Lemmatization

Computational Linguistics
Emory University
Jinho D. Choi
Lemmatization

Convert each word into its root form.

Verbs
- 3rd-person singular
  - past: studies → study
  - studied → study
  - gerund: studying → study
  - irregular: took → take, taken → take

Adjectives
- comparative: easier → easy, easiest → easy
- superlative: better → good, best → good
- irregular: better → well, best → well

Nouns
- plural: studies → study, indices → index
- irregular

Adverbs
- comparative: earlier → early, earliest → early
- superlative: better → well, best → well
- irregular
def readBaseSet(fin):  
    s = set() 

    for line in fin: 
        s.add(line.strip()) 

    return s 

VERB_BASE_SET = readBaseSet(open('verb.base')) 
NOUN_BASE_SET = readBaseSet(open('noun.base')) 
ADJECTIVE_BASE_SET = readBaseSet(open('adjective.base')) 
ADVERB_BASE_SET = readBaseSet(open('adverb.base'))
def readExceptionDictionary(fin):
    d = dict()

    for line in fin:
        l = line.split()
        d[l[0]] = l[1]  # l[0]: irregular form  
                       # l[1]: base form

    return d

VERB_EXC_DICT = readExceptionDictionary(open('verb.exc'))
NOUN_EXC_DICT = readExceptionDictionary(open('noun.exc'))
ADJECTIVE_EXC_DICT = readExceptionDictionary(open('adjective.exc'))
ADVERB_EXC_DICT = readExceptionDictionary(open('adverb.exc'))
Rules

suffix  replacement

VERB_RULES = [
    ('ies', 'y'), studies
    ('ied', 'y'), studied
    ('es', ''), pushes
    ('ed', ''), entered
    ('s', ''), takes
    ('d', ''), heard
    ('ying', 'ie'), lying
    ('ing', ''), studying
    ('ing', 'e'), taking
    ('n', ''), drawn
    ('ung', 'ing'), clung
]

ADJECTIVE_RULES = [
    ('ier', 'y'), easier
    ('iest', 'y'), easiest
    ('er', ''), smaller
    ('est', ''), smallest
    ('er', 'e'), larger
    ('est', 'e'), largest
]

NOUN_RULES = [
    ('ies', 'y'), studies
    ('es', ''), crosses
    ('s', ''), areas
    ('men', 'man'), woman
    ('ae', 'a'), lying
    ('ae', 'a'), vertebral
    ('i', 'us'), foci
]

ADVERB_RULES = [
    ('ier', 'y'), easier
    ('iest', 'y'), easiest
    ('er', ''), smaller
    ('est', ''), smallest
    ('er', 'e'), larger
    ('est', 'e'), largest
]
def lemmatize(word, baseset, excdict, rulelist):
    word = word.lower()

    if word in baseset:
        return word

    if word in excdict:
        return excdict[word]

    for rule in rulelist:
        if word.endswith(rule[0]):
            lemma = word[:len(rule[0])] + rule[1]
            if lemma in baseset:
                return lemma

    return None
def lemmatize(word, pos):
    if pos is 'v': return _lemmatize(word, VERB[0], VERB[1], VERB[2])
    if pos is 'n': return _lemmatize(word, NOUN[0], NOUN[1], NOUN[2])
    if pos is 'j': return _lemmatize(word, ADJT[0], ADJT[1], ADJT[2])
    if pos is 'r': return _lemmatize(word, ADRB[0], ADRB[1], ADRB[2])
    return word.lower()

What if the part-of-speech tag is unknown?
Lemmatize

```python
def lemmatize(word, pos):
    if pos is 'v': return _lemmatize(word, VERB[0], VERB[1], VERB[2])
    if pos is 'n': return _lemmatize(word, NOUN[0], NOUN[1], NOUN[2])
    if pos is 'j': return _lemmatize(word, ADJT[0], ADJT[1], ADJT[2])
    if pos is 'r': return _lemmatize(word, ADRB[0], ADRB[1], ADRB[2])
    return word.lower()
```
Evaluation

space for a **string** whose length is 10

```
word = 'studied'
print '%10s -> %10s' % (word, lemmatize(word) )
word = 'studying'
print '%10s -> %10s' % (word, lemmatize(word) )
word = 'crosses'
print '%10s -> %10s' % (word, lemmatize(word) )
word = 'women'
print '%10s -> %10s' % (word, lemmatize(word) )
word = 'easier'
print '%10s -> %10s' % (word, lemmatize(word) )
word = 'easiest'
print '%10s -> %10s' % (word, lemmatize(word) )
word = 'larger'
print '%10s -> %10s' % (word, lemmatize(word) )
word = 'largest'
print '%10s -> %10s' % (word, lemmatize(word) )
```
Duplicated Consonants

running, zipped, bigger, biggest

running → run
zipped → zip
bigger → big
biggest → big

VERB_RULES = [..., ('ing', '', True), ('ing', 'e', True), ('ed', '', True)]
NOUN_RULES = [...]
ADJECTIVE_RULES = [..., ('er', '', True), ('est', '', True)]
ADVERB_RULES = [..., ('er', '', True), ('est', '', True)]

Exercise:
Modify the _lemmatize function
to adapt these new rules.
def _lemmatize(word, baseset, excdict, rulelist):
    word = word.lower()

    if word in baseset:
        return word

    if word in excdict:
        return excdict[word]

    for rule in rulelist:
        if word.endswith(rule[0]):
            idx = len(rule[0])
            if len(rule) > 2 and rule[2]: idx += 1
            lemma = word[::idx] + rule[1]
            if lemma in baseset: return lemma

    return None