Lecture 5: N-gram Context, List Comprehension

Ling 1330/2330 Computational Linguistics
Na-Rae Han, 9/12/2023

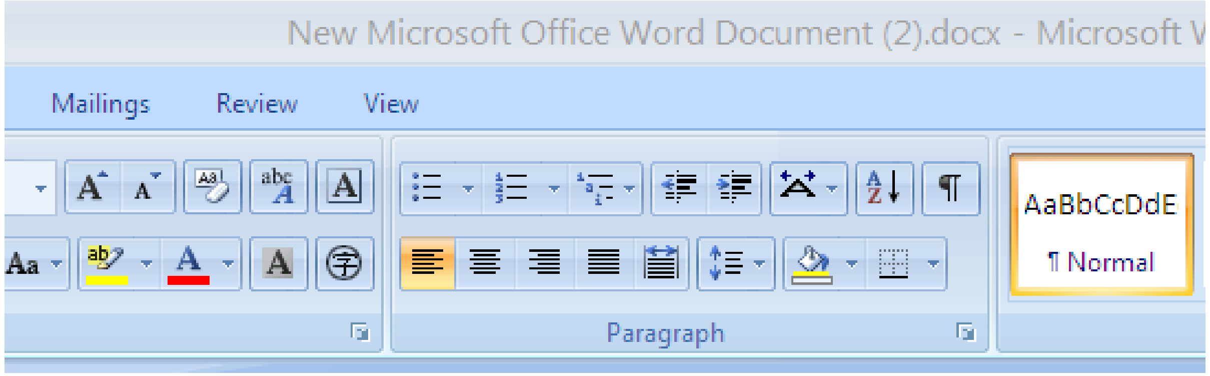
Objectives

- Context-aware spell checkers
  - *n*-gram as context
  - Character-level *n*-grams
  - Word-level *n*-grams
- Frequent *n*-grams in English
- NLTK
  - Buliding n-grams
  - n-gram frequency distribution
- Data resources on the web
  - Enable list, pickling
- List comprehension

Spell checkers

- Which spell checkers work well, which don’t? In what way?
- Anything else you noticed?

MS Word considers word contexts

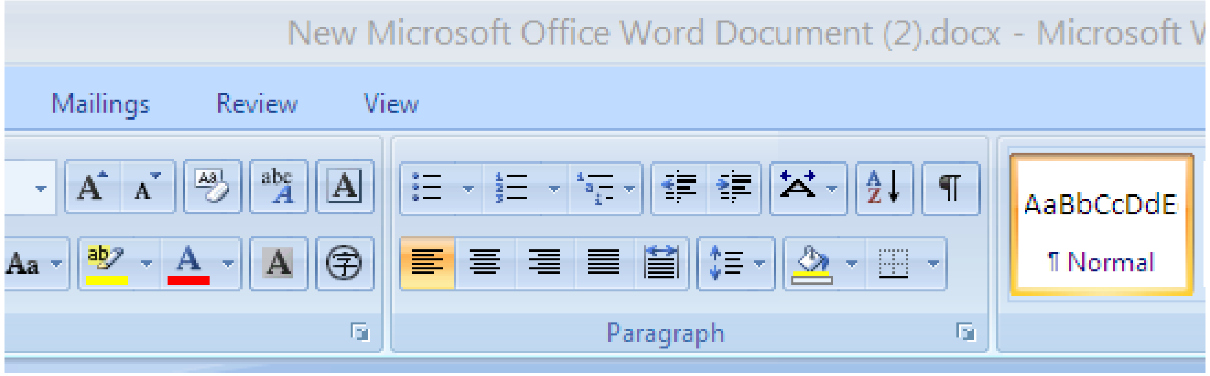


Please submit your **form**.

Please submit your **from**.

Are you **form**

MS Word considers word contexts



Please submit your **form**.

Please submit your **from**.

suggests "your" → "you're" instead of "from" → "form"

Are you **form**

suggests "form" → "from"

n-grams: character-level

- ***n*-gram**: a stretch of text *n* units long
  - unigrams (1), bigrams (2), trigrams (3), 4-grams, 5-grams, …

```
'green ideas'
```

- Character unigrams:

```
['g', 'r', 'e', 'e', 'n', ' ', 'i', 'd', 'e', 'a', 's']
```

- Character bigrams:

```
['gr', 're', 'ee', 'en', 'n ', ' i', 'id', 'de', 'ea', 'as']
```

- Character trigrams:

```
['gre', 'ree', 'een', 'en ', 'n i', ' id', 'ide', 'dea', 'eas']
```

- Character 4-grams:

```
['gree', 'reen', 'een ', 'en i', 'n id', ' ide', 'idea', 'deas']
```

n-grams: word-level

- ***n*-gram**: a stretch of text *n* units long
  - unigrams (1), bigrams (2), trigrams (3), 4-grams, 5-grams, ...

```
'Colorless green ideas sleep furiously.'
```

- Word bigrams:

```
[('colorless', 'green'), ('green', 'ideas'), ('ideas', 'sleep'), ('sleep',
'furiously'), ('furiously', '.')]
```

- Word trigrams:

```
[('colorless', 'green', 'ideas'), ('green', 'ideas', 'sleep'), ('ideas',
'sleep', 'furiously'), ('sleep', 'furiously', '.')]
```

n-grams and probability

- How likely do you think these letter bigrams are in English:
  - 'th' 'ti' 'tb' 'tq' 'tx'
- Putting it in terms of **conditional probability**:
  - After a user typed in letter 't', what is the most likely next character input?
  - How about after 'q'? After 'io'?
- For fun:
  - What are the most frequent English letter bigrams?
    - th, he, in, er, an, re, nd, on, en, at
  - Trigrams?
    - the, and, ing, her, hat, his, tha, ere, for, ent

Word-level *n*-grams

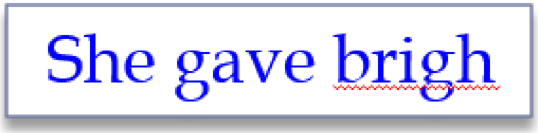
- How likely do you think these n-grams are in English:

are you	46622
is you	4441
are you so	428
are you also	26
are you does	-

- Putting in terms of conditional probability:
  - After a user types in 'are you', what is the most likely next word?
  - How about 'in the'? 'in the middle'?

N-grams in spell checker, NLP

- N-grams play a major role in many NLP applications:
  - They are units for capturing & quantifying **linguistic context**.
- N-grams vs. edit distance in spell checker

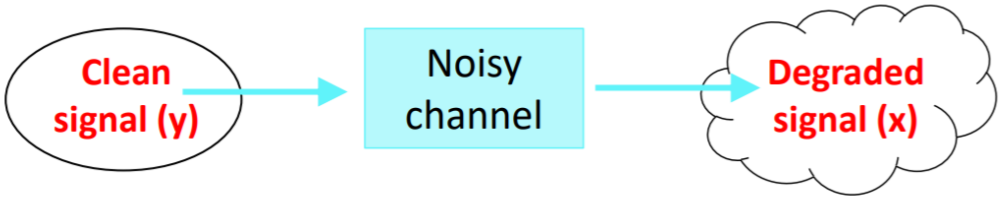


She gave brigh

  - **Edit distance**: which target words are closest to the original misspelled word ("brigh")? (*bright, brig* > *birth* > *brought* > …)
  - **N-gram context**: given two previous words ("She gave"), what is the most likely next word? (*them* > *back* > *birth* > …)

← Choice should weigh between these two competing factors

← The **noisy channel model** (we'll come back to this)



$$\hat{y} = \text{argmax}_y P(y)P(x \mid y)$$

For fun: most frequent bigrams?

2551888	of	the
1887475	in	the
1041011	to	the
861798	on	the
676658	and	the
648408	to	be
578806	for	the
561171	at	the
498217	in	a
479627	do	n't
455367	with	the
451460	from	the
443547	of	a
395939	that	the
362176	is	a
361879	going	to
335255	by	the
330828	as	a
319846	with	a
317431	I	think

Source: http://www.ngrams.info/download_coca.asp

Most frequent trigrams?

198630	I	do	n't
140305	one	of	the
129406	a	lot	of
117289	the	United	States
79825	do	n't	know
76782	out	of	the
75015	as	well	as
73540	going	to	be
61373	I	did	n't
61132	to	be	a

Source: http://www.ngrams.info/download_coca.asp

4-grams? 5-grams?

54647	I	do	n't	know
43766	I	do	n't	think
33975	in	the	United	States
29848	the	end	of	the
27176	do	n't	want	to

12663	I	do	n't	want	to
10663	at	the	end	of	the
8484	in	the	middle	of	the
8038	I	do	n't	know	what
6446	I	do	n't	know	if

Source: http://www.ngrams.info/download_coca.asp

Building n-grams with NLTK

```
>>> chom = 'colorless green ideas sleep furiously'.split()
>>> chom
    ['colorless', 'green', 'ideas', 'sleep', 'furiously']

>>> nltk.bigrams(chom)
    <generator object bigrams at 0x000001C432AEAA98>
>>> list(nltk.bigrams(chom))
    [('colorless', 'green'), ('green', 'ideas'), ('ideas', 'sleep'),
    ('sleep', 'furiously')]

>>> nltk.ngrams(chom, 2)
    <zip object ngrams at 0x000001C432AEAA20>
>>> list(nltk.ngrams(chom, 2))
    [('colorless', 'green'), ('green', 'ideas'), ('ideas', 'sleep'),
    ('sleep', 'furiously')]
>>> list(nltk.ngrams(chom, 3))
    [('colorless', 'green', 'ideas'), ('green', 'ideas', 'sleep'), ('ideas',
    'sleep', 'furiously')]
>>> chom3grams = list(nltk.ngrams(chom, 3))
```

`nltk.bigrams()`

`nltk.ngrams(list, n)`

These return a **generator** object.
Cast into a **list** for multiple use.

Careful with NLTK n-grams

```
>>> rtoks
    ['Rose', 'is', 'a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose', '.']
>>> nltk.ngrams(rtoks, 2)
    <zip object ngrams at 0x0A18B0C0>
>>> for gram in nltk.ngrams(rtoks, 2):
...     print(gram)
...
    ('Rose', 'is')
    ('is', 'a')
    ('a', 'rose')
    ('rose', 'is')
    ('is', 'a')
    ('a', 'rose')
    ('rose', 'is')
    ('is', 'a')
    ('a', 'rose')
    ('rose', '.')
```

`nltk.ngrams()` returns a zip object: a type of *generator*. It is not returned as a whole, but works in `for` loop, ONCE!

Feed to `nltk.FreqDist()` to obtain bigram frequency distribution.

```
>>> r2grams = nltk.ngrams(rtoks, 2)
>>> nltk.FreqDist(r2grams)
    FreqDist({('is', 'a'): 3, ('a', 'rose'): 3,
    ('rose', 'is'): 2, ('rose', '.'): 1, ('Rose',
    'is'): 1})
>>> list(r2grams)
    []
```

Been already used, r2grams is now empty!

Practice with Gettysburg

5 minutes

Process *The Gettysburg Address* (`gettysburg_address.txt`)

- Build word-level bigrams from tokens.
- How many times does the bigram ('to', 'be') occur?
- What are the top 10 most frequent bigrams?
  - Hint: feed bigrams into `nltk.FreqDist()`

`nltk.bigrams(list)`

`nltk.ngrams(list, n)`

Casting bigrams as a list,
so it is persistent

```
>>> g2grams = list(nltk.bigrams(gtoks))
>>> g2grams[-30:]
    [(',', 'shall'), ('shall', 'have'), ('have', 'a'), ('a', 'new'), ('new',
    'birth'), ('birth', 'of'), ('of', 'freedom'), ('freedom', '-'), ('-',
    'and'), ('and', 'that'), ('that', 'government'), ('government', 'of'),
    ('of', 'the'), ('the', 'people'), ('people', ','), (',', 'by'), ('by',
    'the'), ('the', 'people'), ('people', ','), (',', 'for'), ('for', 'the'),
    ('the', 'people'), ('people', ','), (',', 'shall'), ('shall', 'not'),
    ('not', 'perish'), ('perish', 'from'), ('from', 'the'), ('the', 'earth'),
    ('earth', '.')]
>>> g2gramfd = nltk.FreqDist(g2grams)
>>> g2gramfd[('to', 'be')]
    2
>>> g2gramfd.most_common(10)
    [(('nation', ','), 4), (('to', 'the'), 3), (('.', 'It'), 3), (('It',
    'is'), 3), ((',', 'we'), 3), (('we', 'can'), 3), (('can', 'not'), 3),
    (('-', 'that'), 3), (('the', 'people'), 3), (('people', ','), 3)]
```

Large-scale data found on the web

- The Internet is full of pre-compiled data files.

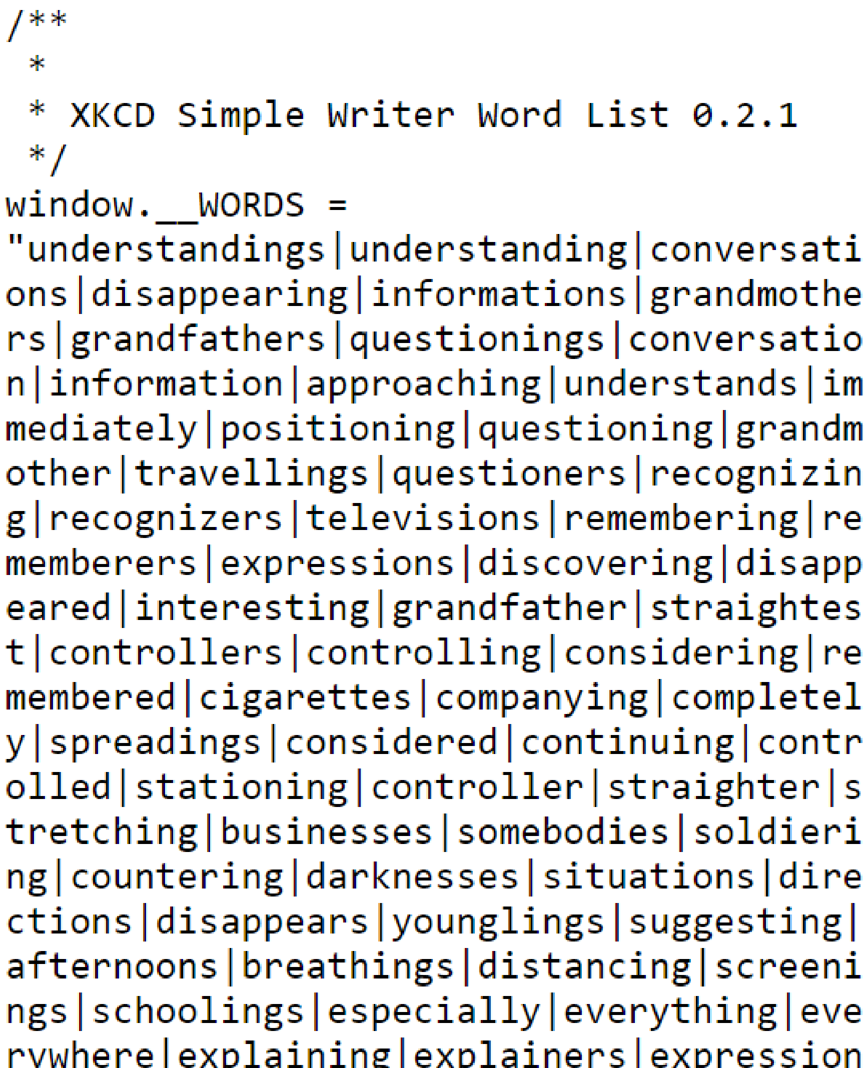
- Peter Norvig's Natural Language Corpus Data
  - https://norvig.com/ngrams/
  - Unigram frequency: `count_1w.txt`
  - Bigram frequency: `count_2w.txt`

    ← How do they look?
  - Common spelling errors: `spell-errors.txt`
  - ENABLE word list (179K words): `enable1.txt`

← Let's process and use them! HOW?

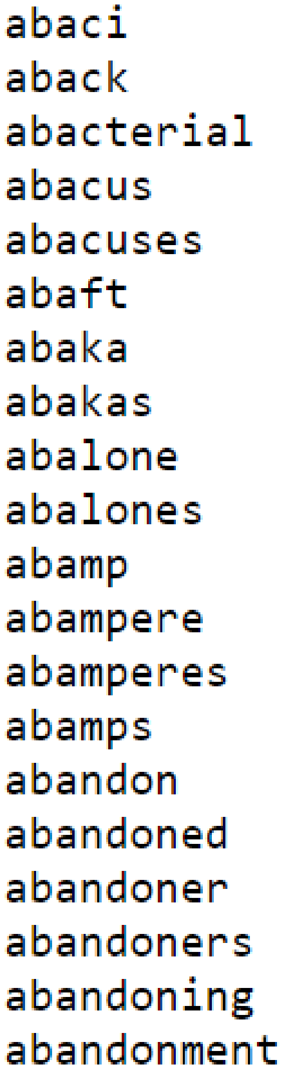
Norvig's data: word lists

- words.js



```
/**
 *
 * XKCD Simple Writer Word List 0.2.1
 */
window.__WORDS =
"understandings|understanding|conversati
ons|disappearing|informations|grandmothe
rs|grandfathers|questionings|conversatio
n|information|approaching|understands|im
mediately|positioning|questioning|grandm
other|travellings|questioners|recognizin
g|recognizers|televisions|remembering|re
memberers|expressions|discovering|disapp
eared|interesting|grandfather|straightes
t|controllers|controlling|considering|re
membered|cigarettes|companying|completel
y|spreadings|considered|continuing|contr
olled|stationing|controller|straighter|s
tretching|businesses|somebodies|soldieri
ng|countering|darknesses|situations|dire
ctions|disappears|younglings|suggesting|
afternoons|breathings|distancing|screeni
ngs|schoolings|especially|everything|eve
rywhere|explaining|explainers|expression
```

- enable1.txt



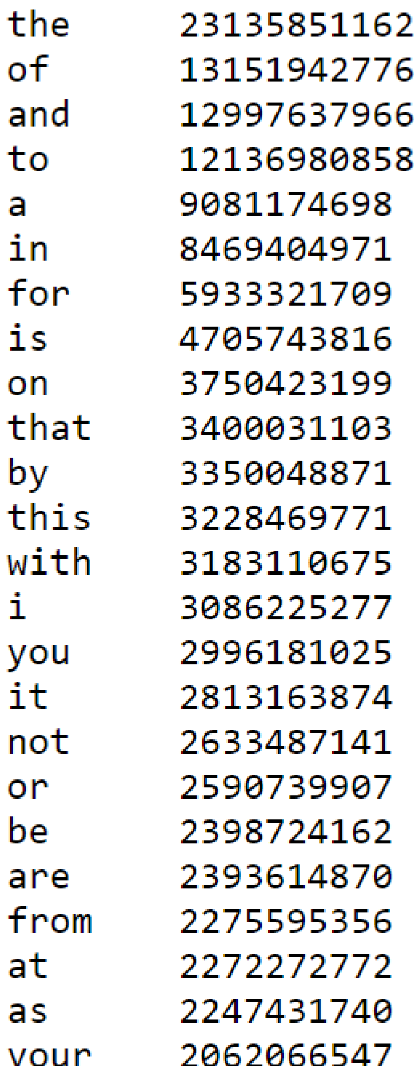
```
abaci
aback
abacterial
abacus
abacuses
abaft
abaka
abakas
abalone
abalones
abamp
abampere
abamperes
abamps
abandon
abandoned
abandoner
abandoners
abandoning
abandonment
```

What are they?

How big?

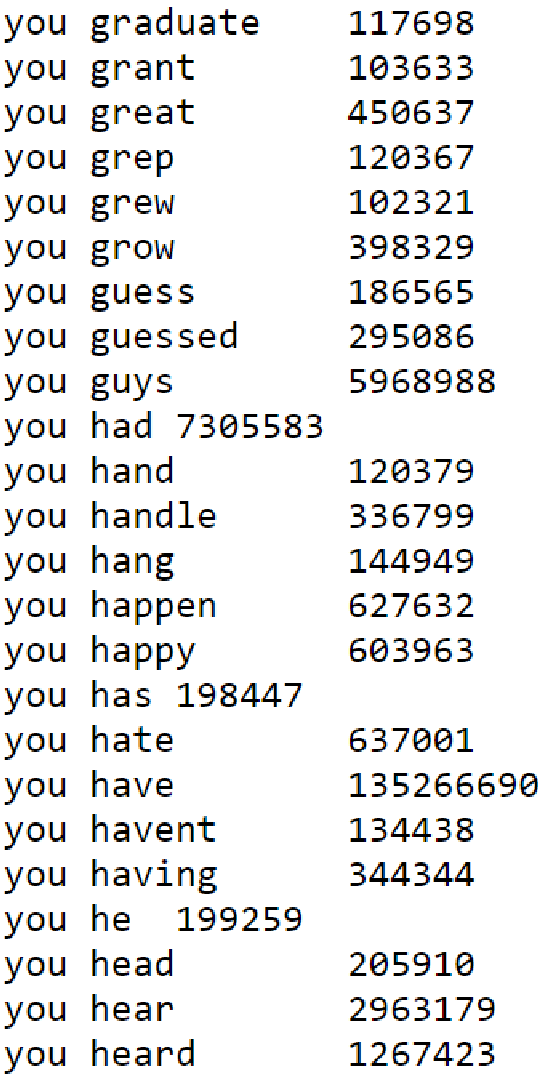
Norvig's data: 1- & 2-grams

- count_1w.txt



```
the     23135851162
of      13151942776
and     12997637966
to      12136980858
a       9081174698
in      8469404971
for     5933321709
is      4705743816
on      3750423199
that    3400031103
by      3350048871
this    3228469771
with    3183110675
i       3086225277
you     2996181025
it      2813163874
not     2633487141
or      2590739907
be      2398724162
are     2393614870
from    2275595356
at      2272272772
as      2247431740
your    2062066547
```

- count_2w.txt



```
you graduate    117698
you grant       103633
you great       450637
you grep        120367
you grew        102321
you grow        398329
you guess       186565
you guessed     295086
you guys        5968988
you had 7305583
you hand        120379
you handle      336799
you hang        144949
you happen      627632
you happy       603963
you has 198447
you hate        637001
you have        135266690
you havent      134438
you having      344344
you he  199259
you head        205910
you hear        2963179
you heard       1267423
```

Where do they come from?

A list of English words

5 minutes

- Download the ENABLE word list, posted on Norvig's site:
  - https://norvig.com/ngrams/
- Open the file and make a word list:

```
>>> f = open('enable1.txt')
>>> txt = f.read()
>>> f.close()
>>> wlist = txt.split()
>>> print(wlist[:100])
    ['aa', 'aah', 'aahed', 'aahing', 'aahs', …
    'abaka', 'abakas', 'abalone', 'abalones', …
```

enable1.txt

```
…
abaka
abakas
abalone
abalones
…
```

- How many words are there?
- Is "phonetician" in there? How about "syntactician"?
- What are top 10 longest words? How long are they?
- "Most words are 9 characters or longer." True or False?

← Use list comprehension!

Fun with ENABLE list

```
>>> wlist[-10:]
    ['zymology', 'zymosan', 'zymosans', 'zymoses', 'zymosis', 'zymotic',
    'zymurgies', 'zymurgy', 'zyzzyva', 'zyzzyvas']

>>> sorted(wlist, key=len, reverse=True)[:10]
    ['ethylenediaminetetraacetates', 'electroencephalographically',
    'ethylenediaminetetraacetate', 'immunoelectrophoretically',
    'phosphatidylethanolamines', 'dichlorodifluoromethanes',
    'electrocardiographically', 'electroencephalographers',
    'electroencephalographies', 'intercomprehensibilities']

>>> for w in sorted(wlist, key=len, reverse=True)[:10]:
...     print(w, len(w))
...
    ethylenediaminetetraacetates 28
    electroencephalographically 27
    ethylenediaminetetraacetate 27
    immunoelectrophoretically 25
```

List-comprehending English words

- Syntax: `[f(x) for x in mylist]`

"Most words are 9 characters or longer."
← True or False?

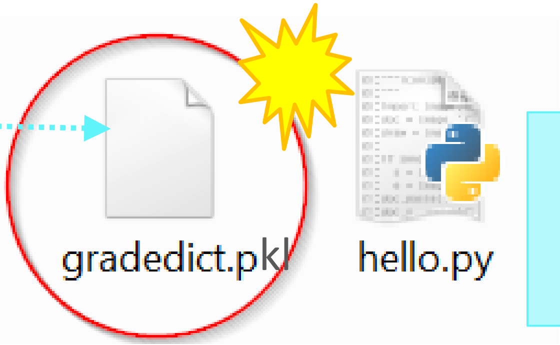
```
>>> TorF = [len(x) >=9 for x in wlist]
>>> TorF[:20]
    [False, False, False, False, False, False, False,
    False, False, False, True, False, True, False, False,
    False, False, False, True, False]
>>> TorF.count(True)
    92452
>>> TorF.count(False)
    80368
>>>
```

TorF is a list of True/False on word x being at least 9 characters long

Saving your Python data: pickling

- Pickling:

```
>>> grades = {'Bart':75, 'Lisa':98, 'Milhouse':80, 'Nelson':65}
>>> import pickle
>>> f = open('gradedict.pkl', 'wb')
>>> pickle.dump(grades, f, -1)
>>> f.close()
```



A new file is created in your CWD

- Unpickling later:

```
>>> import pickle
>>> f = open('gradedict.pkl', 'rb')
>>> mydict = pickle.load(f)
>>> f.close()
>>> print(mydict)
    {'Bart':75, 'Lisa':98, 'Milhouse':80, 'Nelson':65}
```

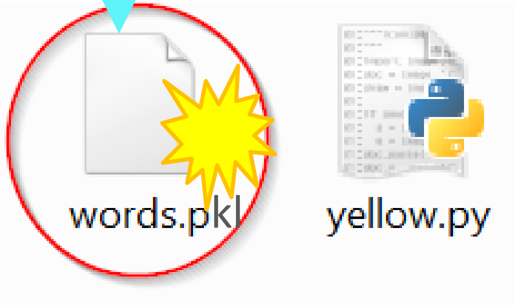
Let's save our Enable word list as a pickle file.

Pickling and unpickling

3 minutes

```
>>> import pickle
>>> f = open('words.pkl', 'wb')
>>> pickle.dump(wlist, f, -1)
>>> f.close()
==================================== RESTART ==================
>>> import pickle
>>> f = open('words.pkl', 'rb')
>>> wds = pickle.load(f)
>>> f.close()
>>> len(wds)
    172820
>>> wds[:10]
    ['aa', 'aah', 'aahed', 'aahing', 'aahs', 'aal', 'aalii',
    'aaliis', 'aals', 'aardvark']
>>>
```

Verify your pickle file was created



More fun with ENABLE list

- How many words have 'wkw' in them?
- Any word that begins with and ends with 'k'?
- Any word that has 'q' in it but no 'u'?

🡐Involves pattern matching.

🡐This type of tasks are commonly solved through **regular expressions**. (We will learn this later.)

🡐Handy solution for now: **list comprehension** as a filtering tool!

List comprehension: transformation & filtering

- Syntax: `[f(x) for x in mylist if ...]`

```
>>> mary = 'Mary had a little lamb'.split()
>>> mary
    ['Mary', 'had', 'a', 'little', 'lamb']

>>> [w for w in mary]
    ['Mary', 'had', 'a', 'little', 'lamb']

>>> [w for w in mary if len(w) >3]
    ['Mary', 'little', 'lamb']
>>> [w for w in mary if 'a' in w]
    ['Mary', 'had', 'a', 'lamb']

>>> [w.upper() for w in mary]
    ['MARY', 'HAD', 'A', 'LITTLE', 'LAMB']
>>> [len(w) for w in mary]
    [4, 3, 1, 6, 4]
```

Same as `mary`

Filter in only those elements that meet a condition

Transform each element in list

Try it out

2 minutes

- Syntax: `[f(x) for x in mylist if ...]`

```
>>> [x for x in wlist if 'wkw' in x]
                              ??

>>> [x for x in wlist if      ??      ]
    ['electroencephalographically', 'ethylenediaminetetraacetate',
    'ethylenediaminetetraacetates', 'immunoelectrophoretically',
    'phosphatidylethanolamines']

>>> [x for x in wlist if                ??                ]
    ['xerographically', 'xeroradiographies', 'xeroradiography']
```

Words that have 'wkw'

Words that are 25+ chars

Words that are 15+ chars and start with 'x'

Try it out

2 minutes

- Syntax: `[f(x) for x in mylist if ...]`

```
>>> [x for x in wlist if 'wkw' in x]
    ['awkward', 'awkwarder', 'awkwardest', 'awkwardly',
    'awkwardness', 'awkwardnesses', 'hawkweed', 'hawkweeds']

>>> [x for x in wlist if len(x) >=25]
    ['electroencephalographically', 'ethylenediaminetetraacetate',
    'ethylenediaminetetraacetates', 'immunoelectrophoretically',
    'phosphatidylethanolamines']

>>> [x for x in wlist if len(x) >=15 and x.startswith('x')]
    ['xerographically', 'xeroradiographies', 'xeroradiography']
```

Words that have 'wkw'

Words that are 25+ chars

Words that are 15+ chars and start with 'x'

Try it out

2 minutes

- Syntax: `[f(x) for x in mylist if ...]`

```
>>> [w for w in wlist if          ??          ]
                       ??

>>> [w for w in wlist if              ??
                       ??
    ??  ]
    [            ??             ]

>>> [w for w in wlist if            ??           ]
    [              ??              ]
```

Words starting with 'lingui'

Words that are 7+ characters and do not have a 'vowel'

Anagrams of ‘cried’

Try it out

2 minutes

- Syntax: `[f(x) for x in mylist if ...]`

```
>>> [w for w in wlist if w.startswith('lingui')]
    ['linguine', 'linguines', 'linguini', 'linguinis',
    'linguist', 'linguistic', 'linguistical',
    'linguistically', 'linguistician', 'linguisticians',
    'linguistics', 'linguists']
```

Words starting with 'lingui'

```
>>> [w for w in wlist if len(w) >=7 and 'a' not in w and 'e'
    not in w and 'i' not in w and 'o' not in w and 'u' not
    in w]
    ['glycyls', 'rhythms', 'tsktsks']
```

Words that are 7+ characters and do not have a 'vowel'

```
>>> [w for w in wlist if sorted(w) == sorted('cried')]
    ['cider', 'cried', 'dicer', 'riced']
```

Anagrams of ‘cried’

Wrap-up

- Exercise #4 out
  - Make sure to study the ANSWER KEY! Don't let your not-so-good Python habits stick!
- Next class (Thu):
  - Conditional probability, conditional frequency distribution
  - Bigrams as conditional frequency distribution
- Review the NLTK Book, chapters 1 through 3.