Lecture 4: Unicode, spell checkers, more NLTK

Ling 1330/2330 Computational Linguistics
Na-Rae Han, 9/7/2023

Objectives

- Unicode wrap up

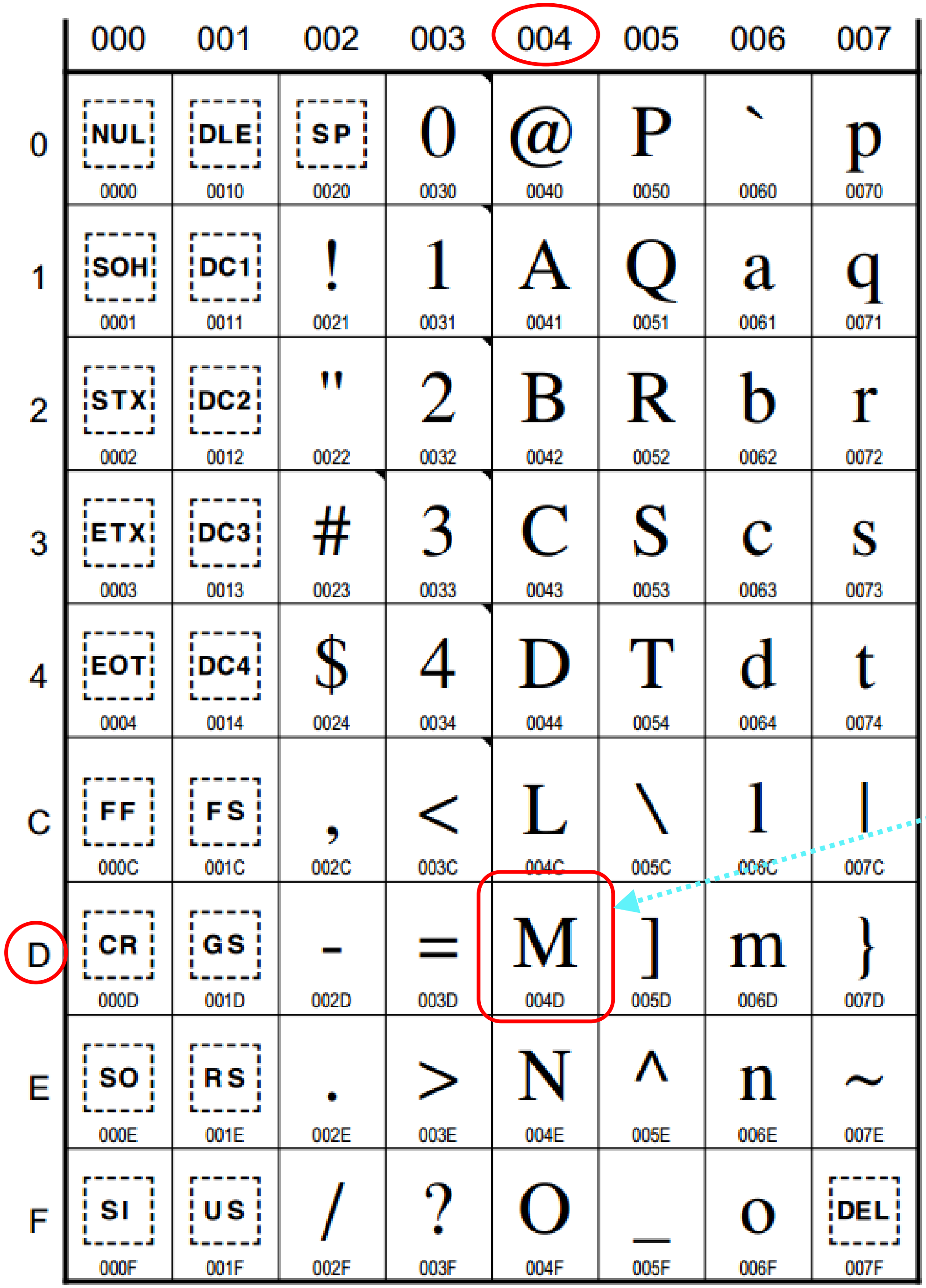
- L&C ch.2: Writers aids, spell checkers
  - Discuss design aspects and challenges in building "writers' aids" applications
  - Types of "writers' aids" utilities
  - Types of errors
  - Spell checkers
    - Edit distance

- More on NLTK: text processing
  - NLTK functions
  - List comprehension (part 1)

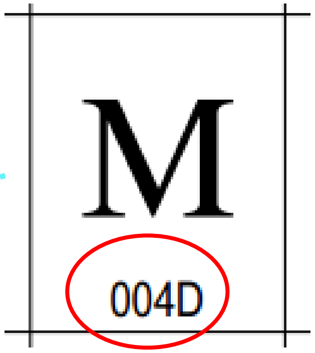
A look at Unicode chart

- How to find your Unicode character:
  - https://www.unicode.org/standard/where/
  - https://www.unicode.org/charts/

- Basic Latin (ASCII)
  - https://www.unicode.org/charts/PDF/U0000.pdf



	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F



Code point for *M*.
But "**004D**"?

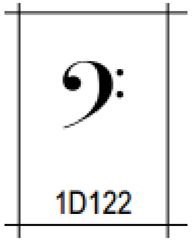
Another representation: hexadecimal

Hexadecimal (hex) = base-16

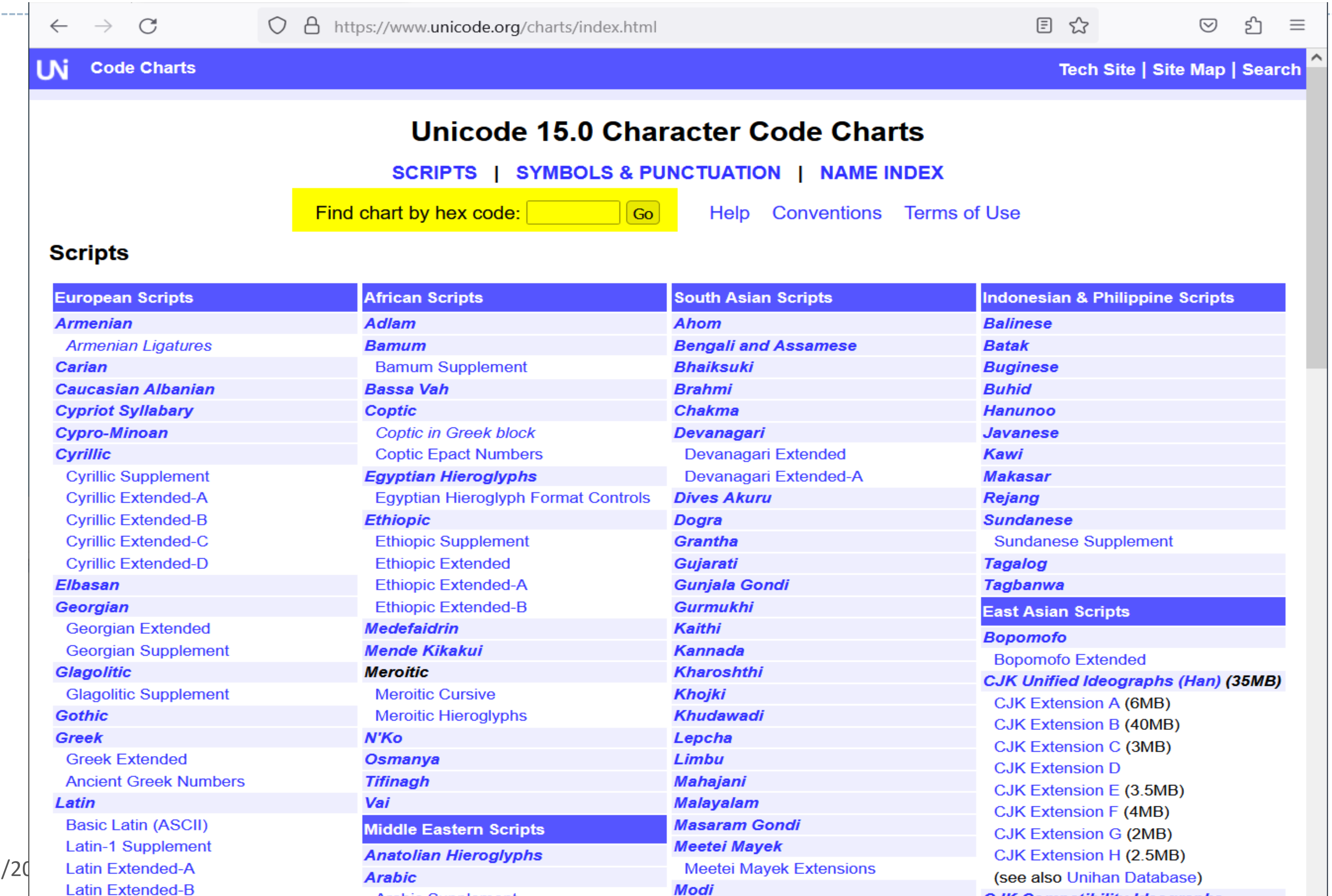
- Utilizes 16 characters: 0 1 2 3 4 5 6 7 8 9 A B C D E F
- Designed for human readability & easy byte conversion
  - 2^4=16: 1 hexadecimal digit is equivalent to 4 bits
  - 1 byte (=8 bits) is encoded with just 2 hex chars!

Letter	Base-10 (decimal)	Base-2 (binary)	Base-16 (hex)
M	77	0000 0000 0100 1101	**004D**

  - Unicode characters are usually referenced by their hexadecimal code
  - Lower-number characters go by their 4-char hex codes (2 bytes), e.g. **U+004D** ("M", U+ designates Unicode)
  - Higher-number characters go by 5 or 6 hex codes, e.g. **U+1D122** (https://www.unicode.org/charts/PDF/U1D100.pdf)



Looking up Unicode by hex code



https://www.unicode.org/charts/index.html

Code Charts

Tech Site | Site Map | Search

Unicode 15.0 Character Code Charts

SCRIPTS | SYMBOLS & PUNCTUATION | NAME INDEX

Find chart by hex code: Go

Help Conventions Terms of Use

Scripts

European Scripts

- ***Armenian***
  - *Armenian Ligatures*
- ***Carian***
- ***Caucasian Albanian***
- ***Cypriot Syllabary***
- ***Cypro-Minoan***
- ***Cyrillic***
  - Cyrillic Supplement
  - Cyrillic Extended-A
  - Cyrillic Extended-B
  - Cyrillic Extended-C
  - Cyrillic Extended-D
- ***Elbasan***
- ***Georgian***
  - Georgian Extended
  - Georgian Supplement
- ***Glagolitic***
  - Glagolitic Supplement
- ***Gothic***
- ***Greek***
  - Greek Extended
  - Ancient Greek Numbers
- ***Latin***
  - Basic Latin (ASCII)
  - Latin-1 Supplement
  - Latin Extended-A
  - Latin Extended-B

African Scripts

- ***Adlam***
- ***Bamum***
  - Bamum Supplement
- ***Bassa Vah***
- ***Coptic***
  - *Coptic in Greek block*
  - Coptic Epact Numbers
- ***Egyptian Hieroglyphs***
  - Egyptian Hieroglyph Format Controls
- ***Ethiopic***
  - Ethiopic Supplement
  - Ethiopic Extended
  - Ethiopic Extended-A
  - Ethiopic Extended-B
- ***Medefaidrin***
- ***Mende Kikakui***
- ***Meroitic***
  - Meroitic Cursive
  - Meroitic Hieroglyphs
- ***N'Ko***
- ***Osmanya***
- ***Tifinagh***
- ***Vai***

Middle Eastern Scripts

- ***Anatolian Hieroglyphs***
- ***Arabic***

South Asian Scripts

- ***Ahom***
- ***Bengali and Assamese***
- ***Bhaiksuki***
- ***Brahmi***
- ***Chakma***
- ***Devanagari***
  - Devanagari Extended
  - Devanagari Extended-A
- ***Dives Akuru***
- ***Dogra***
- ***Grantha***
- ***Gujarati***
- ***Gunjala Gondi***
- ***Gurmukhi***
- ***Kaithi***
- ***Kannada***
- ***Kharoshthi***
- ***Khojki***
- ***Khudawadi***
- ***Lepcha***
- ***Limbu***
- ***Mahajani***
- ***Malayalam***
- ***Masaram Gondi***
- ***Meetei Mayek***
  - Meetei Mayek Extensions
- ***Modi***

Indonesian & Philippine Scripts

- ***Balinese***
- ***Batak***
- ***Buginese***
- ***Buhid***
- ***Hanunoo***
- ***Javanese***
- ***Kawi***
- ***Makasar***
- ***Rejang***
- ***Sundanese***
  - Sundanese Supplement
- ***Tagalog***
- ***Tagbanwa***

East Asian Scripts

- ***Bopomofo***
  - Bopomofo Extended
- ***CJK Unified Ideographs (Han)*** **(35MB)**
  - CJK Extension A (6MB)
  - CJK Extension B (40MB)
  - CJK Extension C (3MB)
  - CJK Extension D
  - CJK Extension E (3.5MB)
  - CJK Extension F (4MB)
  - CJK Extension G (2MB)
  - CJK Extension H (2.5MB)
  - (see also Unihan Database)

Are we now living in the Unicode Utopia?

- Not yet!
- Every OS supports Unicode, but some don't use it as its system-default encoding system ("code page").
- Mac OS X uses UTF-8 as its default encoding
  - Filename, paths are in UTF-8. Text files will be created in UTF-8 encoding by default.
- Windows, however, uses **CP-1252** (aka Windows-1252, aka ANSI) as the OS's default encoding system.
  - ANSI is similar to ISO-8859-1 (=Latin1) but differs in some characters, symbols (such as curly "smart" quotes).

    ‘ ’ “”

  - Be careful when handling text files: you want to check the character encoding setting, manually change to UTF-8 if needed.
  - Another issue with Windows: uses "\r\n" as new line (instead of "\n")

Writers' aids in the wild

- What types of NLP-based **writing helper utilities** are available?
  - Spell checkers
  - Grammar checkers
  - Built-in dictionaries & thesauri
  - Predictive text writing (“next word prediction”)

  - Anything else?
  - What works well and what doesn't?

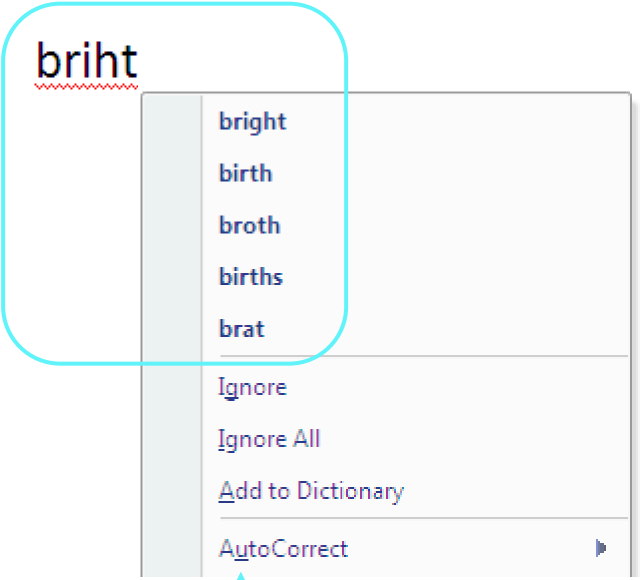
How spell checkers operate 1

Real-time spell checkers

- Spell checker detects errors as you type.
- May make **suggestions** for correction

  → Writer can accept or reject them
- Some systems **auto-correct** without your approval

  → Predominantly on mobile platform
- Must run in background, requires a "real-time" response – system must be light and fast



bright, birth, broth, births, brat

How spell checkers operate 2

- **Global spell checkers**
  - You *run* a checker on the whole document or a set region
  - System has access to wider context (the whole paragraph, etc.)
  - It finds errors and corrects them, often automatically
  - A human may or may not proofread results afterwards

- **Adaptive spell checkers**
  - "Learns" the language of the user, adjust lexicon/rules
  - **Manual**: User has the option to add or remove from the lexicon
  - **Automatic**: adjust lexicon and rules in the background. Often uses <u>context </u>and <u>statistical data</u>.

    ← Modus operandi of most mobile platform

Detection vs. correction

- There are two distinct tasks:
  - **Error detection**

    ← Simply find the misspelled words
  - **Error correction**

    ← Correct the misspelled words (or: provide suggestions)

It is EASY to tell that *briht* is a misspelled word;

But what is the CORRECT word?

bright? *birth*? *births*? *brat*?

Why not *Brit* or *brought*?

← We need a way to measure the degree of similarity between source and target words

Measure of string similarity

- How is a mistyped word related to the intended?
- Types of errors
  - **Insertion:** A letter has been added to a word
    - ex. "arguement" instead of "argument"
  - **Deletion:** A letter has been omitted from a word
    - ex. "pychology" instead of "psychology"
  - **Substitution:** One letter has been replaced by another
    - ex. "miopic" instead of "myopic"
  - **Transposition:** Two adjacent letters have been switched
    - ex. "concsious" instead of "conscious"

Minimum edit distance

- In order to rank possible spelling corrections, it is useful to calculate the **minimum edit distance** (= minimum number of operations it would take to convert *word1* to *word2*).

  ← **Edit distance**; also known as **Levenshtein distance** (without Transposition)

  - Example: briht

    bright? *birth*? *births*? *brat*? *Brit*? *brought*?

briht → bright	(1 insertion)
briht → brit	(1 deletion)
briht → birth	(2 transpositions)
briht → brunt	(2 substitutions)
briht → brat	(1 substitution + 1 deletion = 2)
briht → brought	(1 substitution + 2 insertions = 3)

NOT
2 deletions &
2 insertions!

Minimum edit distance: is that enough?

- Example: briht

bright? *birth*? *births*? *brat*? *Brit*? *brought*?

briht → bright	(1 insertion)
briht → brit	(1 deletion)
briht → birth	(2 transpositions)
briht → brunt	(2 substitutions)
briht → brat	(1 substitution + 1 deletion = 2)
briht → brought	(1 substitution + 2 insertions = 3)

- Any other considerations in ranking these candidates?
  - word frequency
  - context
  - probability of error type
  - keyboard layout

Increasingly important as spell checkers grow more intelligent

Review: Exercise 3

5 minutes

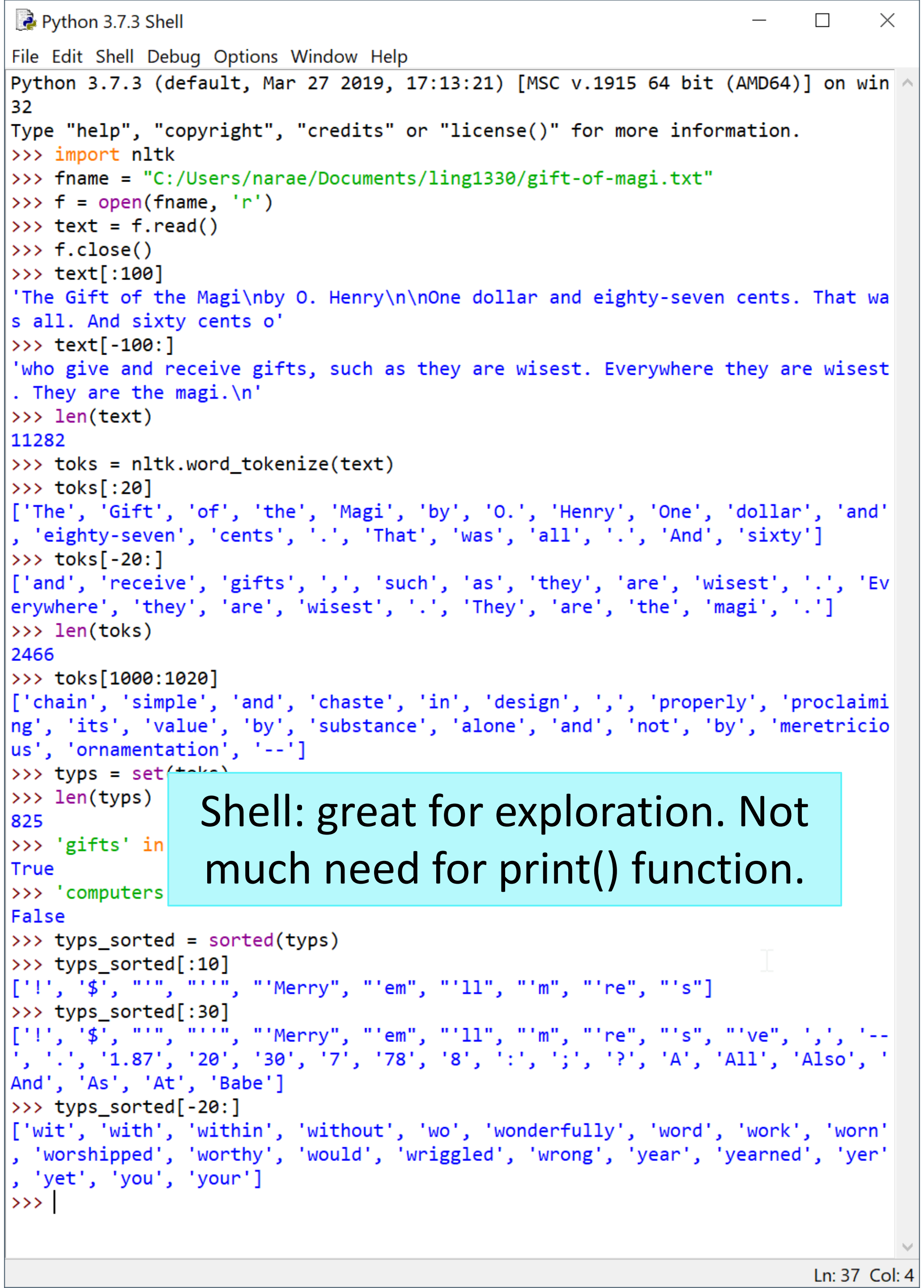
Process *The Gift of the Magi* by O. Henry

- Tokens?
- Types?
- Frequent types?
- How to sort a frequency dictionary?

You should **REVIEW** the **ANSWER KEY**! Don't be shy!

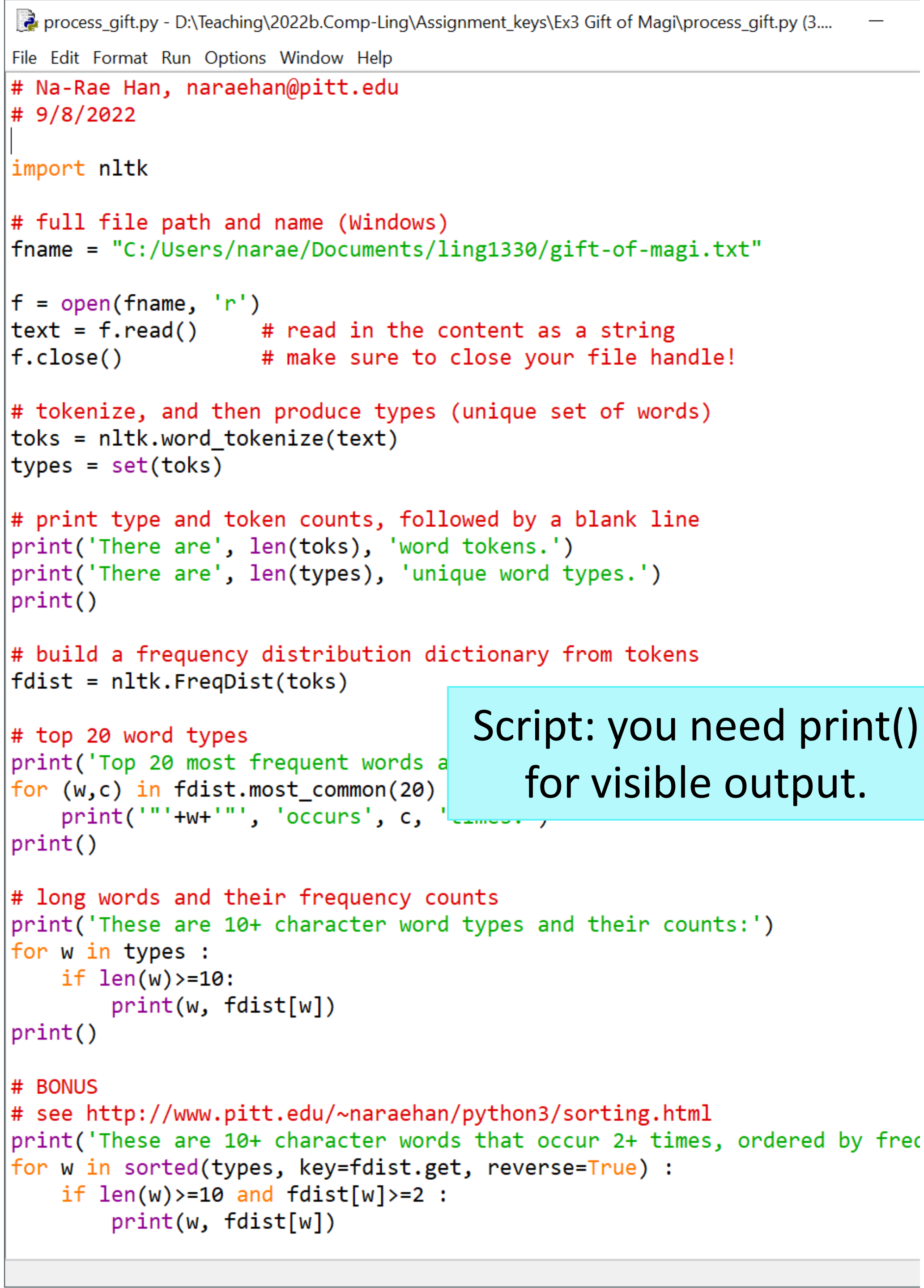
- Common pitfalls: Shell vs. Script context

  ← Related to: Returned value vs. printed output



```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> fname = "C:/Users/narae/Documents/ling1330/gift-of-magi.txt"
>>> f = open(fname, 'r')
>>> text = f.read()
>>> f.close()
>>> text[:100]
'The Gift of the Magi\nby O. Henry\n\nOne dollar and eighty-seven cents. That wa
s all. And sixty cents o'
>>> text[-100:]
'who give and receive gifts, such as they are wisest. Everywhere they are wisest
. They are the magi.\n'
>>> len(text)
11282
>>> toks = nltk.word_tokenize(text)
>>> toks[:20]
['The', 'Gift', 'of', 'the', 'Magi', 'by', 'O.', 'Henry', 'One', 'dollar', 'and'
, 'eighty-seven', 'cents', '.', 'That', 'was', 'all', '.', 'And', 'sixty']
>>> toks[-20:]
['and', 'receive', 'gifts', ',', 'such', 'as', 'they', 'are', 'wisest', '.', 'Ev
erywhere', 'they', 'are', 'wisest', '.', 'They', 'are', 'the', 'magi', '.']
>>> len(toks)
2466
>>> toks[1000:1020]
['chain', 'simple', 'and', 'chaste', 'in', 'design', ',', 'properly', 'proclaimi
ng', 'its', 'value', 'by', 'substance', 'alone', 'and', 'not', 'by', 'meretricio
us', 'ornamentation', '--']
>>> typs = set(toks)
>>> len(typs)
825
>>> 'gifts' in
True
>>> 'computers
False
>>> typs_sorted = sorted(typs)
>>> typs_sorted[:10]
['!', '$', "'", "''", "'Merry", "'em", "'ll", "'m", "'re", "'s"]
>>> typs_sorted[:30]
['!', '$', "'", "''", "'Merry", "'em", "'ll", "'m", "'re", "'s", "'ve", ',', '--
', '.', '1.87', '20', '30', '7', '78', '8', ':', ';', '?', 'A', 'All', 'Also', '
And', 'As', 'At', 'Babe']
>>> typs_sorted[-20:]
['wit', 'with', 'within', 'without', 'wo', 'wonderfully', 'word', 'work', 'worn'
, 'worshipped', 'worthy', 'would', 'wriggled', 'wrong', 'year', 'yearned', 'yer'
, 'yet', 'you', 'your']
>>> 
```

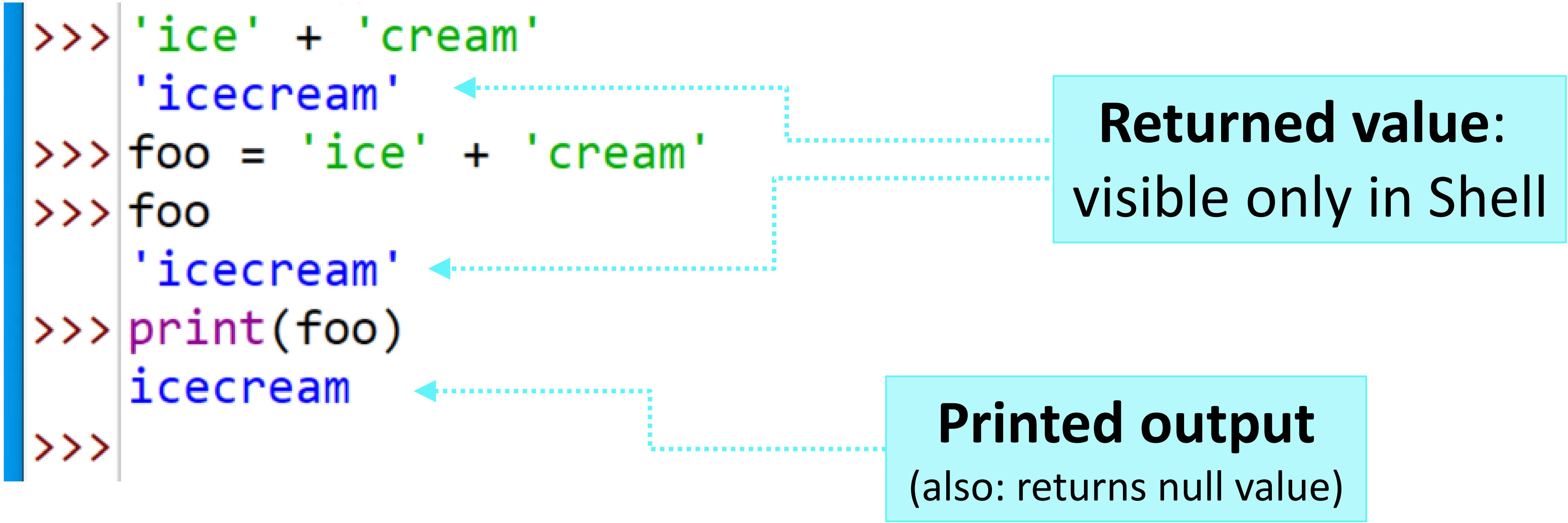
Shell: great for exploration. Not much need for print() function.


process_gift.py - D:\Teaching\2022b.Comp-Ling\Assignment_keys\Ex3 Gift of Magi\process_gift.py (3....
File Edit Format Run Options Window Help

```
# Na-Rae Han, naraehan@pitt.edu
# 9/8/2022

import nltk

# full file path and name (Windows)
fname = "C:/Users/narae/Documents/ling1330/gift-of-magi.txt"

f = open(fname, 'r')
text = f.read()      # read in the content as a string
f.close()            # make sure to close your file handle!

# tokenize, and then produce types (unique set of words)
toks = nltk.word_tokenize(text)
types = set(toks)

# print type and token counts, followed by a blank line
print('There are', len(toks), 'word tokens.')
print('There are', len(types), 'unique word types.')
print()

# build a frequency distribution dictionary from tokens
fdist = nltk.FreqDist(toks)

# top 20 word types
print('Top 20 most frequent words a
for (w,c) in fdist.most_common(20)
    print('"'+w+'"', 'occurs', c, 'times.')
print()

# long words and their frequency counts
print('These are 10+ character word types and their counts:')
for w in types :
    if len(w)>=10:
        print(w, fdist[w])
print()

# BONUS
# see http://www.pitt.edu/~naraehan/python3/sorting.html
print('These are 10+ character words that occur 2+ times, ordered by frec
for w in sorted(types, key=fdist.get, reverse=True) :
    if len(w)>=10 and fdist[w]>=2 :
        print(w, fdist[w])
```

Script: you need print() for visible output.

Returned value vs. Printed output



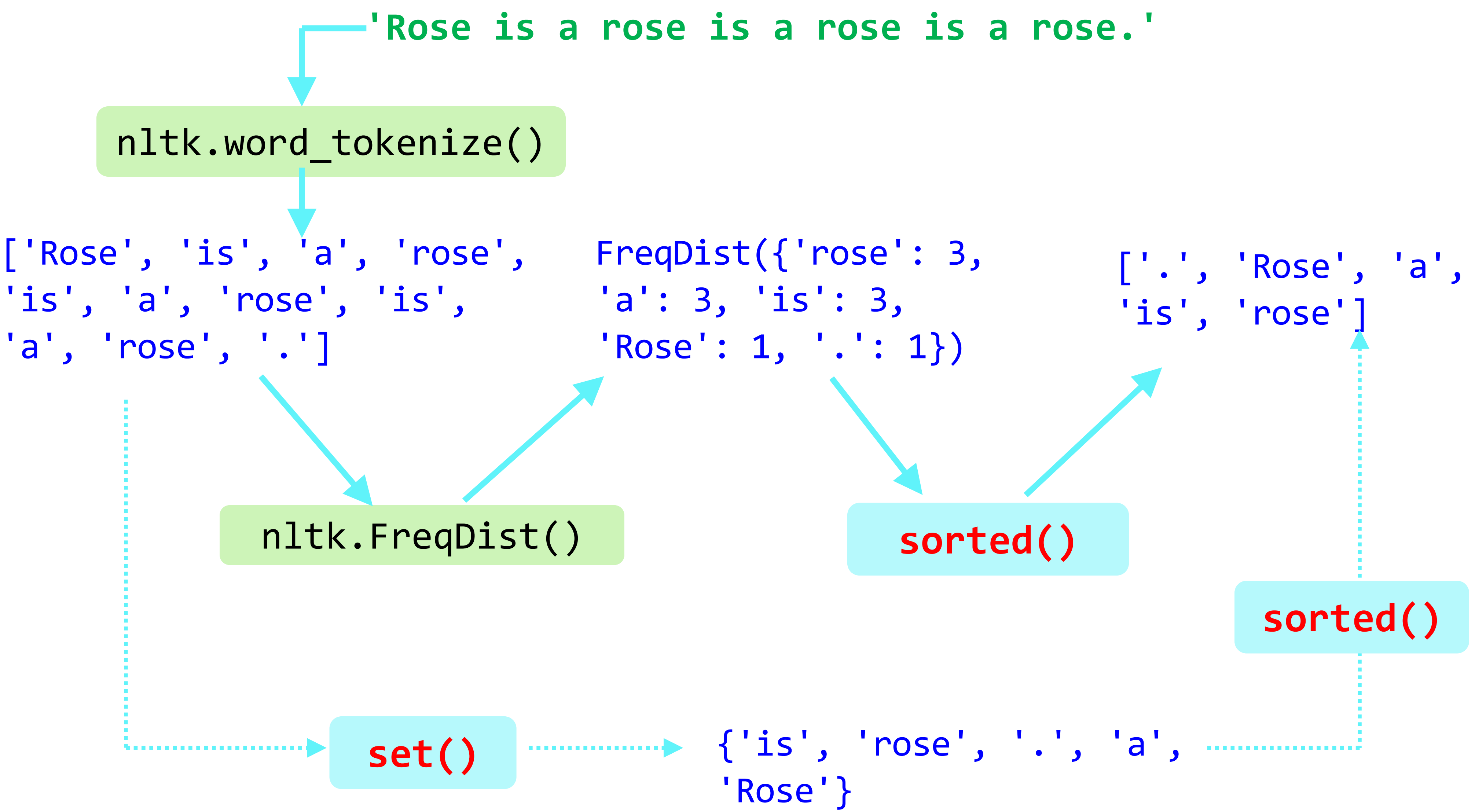
```
>>> 'ice' + 'cream'
'icecream'
>>> foo = 'ice' + 'cream'
>>> foo
'icecream'
>>> print(foo)
icecream
>>>
```

Returned value:
visible only in Shell

Printed output
(also: returns null value)

- Also see: https://sites.pitt.edu/~naraehan/python3/user_defined_functions.html

Back to NLTK: text processing pipeline



```
'Rose is a rose is a rose is a rose.'
```

```
nltk.word_tokenize()
```

```
['Rose', 'is', 'a', 'rose',
'is', 'a', 'rose', 'is',
'a', 'rose', '.']
```

```
nltk.FreqDist()
```

```
FreqDist({'rose': 3,
'a': 3, 'is': 3,
'Rose': 1, '.': 1})
```

```
sorted()
```

```
['.', 'Rose', 'a',
'is', 'rose']
```

```
set()
```

```
{'is', 'rose', '.', 'a',
'Rose'}
```

```
sorted()
```

Sentence tokenization

```
>>> foo = 'Hello, earthlings! I come in peace. Take me to your
    leader.'
>>> nltk.sent_tokenize(foo)
    ['Hello, earthlings!', 'I come in peace.', 'Take me to your
    leader.']
>>> sents = nltk.sent_tokenize(foo)
>>> sents[0]
    'Hello, earthlings!'
>>> sents[1]
    'I come in peace.'
>>> sents[-1]
    'Take me to your leader.'
>>> len(sents)
    3
```

`nltk.sent_tokenize()` takes a text string, returns a list of sentences as strings.

Total number of sentences

Practice: sentence tokenization

1 minute

```
>>> foo = 'Hello, earthlings! I come in peace. Take me to your
    leader.'
>>> nltk.sent_tokenize(foo)
    ['Hello, earthlings!', 'I come in peace.', 'Take me to your
    leader.']
>>> sents = nltk.sent_tokenize(foo)
>>> sents[0]
    'Hello, earthlings!'
>>> sents[1]
    'I come in peace.'
>>> sents[-1]
    'Take me to your leader.'
>>> len(sents)
    3
```

nltk.sent_tokenize() takes a text string, returns a list of sentences as strings.

Total number of sentences

Sentence and word tokenization

```
>>> for s in sents:
...         nltk.word_tokenize(s)
...
    ['Hello', ',', 'earthlings', '!']
    ['I', 'come', 'in', 'peace', '.']
    ['Take', 'me', 'to', 'your', 'leader', '.’]

>>> toksents = []
>>> for s in sents:
...     toksents.append(nltk.word_tokenize(s))
...
>>> toksents
    [['Hello', ',', 'earthlings', '!'], ['I', 'come', 'in', 'peace', '.'],
    ['Take', 'me', 'to', 'your', 'leader', '.']]

>>> foo
    'Hello, earthlings! I come in peace. Take me to your leader.'
>>> nltk.word_tokenize(foo)
    ['Hello', ',', 'earthlings', '!', 'I', 'come', 'in', 'peace', '.',
    'Take', 'me', 'to', 'your', 'leader', '.']
```

word-tokenizing individual sentences

A *list* of lists!

cf. a FLAT list of word tokens

Using list comprehension

```
>>> sents
    ['Hello, earthlings!', 'I come in peace.', 'Take me to your leader.']
>>> for s in sents:
...     print(s, len(s))
...
    Hello, earthlings! 18
    I come in peace. 16
    Take me to your leader. 23

>>> [len(s) for s in sents]
    [18, 16, 23]
>>> [s.upper() for s in sents]
    ['HELLO, EARTHLINGS!', 'I COME IN PEACE.', 'TAKE ME TO YOUR LEADER.']
>>> [s.split() for s in sents]
    [['Hello,', 'earthlings!'], ['I', 'come', 'in', 'peace.'], ['Take', 'me',
    'to', 'your', 'leader.']]
>>> [nltk.word_tokenize(s) for s in sents]
    [['Hello', ',', 'earthlings', '!'], ['I', 'come', 'in', 'peace', '.'],
    ['Take', 'me', 'to', 'your', 'leader', '.']]
```

List comprehension!
Better than for loop

Voila!

Practice: list comprehension

3 minutes

```
>>> sents
    ['Hello, earthlings!', 'I come in peace.', 'Take me to your leader.']
>>> for s in sents:
...     print(s, len(s))
...
    Hello, earthlings! 18
    I come in peace. 16
    Take me to your leader. 23

>>> [len(s) for s in sents]
    [18, 16, 23]
>>> [s.upper() for s in sents]
    ['HELLO, EARTHLINGS!', 'I COME IN PEACE.', 'TAKE ME TO YOUR LEADER.']
>>> [s.split() for s in sents]
    [['Hello,', 'earthlings!'], ['I', 'come', 'in', 'peace.'], ['Take', 'me',
    'to', 'your', 'leader.']]
>>> [nltk.word_tokenize(s) for s in sents]
    [['Hello', ',', 'earthlings', '!'], ['I', 'come', 'in', 'peace', '.'],
    ['Take', 'me', 'to', 'your', 'leader', '.']]
```

Syntax:

```
[f(x) for x in mylist]
```

Wrap up

- Homework #1 out
  - Spell checkers, corpus processing

- Next class (Tue):
  - Spell checkers review
  - n-gram context
  - n-gram resource on the web
  - more on list comprehension

- Review the NLTK Book, chapters 1 through 3.