Lecture 3: Unicode, Text Processing with NLTK

Ling 1330/2330 Intro to Computational Linguistics

Na-Rae Han, 9/5/2023

Objectives

- NLTK intro: text processing
  - NLTK functions
  - File IO: opening and processing a text file

- L&C ch.1: Understand the fundamentals of how language is encoded on a computer
  - Unicode!

Getting started with NLTK book

- NLTK Book, with Na-Rae's navigation panel:
  - https://sites.pitt.edu/~naraehan/ling1330/nltk_book.html
- NLTK Book, without:
  - https://www.nltk.org/book/

- Chapter 1. Language Processing and Python
  - https://www.nltk.org/book/ch01.html
- Chapter 2. Accessing Text Corpora and Language Resources
  - https://www.nltk.org/book/ch02.html

Install NLTK and NLTK data

- NLTK (Mac): https://sites.pitt.edu/~naraehan/python3/faq.html#Q-install-nltk-mac
- NLTK (Win): https://sites.pitt.edu/~naraehan/python3/faq.html#Q-install-nltk-win
- NLTK data: https://sites.pitt.edu/~naraehan/python3/faq.html#Q-nltk-download

- Test to confirm everything works:

```
>>> import nltk
>>> nltk.corpus.brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
>>> import numpy
>>> import matplotlib
>>> import bs4
>>>
```

NLTK's tokenizer

```
>>> import nltk
>>> nltk.word_tokenize('Hello, world!’)
    ['Hello', ',', 'world', '!']
>>> nltk.word_tokenize("I haven't seen Star Wars.")
    ['I', 'have', "n't", 'seen', 'Star', 'Wars', '.']
>>> nltk.word_tokenize("It's 5 o'clock. Call Ted...!")
    ['It', "'s", '5', "o'clock", '.', 'Call', 'Ted', '...', '!']

>>> rose = 'Rose is a rose is a rose is a rose.'
>>> nltk.word_tokenize(rose)
    ['Rose', 'is', 'a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose', '.']
>>> rtoks = nltk.word_tokenize(rose)
>>> rtoks
    ['Rose', 'is', 'a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose', '.']
>>> type(rtoks)
    <class 'list'>
```

nltk.word_tokenize()

No lowercasing, *n't*, *o'clock* a word

Good-old list type.

NLTK and frequency counts

```
>>> rfreq = nltk.FreqDist(rtoks)
>>> rfreq
    FreqDist({'rose': 3, 'a': 3, 'is': 3, 'Rose': 1, '.': 1})

>>> rfreq['is']
    3
>>> rfreq.keys()
    dict_keys(['Rose', 'is', 'a', 'rose', '.'])
>>> rfreq.values()
    dict_values([1, 3, 3, 3, 1])
>>> rfreq.items()
    dict_items([('Rose', 1), ('is', 3), ('a', 3), ('rose', 3), ('.', 1)])

>>> sorted(rfreq)
    ['.', 'Rose', 'a', 'is', 'rose']
>>> type(rfreq)
    <class 'nltk.probability.FreqDist'>
```

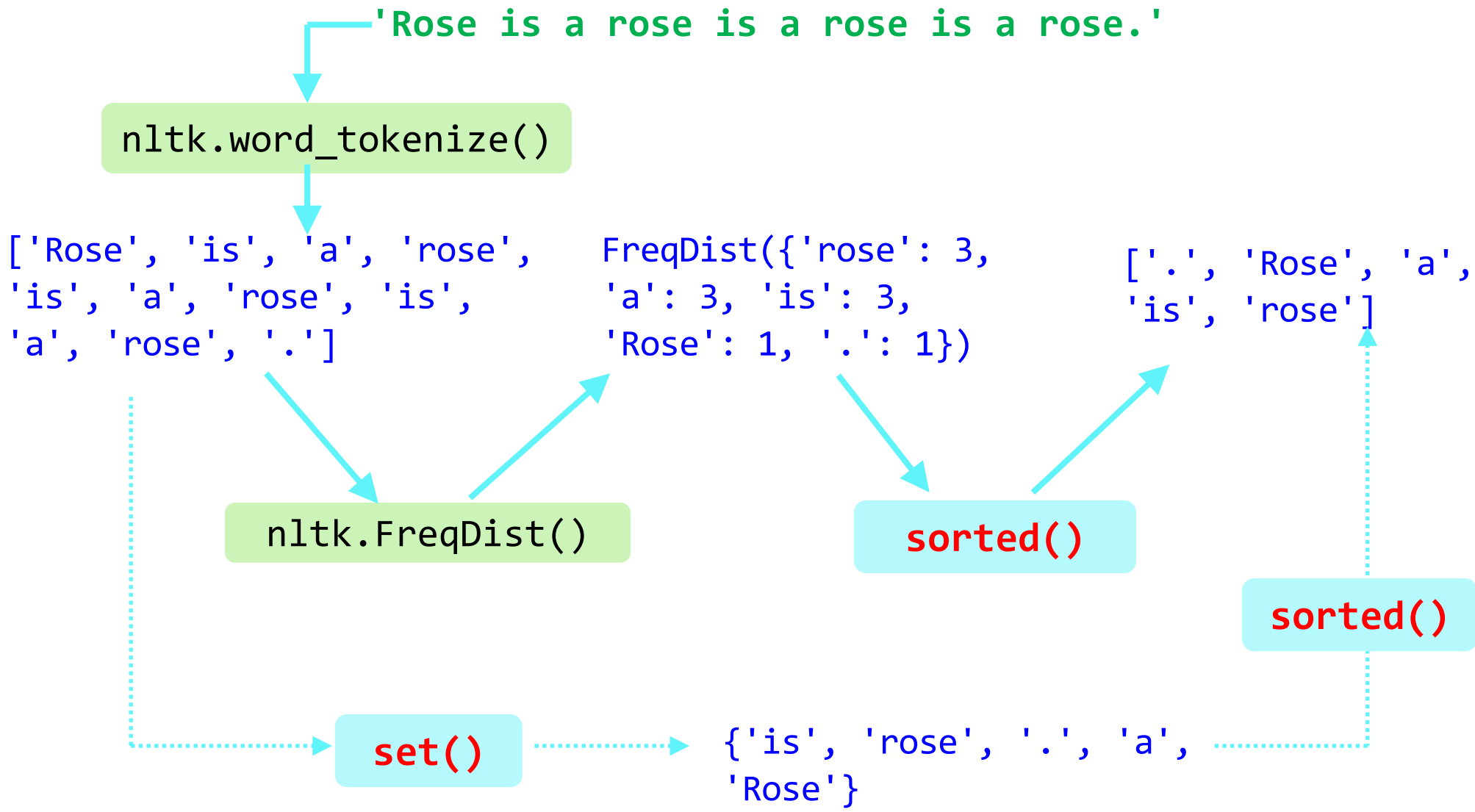
`nltk.FreqDist()`

`FreqDist` works very much like a dictionary…

word **types**

… but it's NLTK's own custom data type!

NLTK's functions, text processing pipeline



FreqDist can do much more

```
>>> dir(rfreq)
    ['B', 'N', 'Nr', '__add__', '__and__', '__class__', …  'clear', 'copy',
    'elements', 'freq', 'fromkeys', 'get', 'hapaxes', 'items', 'keys', 'max',
    'most_common', 'pformat', 'plot', 'pop', 'popitem', 'pprint', 'r_Nr',
    'setdefault', 'subtract', 'tabulate', 'unicode_repr', 'update', 'values']
>>> rfreq.hapaxes()
    ['Rose', '.']
>>> rfreq.tabulate()
    rose    a    is Rose    .
       3    3     3    1    1
>>> rfreq.most_common(2)
    [('a', 3), ('is', 3)]
>>> rfreq['platypus']
    0
>>> rfreq.plot()
```

nltk.FreqDist comes with additional handy methods!

No "key not found" error! Defaults to 0.

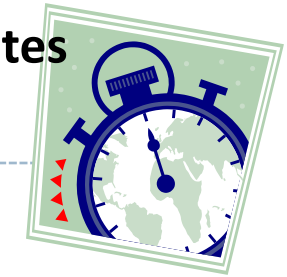
Graph window pops up

```
>>> rfreq.max()
    'a'
>>> rfreq['is']
    3
>>> rfreq.freq('is')
    0.2727272727272727
```

Relative frequency (= probability)

Practice: Gettysburg Address

15 minutes



Process the famous Gettysburg Address:

https://sites.pitt.edu/~naraehan/python3/gettysburg_address.txt

Learning Python:

- Python 3 Notes
- FAQ
- Text samples (for copy-pasting)
- Short text files: mary-short.txt, tale.txt, how-do-i.txt, gettysburg_address.txt, gift-of-magi.txt

- Tasks:
  - Save the text file in your usual script directory
  - Open the file in IDLE shell, read in the string content, then close. Examine the raw text: **how many characters**?
  - Tokenize, and then examine: how many **word tokens**? How many unique **word types**?
  - Build a frequency distribution of word tokens. How many **tokens of 'people'**? What are the **most common word types**?

```
nltk.word_tokenize()
sorted()
```

```
nltk.FreqDist()
.most_common()
.tabulate()
```

Python 3.8.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> fname = 'C:/Users/narae/Documents/ling1330/gettysburg_address.txt'
>>> myfile = open(fname, 'r')
>>> gtxt = myfile.read()
>>> myfile.close()
>>> gtxt[:100]
'Four score and seven years ago our fathers brought forth on this continent a new nation, conceiv
ed i'
>>> gtxt[-100:]
' and that government of the people, by the people, for the people, shall not perish from the ear
th.\n'
>>> len(gtxt)
1465
>>> gtoks = nltk.word_tokenize(gtxt)
>>> gtoks[:10]
['Four', 'score', 'and', 'seven', 'years', 'ago', 'our', 'fathers', 'brought', 'forth']
>>> gtoks[-10:]
['the', 'people', ',', 'shall', 'not', 'perish', 'from', 'the', 'earth', '.']
>>> len(gtoks)
309
>>> gfreq = nltk.FreqDist(gtoks)
>>> len(gfreq)
145
>>> gfreq['the']
9
>>> gfreq['penguin']
0
>>> dir(gfreq)
['B', 'N', 'Nr', '_N', '__add__', '__and__', '__class__', '__contains__', '__delattr__', '__delit
em__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '_
_getitem__', '__gt__', '__hash__', '__iadd__', '__iand__', '__init__', '__init_subclass__', '__io
r__', '__isub__', '__iter__', '__le__', '__len__', '__lt__', '__missing__', '__module__', '__ne__
', '__neg__', '__new__', '__or__', '__pos__', '__reduce__', '__reduce_ex__', '__repr__', '__rever
sed__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__
weakref__', '_cumulative_frequencies', '_keep_positive', 'clear', 'copy', 'elements', 'freq', 'fr
omkeys', 'get', 'hapaxes', 'items', 'keys', 'max', 'most_common', 'pformat', 'plot', 'pop', 'popi
tem', 'pprint', 'r_Nr', 'setdefault', 'subtract', 'tabulate', 'update', 'values']
>>> gfreq.most_common(20)
[(',', 24), ('that', 13), ('.', 10), ('the', 9), ('to', 8), ('we', 8), ('here', 8), ('a', 7), ('a
nd', 6), ('nation', 5), ('can', 5), ('of', 5), ('have', 5), ('for', 5), ('not', 5), ('this', 4),
('in', 4), ('dedicated', 4), ('-', 4), ('are', 3)]
>>>
```

Ln: 32 Col: 4

File referencing using the full path + name (Windows)

More on File Path and CWD: https://sites.pitt.edu/~naraehan/python3/file_path_cwd.html

File IO: file path vs. CWD

Python 3.8.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.getcwd()
'C:\\Users\\narae\\Documents\\ling1330'
>>> myfile = open('gettysburg_address.txt')
>>> gtxt = myfile.read()
>>> myfile.close()
>>> gtxt[:100]
'Four score and seven years ago our fathers brought forth on this continen
new nation, conceived i'
>>> os.chdir('..')
>>> os.getcwd()
'C:\\Users\\narae\\Documents'
>>> myfile = open('ling1330/gettysburg_address.txt')
>>> gtxt[-100:]
' and that government of the people, by the people, for the
perish from the earth.\n'
>>>
```

My CWD is my script folder where the text file is.
I can reference the file with the file name only!

Hit TAB for file name completion.

If my CWD is "Documents" (one level up), then I have to start the file reference from the "ling1330" folder.

The ASCII chart

- https://en.wikipedia.org/wiki/ASCII

Decimal	Binary (7-bit)	Character
0	000 0000	(NULL)
…	…	…
35	010 0011	#
36	010 0100	&
…	…	…
48	011 0000	0
49	011 0001	1
50	011 0010	2
…	…	…

Decimal	Binary (7-bit)	Character
65	100 0001	A
66	100 0010	B
67	100 0011	C
…	…	…
97	110 0001	a
98	110 0010	b
99	110 0011	c
…	…	…
127	111 1111	(DEL)

ASCII (the American Standard Code for Information Interchange)

- The ASCII encoding scheme
  - First published in 1963
  - Uses **7-bit** code (= 128 characters) for storing English text, ranging from 0 to 127

    ← In an **8-bit (1 byte)** representation, the highest bit is always 0
  - **Printable characters**
    - Upper and lower case roman alphabet
    - Digits
    - Punctuation marks, symbols, and space
  - Includes **32 non-printing characters**
    - Control characters: BELL, ACKNWOLEDGE, BACKSPACE, DELETE, etc. → originally for typewriters, many obsolete now
    - *WHITESPACE* characters: TAB, LINE FEED, CARRIAGE RETURN

Practice

- What is this English text?
  - Note: byte (=8-bit) ASCII representation instead of 7-bit
  - Space provided for your convenience only!

01001000 01101001 00100001

- Answer:

Hi!

Extending ASCII: ISO-8859, etc.

- ASCII (=7 bit, 128 characters) was sufficient for encoding English. But what about characters used in other languages?
- Solution: Extend ASCII into 8-bit (=256 characters) and use the additional 128 slots for non-English characters
  - **ISO-8859**: has 16 different implementations!
    - ISO-8859-1 aka Latin-1: French, German, Spanish, etc.
    - ISO-8859-7 Greek alphabet
    - ISO-8859-8 Hebrew alphabet
  - JIS X 0208: Japanese characters

🡸 Problem: overlapping character code space.

224_{dec} means **à** in Latin-1 but **א** in ISO-8859-8!

Unicode

- A character encoding standard developed by the Unicode Consortium
- Provides a single representation for *all* world's writing systems

- "Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language."

  (https://www.unicode.org)

How big is Unicode?

- Version 15.0.0 (2022) has codes for 149,186 characters
  - Full Unicode standard uses **32 bits** (**4 bytes**) : it can represent **2^{32}** = 4,294,967,296 characters!

    ← In reality, only 21 bits are needed

- Unicode has three encoding versions
  - UTF-32 (32 bits/4 bytes): direct representation
  - UTF-16 (16 bits/2 bytes)
  - UTF-8 (8 bits/1 byte)

8-bit, 16-bit, 32-bit

- UTF-32 (32 bits/4 bytes): direct representation
- UTF-16 (16 bits/2 bytes): 2^{16}=65,536 possibilities
- UTF-8 (8 bits/1 byte): 2^8=256 possibilities

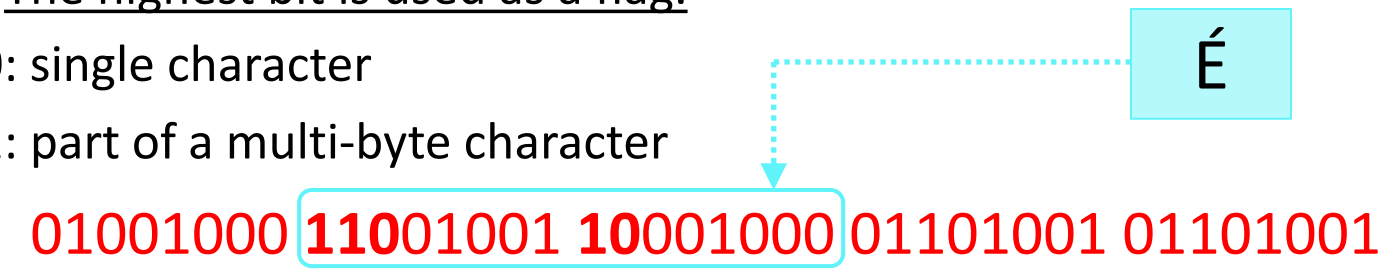
- Wait! But how do you represent all of 2^{32} (=4 billion) code points with only one byte (UTF-8: 2^8 =256 slots)?
  - You don't.
  - In reality, only 2^{21} bits are ever utilized for 144K characters.
  - UTF-8 and UTF-16 use a variable-width encoding.

- Why UTF-16 and UTF-8?
  - They are more compact (more so for certain languages, i.e., English)

Variable-width encoding

- 'H' as 1 byte (8 bits): 01001000
  cf. 'H' as 2 bytes (16 bits): 0000000001001000
  as 4 bytes (32 bits): 00000000000000000000000001001000

- UTF-8 as a variable-width encoding
  - **ASCII** characters get encoded with just **1 byte**
    ← ASCII is originally 7-bits, so the highest bit is always 0 in an 8-bit encoding
  - All other characters are encoded with **multiple (2-4) bytes**
    - How to tell? The highest bit is used as a flag.
    - Highest bit 0: single character
    - Highest bit 1: part of a multi-byte character



  - Advantage for English: 8-bit ASCII is already a valid UTF-8!

- https://www.twilio.com/docs/glossary/what-utf-8

é

("LATIN SMALL LETTER E WITH ACUTE")

U+00E9/11101001

1100001110101001

Indicates that sequence will be two bytes
Indicates that code point bits start next
Indicates a continuation byte
Padding bits
Code point bits

If lead unit starts with 1110, means two following bytes belong to multi unit

Wrap-up

- Exercise #3 out
  - Due Thursday morning, on Canvas
- Next class (Thu):
  - Spell checking
  - More on NLTK
- Review the NLTK Book, chapters 1 through 3.