Processing Raw Text
(Lab Session IV)

Dr. Jasmeet Singh
Thapar University

Introduction

- The goal of this session is to answer the following questions:
- 1. How can we write programs to access text from local files and from the Web, in order to get hold of an unlimited range of language material?
- 2. How can we split documents up into individual words and punctuation symbols, so we can carry out the same kinds of analysis we did with text corpora in earlier sessions?
- 3. How can we write programs to produce formatted output and save it in a file?

Accessing Text from the Web

- We can access the text files from the web and then apply various methods studied in previous sessions.
- In order to read text from web we need to import urlopen from urllib library as follows:
  - from urllib import urlopen (For python version 3 or more urllib package is replaced by urllib.request)
- We then access the text with the help of url as follows:
- raw=urlopen("https://textfiles.com/internet/31.txt").read()
- The variable raw contains a string of characters.
- This is the raw content of the book, including many details we are not interested in, such as whitespace, line breaks, and blank lines.

Accessing Text from the Web Contd..

- For our language processing, we want to break up the string into words and punctuation.
- This step is called **tokenization**, and it produces our familiar structure, a list of words and punctuation. It is done as follows:
- tokens = nltk.word_tokenize(raw)
- We can now create an NLTK text from this list, and can carry out all of the other linguistic processing.
- text = nltk.Text(tokens)

Dealing with HTML

- Much of the text on the Web is in the form of HTML documents.
- We can access these HTML documents also with urlopen as follows:
- html =urlopen("http://news.bbc.co.uk/2/hi/health/2284783.stm").read()
- html contains HTML content including meta tags, an image map, JavaScript, forms, and tables.
- The html content can be cleaned using bs4 package as follows:
- from bs4 import BeautifulSoup
- raw1 = BeautifulSoup(html)
- raw= raw1.get_text()
- The raw content is now tokenized and converted into text as follows:
- token=nltk.word_tokenize(raw)
- text=nltk.Text(token)

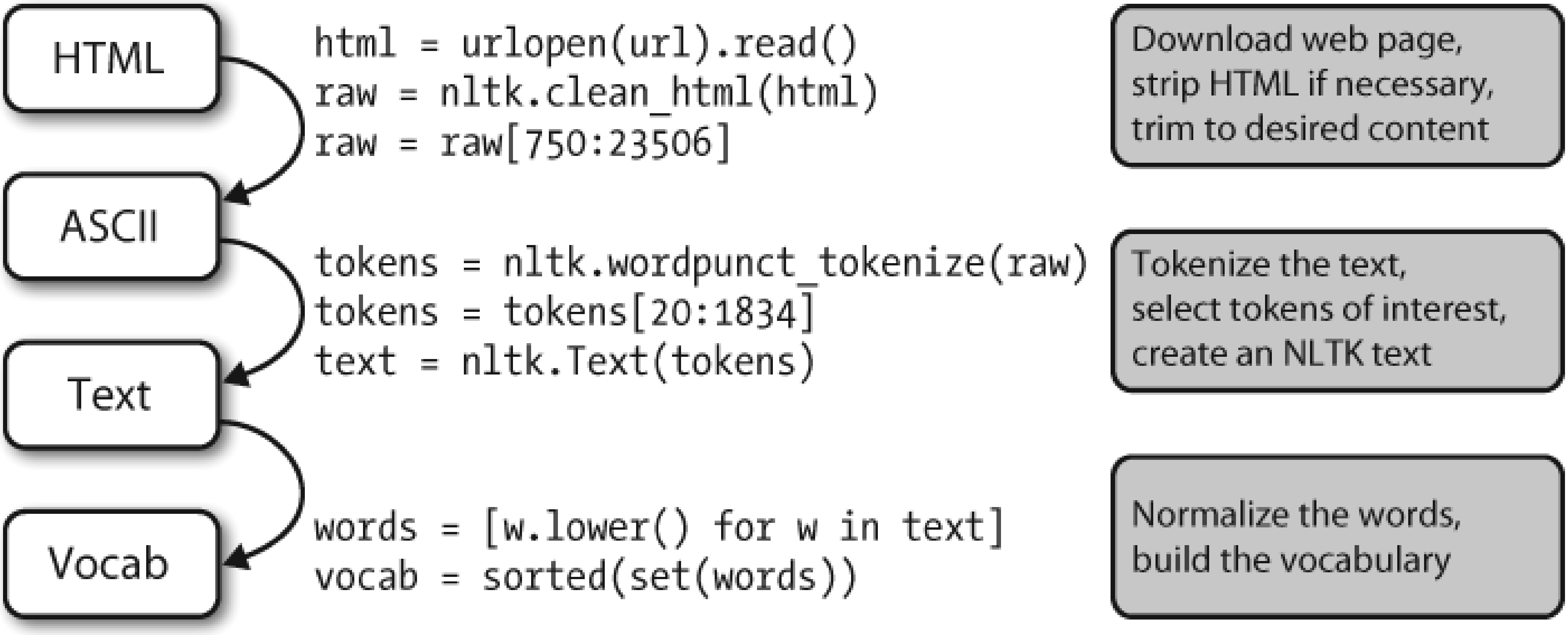
Dealing with RSS Feeds

- We can access the content of a blog, using feedparser package as follows:
- import feedparser
- llog = feedparser.parse("http://languagelog.ldc.upenn.edu/nll/?feed=atom")
- raw1=blog.entries[2].content[0].value
- raw1=BeautifulSoup(raw1)
- raw=raw1.get_text()
- token=nltk.word_tokenize(raw)
- text=nltk.Text(token)

Reading Local Files

- In order to read a local file, we need to use Python’s built-in open() function, followed by the read() method.
- f = open('document.txt')
- raw = f.read()
- token = nlrk.word_tokenize(raw)
- text=nltk.Text(token)
- Text in binary formats—such as PDF and MSWord can only be opened using specialized software. Third-party libraries such as pypdf and pywin32 provide access to these formats.
- The user input can be captured as follows:
- s = raw_input("Enter some text: ")

The NLP Pipeline



```
html = urlopen(url).read()
raw = nltk.clean_html(html)
raw = raw[750:23506]
```

Download web page,
strip HTML if necessary,
trim to desired content

```
tokens = nltk.wordpunct_tokenize(raw)
tokens = tokens[20:1834]
text = nltk.Text(tokens)
```

Tokenize the text,
select tokens of interest,
create an NLTK text

```
words = [w.lower() for w in text]
vocab = sorted(set(words))
```

Normalize the words,
build the vocabulary

Strings: Text Processing

- Strings are specified using single quotes or double quotes
- If a string contains a single quote, we must backslash-escape the quote.
- For example, circus = 'Monty Python\'s Flying Circus'
- In case string is declared in multiple lines we need to use backslash or parentheses so that the interpreter knows that the statement is not complete after the first line. For example,
- str = "Shall I compare thee to a Summer's day?"\
- ..."Thou are more lovely and more temperate:"
- abc = ("Rough winds do shake the darling buds of May,"
- … "And Summer's lease hath all too short a date:")
- In order to introduce new line in string we use a triple-quoted string
- ""Shall I compare thee to a Summer's day?
- ... Thou are more lovely and more temperate:"""

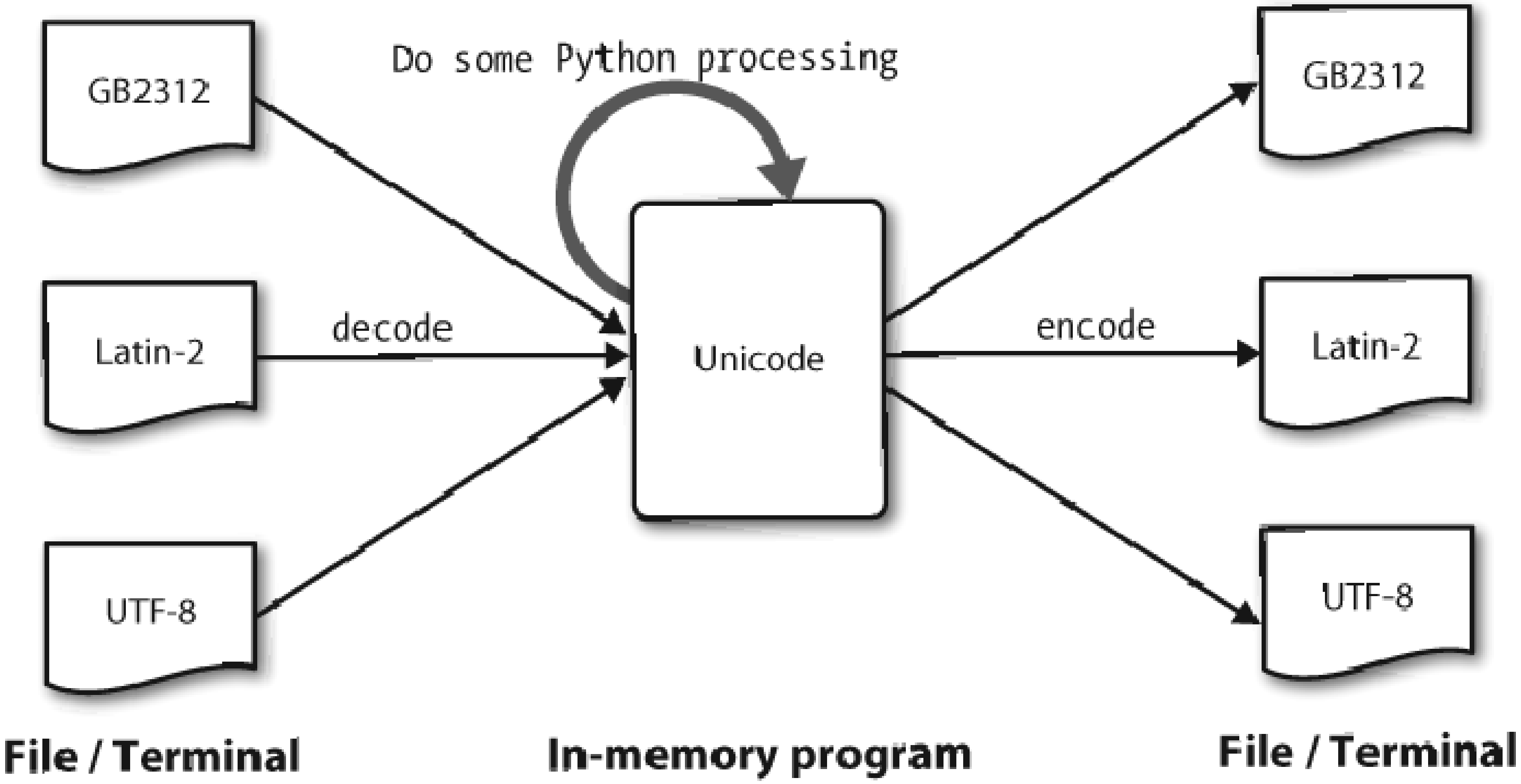
Operations on Strings

- s.find(t) -Index of first instance of string t inside s (-1 if not found)
- s.rfind(t) -Index of last instance of string t inside s (-1 if not found)
- s.index(t) -Like s.find(t) , except it raises ValueError if not found
- s.rindex(t) -Like s.rfind(t) , except it raises ValueError if not found
- s.join(text)- Combine the words of the text into a string using s as the glue
- s.split(t) -Split s into a list wherever a t is found (whitespace by default)
- s.splitlines() -Split s into a list of strings, one per line
- s.lower() -A lowercased version of the string s
- s.upper() -An uppercased version of the string s
- s.titlecase()- A titlecased version of the string s
- s.strip() -A copy of s without leading or trailing whitespace
- s.replace(t, u)- Replace instances of t with u inside s

Text Processing with Unicode

- Unicode supports over a million characters.
- Each character is assigned a number, called a code point. In Python, code points are written in the form \u XXXX, where XXXX is the number in four-digit hexadecimal form.
- Text in files will be in a particular encoding, so we need some mechanism for translating it into Unicode—translation into Unicode is called decoding.
- Conversely, to write out Unicode to a file or a terminal, we first need to translate it into a suitable encoding—this translation out of Unicode is called encoding

Text Processing with Unicode



Text Processing with Unicode

- The Python codecs module provides functions to read encoded data into Unicode strings, and to write out Unicode strings in encoded form.
- import codecs
- path = nltk.data.find('corpora/unicode_samples/polish-lat2.txt')
- f = codecs.open(path, encoding='latin2')
- raw=f.read()

Regular Expressions for Detecting Word Patterns

Many linguistic processing tasks involve pattern matching.

For example, we can find words ending with ed using endswith('ed') .

Regular expressions give us a more powerful and flexible method for describing the character patterns we are interested in.

To use regular expressions in Python, we need to import the re library using: import re .

We use re.search(p, s) function to check whether the pattern p can be found somewhere inside the string s .

```
wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()]
[w for w in wordlist if re.search('ed$', w)]
```

Regular Expressions for Detecting Word Patterns

. -Wildcard, matches any character
^abc - Matches some pattern abc at the start of a string
abc$ -Matches some pattern abc at the end of a string
[abc] -Matches one of a set of characters
[A-Z0-9] - Matches one of a range of characters
ed|ing|s -Matches one of the specified strings (disjunction)
* -Zero or more of previous item, e.g., a* , [a-z]* (also known as Kleene Closure)
+ -One or more of previous item, e.g., a+ , [a-z]+
? -Zero or one of the previous item (i.e., optional), e.g., a? , [a-z]?
{n} - Exactly n repeats where n is a non-negative integer
{n,} - At least n repeats
{,n} -No more than n repeats
{m,n} - At least m and no more than n repeats
a(b|c)+ Parentheses that indicate the scope of the operators

Regular Expressions for Detecting Word Patterns

Examples:

```
wsj = sorted(set(nltk.corpus.treebank.words()))
[w for w in wsj if re.search('^[0-9]+\.[0-9]+$', w)]
[w for w in wsj if re.search('^[A-Z]+\$$', w)]
[w for w in wsj if re.search('^[0-9]{4}$', w)]
[w for w in wsj if re.search('^[0-9]+-[a-z]{3,5}$', w)]
[w for w in wsj if re.search('^[a-z]{5,}-[a-z]{2,3}-[a-z]{,6}$', w)]
[w for w in wsj if re.search('(ed|ing)$', w)]
```