Lab 5: List and Dictionary Methods

Ling 1330/2330: Intro to Computational Linguistics
Na-Rae Han
Objectives

- Building a list, sorting with `sorted()`
- Introduction dictionary, dictionary methods
- Counting with dictionary
- Tokenizer function

- Assignment review
  - Pig Latin
Pig Latin

```python
def vowelIndex(wd):

def getInitialCs(wd):
    return wd[:vowelIndex(wd)]

def getTheRest(wd):
    return wd[vowelIndex(wd):]

def pigLatin(wd):
    if wd[0] in 'aeiou':
        return wd + 'way'
    else:
        return getTheRest(wd) + getInitialCs(wd) + 'ay'
```

>>> pigLatin('')
Traceback (most recent call last):
...
    if wd[0] in 'aeiou':
IndexError: string index out of range
Pig Latin: more robust

```
def vowelIndex(wd):
    ...

def getInitialCs(wd):
    return wd[:vowelIndex(wd)]

def getTheRest(wd):
    return wd[vowelIndex(wd):]

def pigLatin(wd):
    if len(wd) == 0: return ''
    elif wd[0] in 'aeiou':
        return wd + 'way'
    else:
        return getTheRest(wd) + getInitialCs(wd) + 'ay'

>>> pigLatin('')
''
```

Addresses fringe cases. Function is more robust.

Any other improvements?
Pig Latin: 'y' as a vowel

```python
>>> pigLatin('yellow')
'ellowyay'

>>> pigLatin('ploy')
'oyplay'

>>> pigLatin('python')
'onpythay'

>>> vowelIndex('python')
4

>>> getInitialCs('python')
'pyth'
```

Good. 'y' is a consonant

Also good. 'y' is a part of a diphthong

Not good. 'y' is a vowel here but not treated as one.

To properly handle words with 'y' as the initial vowel, vowelIndex() function must be further refined.
Issues in tokenization

>>> sent = 'rose is a rose is a rose is a rose .'
>>> sent.split()
['rose', 'is', 'a', 'rose', 'is', 'a', 'rose', '.']

>>> sent = 'Rose is a rose is a rose is a rose.'
>>> sent.split()
['Rose', 'is', 'a', 'rose', 'is', 'a', 'rose.]

>>> joke = "Knock knock, who's there?"
>>> joke.split()
['Knock', 'knock,', 'who's', 'there?']

.split() is too simple as a tokenizer.

We need a better tokenizer:
- folds case
- separates out symbols
Lower, space out symbols, then split

"Knock knock, who's there?"

```
.lower()
"knock knock, who's there?"
```

```
.replace()
"knock knock, who ' s there ?"
```

```
.split()
['knock', 'knock', ',', ',', 'who', '', '', 's', ', ', 'there', ', ', '?']
```
Tokenizer function, first try

A function that tokenizes a sentence:

```python
def getTokens(sent):
    new = sent.lower()
    new = new.replace(",", " , ").replace("'", " ' ").replace("?", " ? ")
    return new.split()
```

```python
>>> joke = "Knock knock, who's there?"
>>> getTokens(joke)
['knock', 'knock', ',', ',', 'who', '¨', 's', 'there', '?'
```

Success!

Why "¨ " instead of just " "?

inserts a space before and after '

'? is its own word.
A function that tokenizes a sentence:

```python
def getTokens(sent):
    new = sent.lower()
    new = new.replace(",", " , ").replace("'", " ' ").replace("?", " ? ")
    return new.split()
```

But there are so many more symbols: 

`. : # $ ! …`

```python
>>> joke = "Knock knock, who's there?"
>>> getTokens(joke)
['knock', 'knock', ',', ',', 'who', "'", 's', 'there', '?']
```

This replace line is about to become a mile long!
Tokenizer function, second try

A function that tokenizes a sentence:

```python
def getTokens(sent):
    new = sent.lower()
    for s in ',.!?':;#$':
        new = new.replace(s, " +s+ ")
    return new.split()
```

```
>>> joke = "Knock knock, who's there?"
>>> getTokens(joke)
['knock', 'knock', ',', 'who', 's', 'there', '?']
>>> pal = "A man, a plan, a canal: Panama."
>>> getTokens(pal)
['a', 'man', ',', 'a', 'plan', ',', 'a', 'canal', ':', 'panama', '.']
```
Tokenizer function

A function that tokenizes a sentence:

```python
def getTokens(sent):
    new = sent.lower()
    for s in ',.!?;:#$':
        new = new.replace(s, ' '+s+' ')
    return new.split()
```

```python
>>> joke = "Knock knock, who's there?"
>>> getTokens(joke)
['knock', 'knock', ',', 'who', ' ', 's', ' ', 'there', ' ', '?']
>>> pal = "A man, a plan, a canal: Panama."
>>> getTokens(pal)
['a', 'man', ',', 'a', 'plan', ',', 'a', 'canal', ':', 'panama', '.']
>>>
Python data types

33 \(\rightarrow\) \textbf{int}: integer
5.49 \(\rightarrow\) \textbf{float}: floating point number

'Bart'
'Hello, world!' \(\rightarrow\) \textbf{str}: string (a piece of text)

['cat', 'dog', 'fox', 'hippo'] \(\rightarrow\) \textbf{list}

('gold', 'silver', 'bronze') \(\rightarrow\) \textbf{tuple}

{'Homer':36, 'Marge':36, 'Bart':10, 'Lisa':8, 'Maggie':1} \(\rightarrow\) \textbf{dict}: (dictionary) maps a value to an object
Changing list entry

```python
>>> li = [1,2,3]
>>> li
[1, 2, 3]
>>> li[2] = 10
>>> li
[1, 2, 10]

>>> ani = ['fox', 'owl', 'dog']
>>> ani
['fox', 'owl', 'dog']
>>> ani[1] = 'bat'
>>> ani
['fox', 'bat', 'dog']
```
Changing list entry

```python
>>> li = [1, 2, 3]
>>> li
[1, 2, 3]
>>> li[2] = 10
>>> li
[1, 2, 10]
```  

```python
>>> ani = ['fox', 'owl', 'dog']
>>> ani
['fox', 'owl', 'dog']
>>> ani[1] = 'bat'
>>> ani
['fox', 'bat', 'dog']
```  

A new assignment statement updates an existing list entry
Adding to a list

```python
>>> li = [1,2,3]
>>> li
[1, 2, 3]
>>> li.append(4)
>>> li
[1, 2, 3, 4]

>>> ani = ['fox', 'owl', 'dog']
>>> ani
['fox', 'owl', 'dog']
>>> ani.append('cat')
>>> ani
['fox', 'owl', 'dog', 'cat']
```
Adding to a list

```python
>>> li = [1,2,3]
>>> li
[1, 2, 3]
>>> li.append(4)
>>> li
[1, 2, 3, 4]
>>> ani = ['fox', 'owl', 'dog']
>>> ani
['fox', 'owl', 'dog']
>>> ani.append('cat')
>>> ani
['fox', 'owl', 'dog', 'cat']
```

`.append(x)` adds a *single item* at the end.
.append() syntax

```python
>>> li = [1,2,3]
>>> li
[1, 2, 3]
>>> li.append(4)
>>> li
[1, 2, 3, 4]
```

```python
>>> ani = ['fox', 'owl', 'dog']
>>> ani
['fox', 'owl', 'dog']
>>> ani.append('cat')
>>> ani
['fox', 'owl', 'dog', 'cat']
```

Note the syntax!!

This part is never, ever necessary for .append()
def pigLatinSent() :
    """Prompts for input sentence, prints Pig Latin translation
    """
    tran = []
    sent = input('Give me a sentence: ')
    for w in sent.lower().replace('.', '').split():
        tran.append(pigLatin(w))

    print("Your Pig Latin sentence is:")
    print('"' + ' '.join(tran).capitalize() + '."')
Sorting a list with `sorted()`

- `sorted(list)`

```python
>>> li = [2, 3, 1, 4]
>>> sorted(li)
[1, 2, 3, 4]
>>> li
[2, 3, 1, 4]

>>> ani = ['fox', 'owl', 'dog', 'cat']
>>> sorted(ani)
['cat', 'dog', 'fox', 'owl']
>>> ani
['fox', 'owl', 'dog', 'cat']
```
Sorting a list with `sorted()`

- **`sorted(list)`**

```python
>>> li = [2, 3, 1, 4]
>>> sorted(li)
[1, 2, 3, 4]
>>> li
[2, 3, 1, 4]
>>> li is not changed

*Creates and returns a new sorted list*
Sorting a list with **sorted()**

- **sorted(list)**

  *Creates and returns a new alphabetically sorted list*

  ```python
  >>> ani = ['fox', 'owl', 'dog', 'cat']
  >>> sorted(ani)
  ['cat', 'dog', 'fox', 'owl']
  >>> ani
  ['fox', 'owl', 'dog', 'cat']
  >>>
  ```

  *ani is unchanged*
Returning consonant list

- Returning a list of consonants:

```python
def conlist(wd):
    con = []
    for c in wd:
        if c not in 'aeiou':
            con.append(c)
    return con

>>> conlist('world')
['w', 'r', 'l', 'd']
>>> conlist('procrastination')
['p', 'r', 'c', 'r', 's', 't', 'n', 't', 'n']
```

Would be great to have this in alphabetical order…
Returning consonant list, sorted

- Returning a list of consonants:

```python
def conlist(wd):
    con = []
    for c in wd:
        if c not in 'aeiou':
            con.append(c)
    return sorted(con)

>>> conlist('world')
['d', 'l', 'r', 'w']
>>> conlist('procrastination')
['c', 'n', 'n', 'p', 'r', 'r', 's', 't', 't']
>>> SUCCESS!
```
dict: a dictionary data type

{ 'Homer':36, 'Marge':36, 'Bart':10, 'Lisa':8, 'Maggie':1}  
⇒ A dictionary of the Simpson family members' age

{ 'go':'went', 'eat':'ate', 'see':'saw', 'say':'said' }  
⇒ A dictionary of verb past tense

- Dictionaries store a mapping between a set of keys and a set of values.
  - Keys can be any immutable type: string, integer, tuple
  - Values can be any type (can also be mixed types)
  - There is no inherent order (unlike lists and tuples)
  - You can define, modify, view, lookup, and delete the key-value pairs in the dictionary.
>>> en2es = {'cat': 'gato', 'dog': 'perro', 'tiger': 'tigre'}
>>> en2es
{'tiger': 'tigre', 'dog': 'perro', 'cat': 'gato'}

>>> en2es['cat']
'gato'

>>> en2es['dog']
'perro'

Scrambled. There is no inherent ordering.

Looking up is done through key
Dictionary lookup errors

>>> en2es = {'cat':'gato', 'dog':'perro', 'tiger':'tigre'}
>>> en2es
{'tiger': 'tigre', 'dog': 'perro', 'cat': 'gato'}

>>> en2es['platypus']
Traceback (most recent call last):
  File "<pyshell#31>" , line 1, in <module>
    en2es['platypus']
KeyError: 'platypus'

Dictionary is one way.
Cannot look up based on the value.
← Mapping can be many-to-one

>>> en2es['gato']
Traceback (most recent call last):
  File "<pyshell#32>" , line 1, in <module>
    en2es['gato']
KeyError: 'gato'
Checking if something's in a dictionary

```python
>>> en2es
{'tiger': 'tigre', 'dog': 'perro', 'cat': 'gato'}

>>> 'platypus' in en2es
False

>>> 'cat' in en2es
True

>>> 'gato' in en2es
False
```

"in" tests if a key is in a dictionary

"in" does not work with value
Practice: getPlural()

- Write a "pluralizer" function

```python
irrpl = {'man':'men', 'goose':'geese', 'mouse':'mice', 'child':'children'}

def getPlural(noun):
    ??
```

>>> getPlural('cat')
'cats'
>>> getPlural('goose')
'geese'
Practice: getPlural()

- Write a "pluralizer" function

```python
irrpl = {'man':'men', 'goose':'geese', 'mouse':'mice', 'child':'children'}

def getPlural(noun):
    if noun in irrpl:
        return irrpl[noun]
    else:
        return noun + 's'

>>> getPlural('cat')
'cats'
>>> getPlural('goose')
'geese'
```

If noun is in the irregular pl noun dictionary, return the dict value.
Updating vs. adding new entry

```python
>>> sim = {'Homer': 36, 'Marge': 35, 'Bart': 10, 'Lisa': 8}
>>> sim['Marge']
35
>>> sim['Marge'] = 36
>>> sim
{'Homer': 36, 'Marge': 36, 'Bart': 10, 'Lisa': 8}
>>> sim['Maggie'] = 1
>>> sim
{'Homer': 36, 'Marge': 36, 'Bart': 10, 'Maggie': 1, 'Lisa': 8}
```

- **key already exists.** Updates its value.
- **key does not exist!** Creates a new key and gives value.
Deleting entry

```python
>>> sim
{'Homer': 36, 'Marge': 36, 'Bart': 10, 'Maggie': 1, 'Lisa': 8}
>>> del sim['Bart']
>>> sim
{'Homer': 36, 'Marge': 36, 'Maggie': 1, 'Lisa': 8}
```
Finding out what's in

>>> sim = {'Lisa':8, 'Bart':10, 'Marge':35, 'Homer':36}

>>> ???
dict_keys(['Lisa', 'Marge', 'Bart', 'Homer'])

>>> ???
dict_values([8, 35, 10, 36])
Finding out what's in

```python
>>> sim = {'Lisa':8, 'Bart':10, 'Marge':35, 'Homer':36}

>>> sim.keys()
dict_keys(['Lisa', 'Marge', 'Bart', 'Homer'])

>>> sim.values()
dict_values([8, 35, 10, 36])
```

`.keys()` returns a quasi-list of keys, `.values()` returns a quasi-list of values.

The orders match!
.keys() .values(): list-like, but not quite

```python
>>> sim = {'Lisa': 8, 'Bart': 10, 'Marge': 35, 'Homer': 36}
>>> sim.keys()
dict_keys(['Lisa', 'Marge', 'Bart', 'Homer'])
>>> sim.values()
dict_values([8, 35, 10, 36])
>>> for v in sim.values():
    print(v)
8
35
10
36
```

Quasi-lists: These behave very much like a list, but not always

```python
>>> len(sim.keys())
4
>>> sum(sim.values())
89
```

```python
>>> sim.keys()[2]
Traceback (most recent call last):
  File "<pyshell#56>" , line 1, in <module>
    sim.values()[2]
TypeError: 'dict_keys' object does not support indexing
```
**.items() to reference (key, value) pairs**

```python
>>> sim
{'Lisa': 8, 'Marge': 35, 'Bart': 10, 'Homer': 36}

>>> sim.items()
dict_items([(‘Lisa’, 8), (‘Marge’, 35), (‘Bart’, 10), (‘Homer’, 36)])
```

items() returns a quasi-list of (key, value) pairs
Iterating through (key, value) pairs

```python
>>> sim
{'Lisa': 8, 'Marge': 35, 'Bart': 10, 'Homer': 36}

>>> sim.items()
dict_items([('Lisa', 8), ('Marge', 35), ('Bart', 10), ('Homer', 36)])

>>> for (k, v) in sim.items():
    print(k, 'is', v, 'years old.')</code>

Lisa is 8 years old.
Marge is 35 years old.
Bart is 10 years old.
Homer is 36 years old.
```

.items() lets you reference key and its value side by side as (k, v) in a for loop!
Iterating through keys

```python
>>> sim
{'Lisa': 8, 'Marge': 35, 'Bart': 10, 'Homer': 36}
```

```python
>>> for k in sim:
    print(k, 'is', sim[k], 'years old."
```

Lisa is 8 years old.
Marge is 35 years old.
Bart is 10 years old.
Homer is 36 years old.

Looping through keys then accessing value through `dict[key]`. Does the same thing!
Common dict application: counting

```python
>>> tally = {'gold':1, 'bronze':3}
>>> tally['bronze']
3
>>> newmedals = ['bronze', 'silver', 'gold', 'gold', 'silver']
>>> for m in newmedals:
...    tally[m] += 1

Traceback (most recent call last):
  File "<pyshell#80>", line 2, in <module>
    tally[m] += 1
KeyError: 'silver'
```

Error:
'silver' is not yet in the dictionary as a key
Initial key created, value set to 1

```python
>>> tally = {'gold':1, 'bronze':3}
>>> tally['bronze']
3
>>> newmedals = ['bronze', 'silver', 'gold', 'gold', 'silver']
>>> for m in newmedals:
...     if m not in tally:
...         tally[m] = 1
...     else:
...         tally[m] += 1

>>> tally
{'bronze': 4, 'gold': 3, 'silver': 2}
```

Make sure to account for the initial key creation & assignment
Dictionary of frequency count

```python
>>> chars = list('abracadabra')
>>> chars
['a', 'b', 'r', 'a', 'c', 'a', 'd', 'a', 'b', 'r', 'a']
>>> freq = {}
>>> for c in chars:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1

>>> freq
{'r': 2, 'a': 5, 'b': 2, 'c': 1, 'd': 1}

Initialize an empty dictionary
For loop through list, tally up the counts
```
Try it out

>>> chars = list('abracadabra')
>>> chars
['a', 'b', 'r', 'a', 'c', 'a', 'd', 'a', 'b', 'r', 'a']
>>> freq = {}
>>> for c in chars:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1

>>> freq
{'r': 2, 'a': 5, 'b': 2, 'c': 1, 'd': 1}

>>> for (k, v) in freq.items():
    print(k, v)
also try

2 minutes
When we have a text, we typically would like to know:

- How many words are in it? (Token count)
- How many unique words are in it? (Type count) What are their frequency counts? (Frequency dictionary)
- How many times is word x found?
- Are there a lot of long/short words? Which ones are long?
- Which words are very frequent? How frequent?
Wrap-up

- **Next class:**
  - *Language and Computers*, Ch. 2 Writer's aids: spell checkers

- **Homework #2**
  - [http://www.pitt.edu/~naraehan/ling1330/hw2.html](http://www.pitt.edu/~naraehan/ling1330/hw2.html)
  - Beginning programmers: plan on getting help for "Secondary Stats" part

- **Recitation**
  - All 3 sessions will be held at [Hillman Library Room 389](http://www.pitt.edu/~naraehan/ling1330/hw2.html) (this week only)
  - Try PART 2 "Basic Text Stats" before coming