltk.tokenize.punkt.PunktLanguageVars
```

In [195…
```
>>> pst = PunktSentenceTokenizer(lang_vars=None)
```

```
>>> type(pst._lang_vars)
```

Out[195…
```
nltk.tokenize.punkt.PunktLanguageVars
```

Regression Tests: align_tokens

In [198…
```
>>> from nltk.tokenize.util import align_tokens
>>> list(align_tokens([''], ""))
```

Out[198…
```
[(0, 0)]
```

In [200…
```
>>> list(align_tokens(['abc', 'def'], "abcdef"))
```

Out[200…
```
[(0, 3), (3, 6)]
```

In [202…
```
>>> list(align_tokens(['ab', 'cd', 'ef'], "ab cd ef"))
```

Out[202…
```
[(0, 2), (3, 5), (6, 8)]
```

In [208…
```
>>> list(align_tokens(['The', 'plane', ',', 'bound', 'for', 'St', 'Petersburg',
```

Out[208…
```
[(0, 3),
 (4, 9),
 (9, 10),
 (11, 16),
 (17, 20),
 (21, 23),
 (24, 34),
 (34, 35),
 (36, 43),
 (44, 46),
 (47, 52),
 (52, 54),
 (55, 60),
 (61, 67),
 (68, 72),
 (73, 75),
 (76, 83),
 (84, 89),
 (90, 98),
 (99, 103),
 (104, 109),
 (110, 119),
 (120, 122),
 (123, 131),
 (131, 132)]
```

Regression Tests: MWETokenizer

In [211…
```
>>> from nltk.tokenize import MWETokenizer
>>> import pickle
```

In [213…
```
>>> tokenizer = MWETokenizer([('hors', "d'oeuvre")], separator='+')
>>> p = pickle.dumps(tokenizer)
>>> unpickeled = pickle.loads(p)
>>> unpickeled.tokenize("An hors d'oeuvre tonight, sir?".split())
```

Out[213… ['An', "hors+d'oeuvre", 'tonight,', 'sir?']

Regression Tests: TextTilingTokenizer

In [216…
```
>>> from nltk.tokenize import TextTilingTokenizer
>>> from nltk.corpus import brown
>>> tt = TextTilingTokenizer()
>>> tt.tokenize(brown.raw()[0:1000])
```

Out[216… ["\n\n\tThe/at Fulton/np-tl County/nn-tl Grand/jj-tl Jury/nn-tl said/vbd Friday/nr an/at investigation/nn of/in Atlanta's/np$ recent/jj primary/nn election/nn produced/vbd ``/`` no/at evidence/nn ''/'' that/cs any/dti irregularities/nns took/vbd place/nn ./.\n\n\n\tThe/at jury/nn further/rbr said/vbd in/in term-end/nn presentments/nns that/cs the/at City/nn-tl Executive/jj-tl Committee/nn-tl ,/, which/wdt had/hvd over-all/jj charge/nn of/in the/at election/nn ,/, ``/`` deserves/vbz the/at praise/nn and/cc thanks/nns of/in the/at City/nn-tl of/in-tl Atlanta/np-tl ''/'' for/in the/at manner/nn in/in which/wdt the/at election/nn was/bedz conducted/vbn ./.\n\n\n\tThe/at September-October/np term/nn jury/nn had/hvd been/ben charged/vbn by/in Fulton/np-tl Superior/jj-tl Court/nn-tl Judge/nn-tl Durwood/np Pye/np to/to investigate/vb reports/nns of/in possible/jj ``/`` irregularities/nns ''/'' in/in the/at hard-fought/jj primary/nn which/wdt was/bedz won/vbn by/in Mayor-nominate/nn-tl Ivan/np Allen/np Jr./"]

Regression Test: PorterStemmer

In [221…
```
from nltk.stem import PorterStemmer
pst = PorterStemmer()
```

In [223…
```
pst.stem('affection')
```

Out[223… 'affect'

In [225…
```
pst.stem('playing')
```

Out[225… 'play'

In [227…
```
pst.stem('maximum')
```

Out[227… 'maximum'

In [229…
```
words_to_stem=['give','giving','given','gave']

for words in words_to_stem:
    print(words+ ' : ' + pst.stem(words))
```

```
give : give
giving : give
given : given
gave : gave
```

In [ ]: