Beyond regex:
Natural Language Processing

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("<u>Barack Obama</u> lived in the <u>White House</u>.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The <u>*President*</u> promised <u>*he*</u> would attend.")
11. Sentiment analysis

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("Barack Obama lived in the White House.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The *President* promised *he* would attend.")
11. Sentiment analysis

Layers of text processing

1. **Encoding (e.g. ISO-Latin-1 to Unicode)**
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("Barack Obama lived in the White House.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The *President* promised *he* would attend.")
11. Sentiment analysis

What does it mean to convert a sequence of bytes to a string?

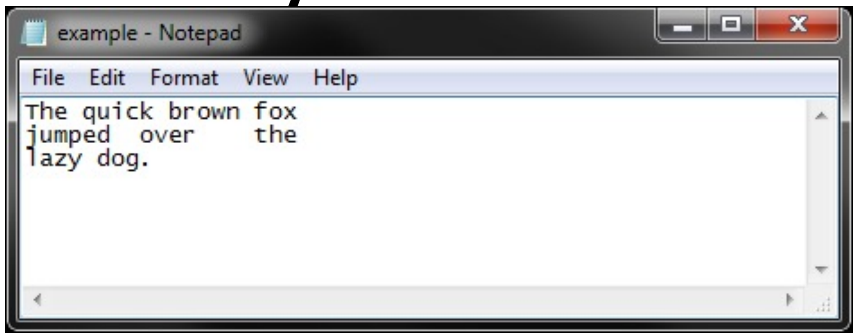
We must assume an <u>interpretation</u> for the sequence of bytes. What does that mean?

<u>`A string is a sequence of characters.`</u>

- What is a character?
- Smallest possible unit of text
- ‘A’, ‘B’, ‘C’, etc., are all different characters. So are È and Í and木.
- Characters are abstractions
  - The symbol for ohms (Ω) is usually drawn much like the capital letter omega (Ω) in the Greek alphabet
  - But these are two different characters that have different meanings.

Source: https://docs.python.org/3/howto/unicode.html

How are characters of plain text stored as bytes in a file?



The text is encoded as a stream of numbers. Each number represents a letter, symbol, or special character like tab or space.

Byte 00	84	104	101	32	113	117	105	99	107	32	98	114	111	119	110	32
	T	**h**	**e**		**q**	**u**	**i**	**c**	**k**		**b**	**r**	**o**	**w**	**n**	
Byte 16	102	111	120	10	106	117	109	112	101	100	9	111	118	101	114	9
	f	**o**	**x**	**\n**	**j**	**u**	**m**	**p**	**e**	**d**	**\t**	**o**	**v**	**e**	**r**	**\t**
Byte 32	116	104	101	10	108	97	122	121	32	100	111	103	46	10		
	t	**h**	**e**	**\n**	**l**	**a**	**z**	**y**		**d**	**o**	**g**	**.**	**\n**		

Delimiter: a sequence of one or more characters used to specify the boundary between separate, independent regions in plain text or other data streams

Special whitespace *delimiters*:

`\t` Tab = 9

`\n` End-of-line = 10

' ' Space = 32

Who decided this?

Telex: early text messaging and real-time chat
Wire services: receive-only teleprinters

Most teleprinters used 5-bit Baudot code (ITA2):
A = 00011
B = 11001
C = 01110
Carriage return = 01000
etc.
Still used in RTTY radioteletype

To support growing telecommunications market: The ASCII standard character set was created (1963)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x

0000000	84	104	101	32	113	117	105	99	107	32	98	114	111	119	110	32
	T	**h**	**e**		**q**	**u**	**i**	**c**	**k**		**b**	**r**	**o**	**w**	**n**	
0000016	102	111	120	10	106	117	109	112	101	100	9	111	118	101	114	9
	f	**o**	**x**	**\n**	**j**	**u**	**m**	**p**	**e**	**d**	**\t**	**o**	**v**	**e**	**r**	**\t**
0000032	116	104	101	10	108	97	122	121	32	100	111	103	46	10		
	t	**h**	**e**	**\n**	**l**	**a**	**z**	**y**		**d**	**o**	**g**	**.**	**\n**		

* ASCII = *American* Standard Code for Information Interchange

A <u>character set</u> specifies how numbers should be interpreted as character symbols

- ASCII
  - 7 bits per character (0-127) = 1 byte
  - Since 1960, from telegraph codes
- ISO-Latin-1 (ISO-8859-1)
  - "Code page"
  - 8-bit (0-255) = 1 byte
  - Superset of ASCII
  - Basis for original Web standard for HTTP and HTML
  - High-bit characters add e.g. accented characters for most 'Western' languages
  - Microsoft Windows ANSI similar variant

Char	Code	Name	Description
à	224	agrave	a grave
á	225	aacute	a acute
â	226	acirc	a circumflex
ã	227	atilde	a tilde
ä	228	auml	a umlaut
å	229	aring	a ring
æ	230	aelig	ae ligature
ç	231	ccedil	c cedilla
è	232	egrave	e grave
é	233	eacute	e acute
ê	234	ecirc	e circumflex
ë	235	euml	e umlaut
ì	236	igrave	i grave
í	237	iacute	i acute
î	238	icirc	i circumflex
ï	239	iuml	i umlaut

Char	Code	Name	Description
ð	240	eth	eth
ñ	241	ntilde	n tilde
ò	242	ograve	o grave
ó	243	oacute	o acute
ô	244	ocirc	o circumflex
õ	245	otilde	o tilde
ö	246	ouml	o umlaut
÷	247	divide	division sign
ø	248	oslash	o slash
ù	249	ugrave	u grave
ú	250	uacute	u acute
û	251	ucirc	u circumflex
ü	252	uuml	u umlaut
ý	253	yacute	y acute
þ	254	thorn	thorn
ÿ	255	yuml	y umlaut

The last 32 characters of the ISO-Latin-1 encoding

As international markets grew, so did the number of different character sets

Source: https://en.wikipedia.org/wiki/Character_encoding

There is no such thing as plain text.*

- We live in a multilingual world
- It doesn't make sense to have a string without knowing what encoding it uses.
- To deal with textual data, you first have to know how to decode the text!
- Every working programmer must know the basics of character sets, encodings.
- Enter… Unicode.

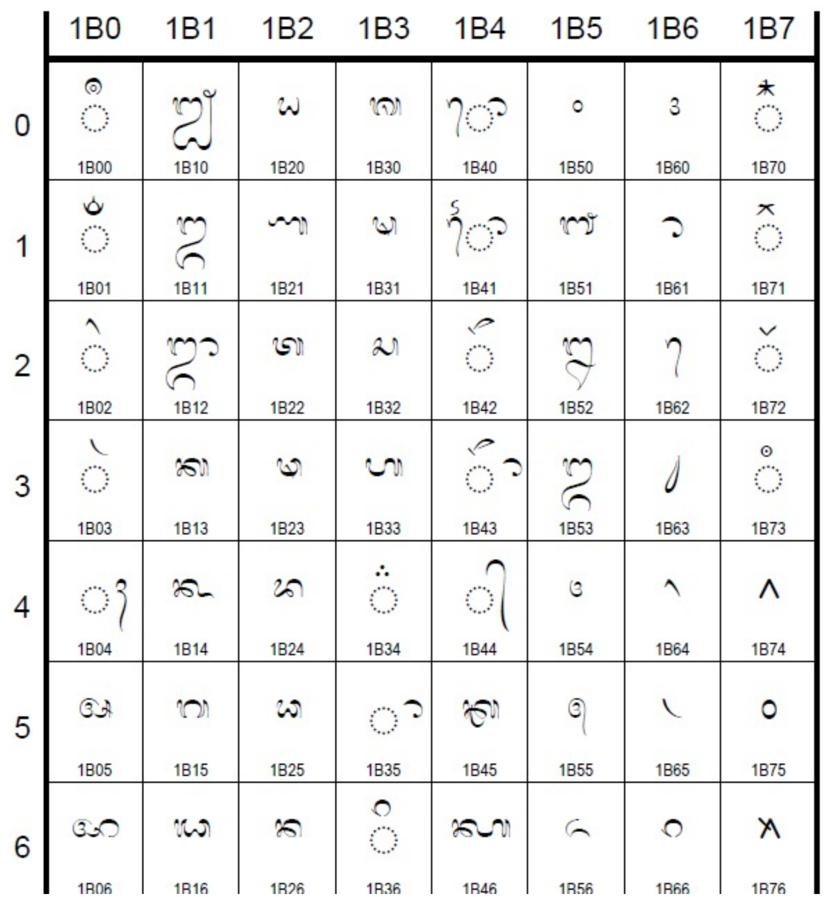
[*Source: http://www.joelonsoftware.com/articles/Unicode.html](http://www.joelonsoftware.com/articles/Unicode.html)

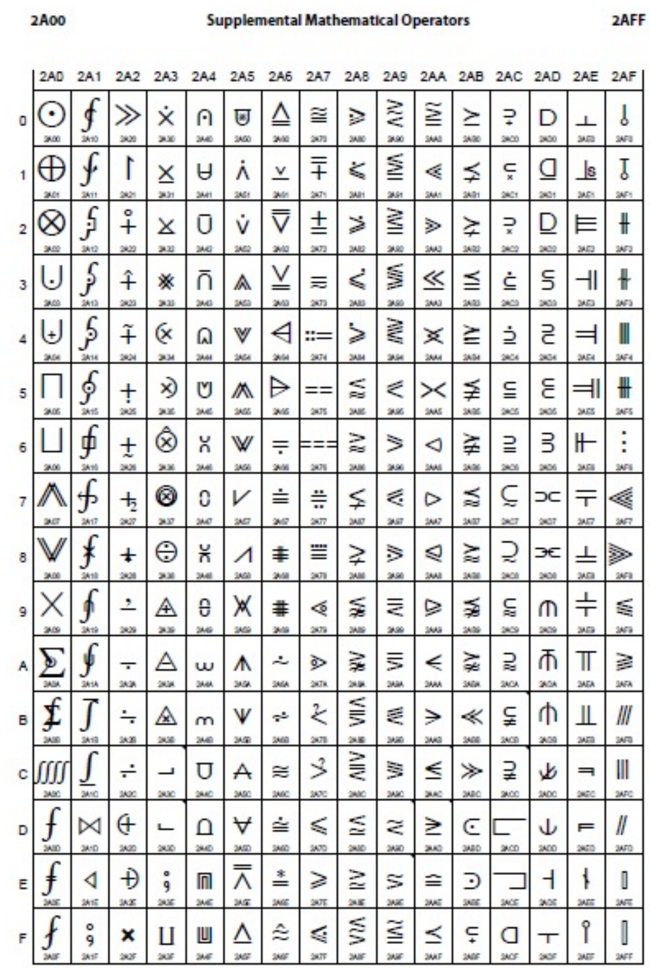
Unicode provides one universal character set

- Standard that covers more than 110,000 characters and more than 100 human languages.
  - Contains ISO-Latin-1: first 256 characters the same
  - First draft standard 1991: owned by Unicode Consortium
  - Beginning to replace ASCII and ISO character sets
- In theory, 0x0 to 0x10FFFF (1,114,112 characters; 21 bits)
  - Almost every language, past/present symbol you can think of
    - Emojis, Star Trek languages, …
- Implemented in most modern operating systems programming languages, and software
  - Including XML, Java, .NET framework, etc.

Source: http://www.joelonsoftware.com/articles/Unicode.html

In case you needed convincing that Unicode is serious about being a universal encoding…





[Source: http://www.unicode.org/charts/PDF/U1B00.pdf http://www.unicode.org/charts/PDF/U2A00.pdf]

Unicode support in a correctly-implemented Web browser

Azerbaijan (Latin script)	Heydar Aliyev (president)	Azərbaycan	Heydər Əliyev
Azerbaijan (Cyrillic script)	Heydar Aliyev (president)	Азәрбајҹан	Һејдәр Әлијев
Belgium (Flemish)	Rene Magritte (painter)	België	René Magritte
Belgium (French)	Rene Magritte (painter)	Belgique	René Magritte
Belgium (German)	Rene Magritte (painter)	Belgien	René Magritte
Bengal	Sukumar Ray	বাংলা	সুকুমার রায়
Bhutan	Gonpo Dorji (film actor)	འབྲུག་ཡུལ།	མགོན་པོ་རྡོ་རྗེ།
Cambodia (Khmer)	Venerable PreahBuddhaghosachar Chuon Nath	ប្រទេសកម្ពុជា	ព្រះពុទ្ធឃោសាចារ្យជួនណាត
Canada	Celine Dion (singer)	Canada	Céline Dion
Canada - Nunavut (Inuktitut language)	Susan Aglukark (singer)	ᓄᓇᕗᒻᒥᐅᑦ	ᓱᓴᓐ ᐊᒡᓗᒃᑲᖅ
Southeast USA (Cherokee Nation)	Sequoyah (invented syllabary)	ᏣᎳᎩ (Tsalagi)	ᏍᏏᏉᏯ
People's Rep. of China	ZHANG Ziyi (actress)	中国	章子怡
People's Rep. of China	WONG Faye (singer)	中国	王菲
Czechia (Czech Republic)	Antonin Dvorak (composer)	Česko (Česká republika)	Antonín Dvořák
Denmark	Soren Hauch-Fausboll	Danmark	Søren Hauch-Fausbøll
Denmark	Soren Kierkegaard (theologian 1813-1855)	Danmark	Søren Kierkegård
Egypt (Masr)	Abdel Halim Hafez (singer)	مصر	عبدالحليم حافظ
Egypt (Masr)	Om Kolthoum (singer)	مصر	أم كلثوم
Eritrea	Berhane Zeray	ኤርትራ	ብርሃነ ዘርኣይ
Ethiopia	Haile Gebreselassie (Fastest man)	ኢትዮጵያ	ኃይሌ ገብረሥላሴ

Unicode characters: 'code points'

- Every letter in every alphabet is assigned a magic number by the Unicode consortium, like this: **U+0639**. This magic number is called a *code point*.
- The U+ means "Unicode" and the numbers are hexadecimal (base 16)
- They're all listed on the Unicode web site. (charmap utility in Windows)

H	**E**	**L**	**L**	**O**
U+0048	U+0065	U+006C	U+006C	U+006F

- Unicode can be encoded in a file or string in many different ways, depending on efficiency considerations.

Encodings:	H	E	L	L	O	
`UTF-16:`	`00 48`	`00 65`	`00 6C`	`00 6C`	`00 6F`	(two bytes/char)

The difference between character sets, encodings, and fonts

- The character set maps characters to numbers (code points)
- The encoding stores the number in a particular format (in file/memory/byte stream)
- A font has instructions for displaying the visual form of a character ('glyph') given a code point.

H	**E**	**L**	**L**	**O**
U+0048	U+0065	U+006C	U+006C	U+006F

```
UTF-8:    48 65 6C 6C 6F
```
(1 byte for common characters)

```
UTF-16:   00 48 00 65 00 6C 00 6C 00 6F
```
(~2 bytes/char)

H	*E*	*L*	*L*	*O*
U+0048	U+0065	U+006C	U+006C	U+006F

"Windows Edwardian Script MT"

UTF-8 is the default Unicode encoding

- UTF-8 Encoding:
  - In UTF-8, every code point from U+0 to U+127 is stored *in a single byte*.
  - Only code points 128 and above are stored using 2, 3, … , up to 6 bytes.
- Key result: English text looks *exactly the same in UTF-8 as it did in ASCII*

Encodings:

```
UTF-16:   00 48 00 65 00 6C 00 6C 00 6F
UTF-8:    48 65 6C 6C 6F
          H  E  L  L  O
```

(two bytes/char)

How do we know what encoding a text stream uses? The byte order mark (BOM) in the first few bytes

```
                中    国     是    美    丽
0000000 feff 4e2d 56fd 662f 7f8e 4e3d
0000016 7684 56fd 5bb6 3002
          的    国    家    。
```

UTF-16 0xFE 0xFF

UTF-8 0xEF 0xBB 0xBF

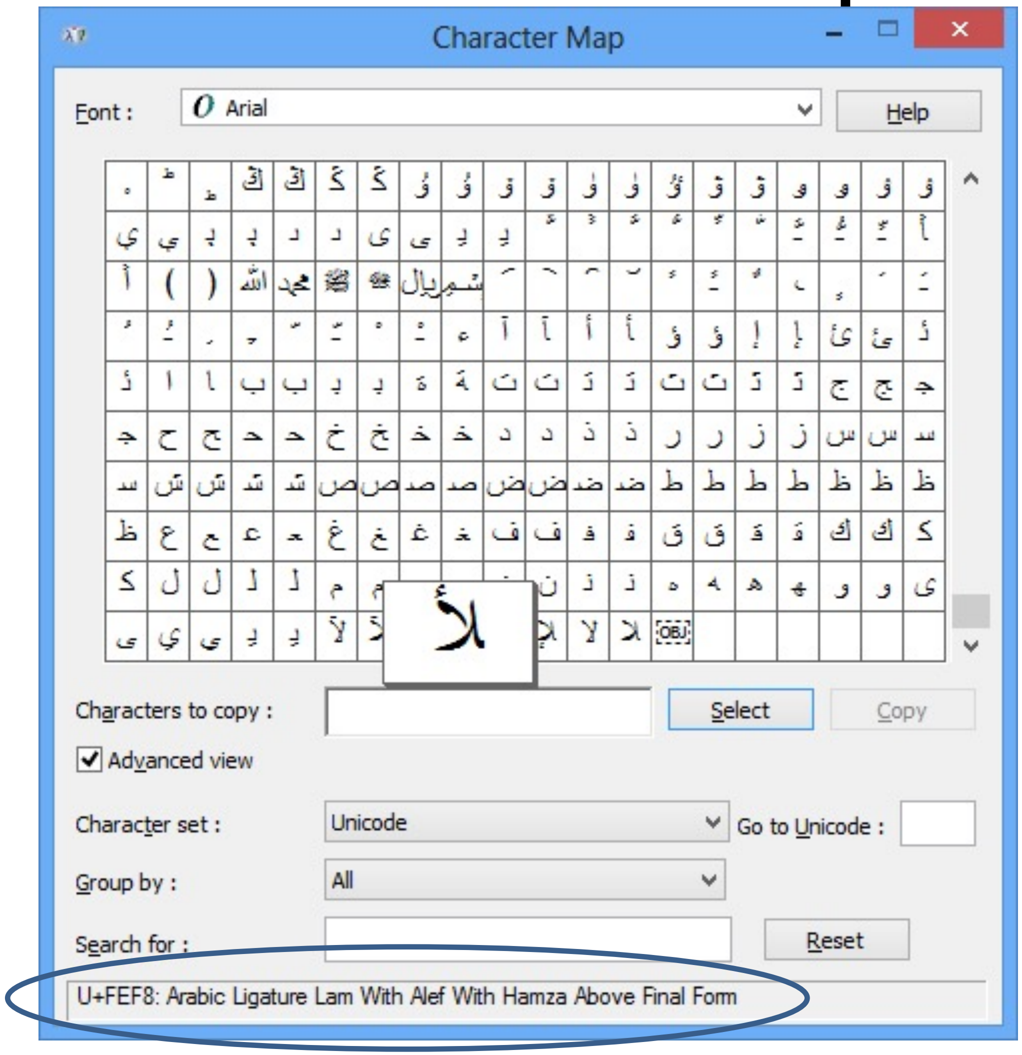
The byte order mark (BOM) is a Unicode character
that serves as a "magic number" at the start of a text stream.

Encodes:

- That this is (very likely) a Unicode text stream
- Which type of Unicode encoding is used (8-bit, 16-bit, 32-bit)
- What byte order (or "endianness") the text stream uses

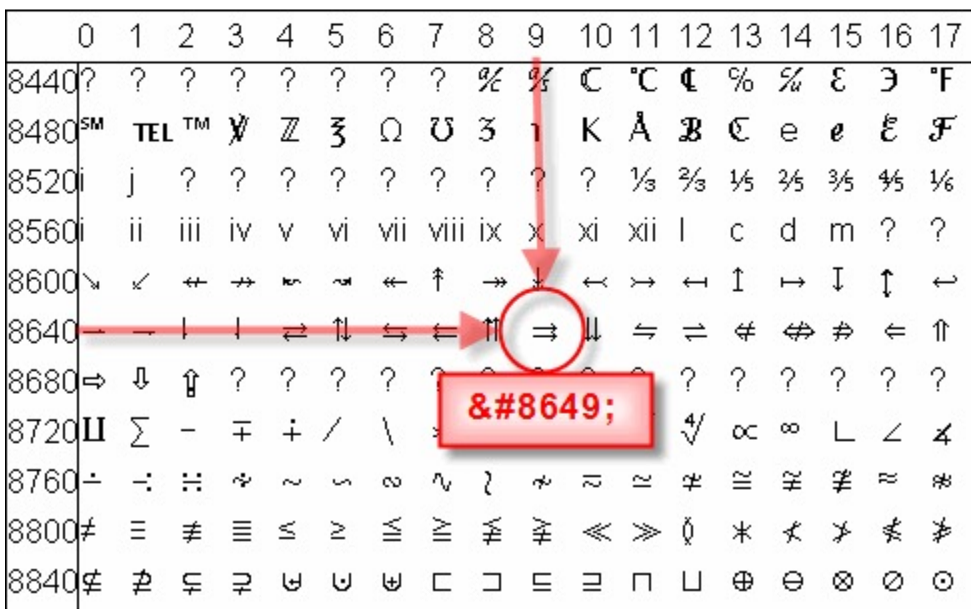
The BOM is a zero-width non-breaking space if it occurs
in the middle of the text stream.

The Windows charmap utility



Unicode entities in HTML

- HTML files originally were encoded in ISO-Latin-1
- HTML standard extended to Unicode in 1997



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
8440	?	?	?	?	?	?	?	?	℀	℁	ℂ	℃	℄	℅	℆	ℇ	℈	℉
8480	℠	℡	™	℣	ℤ	℥	Ω	℧	ℨ	℩	K	Å	ℬ	ℭ	℮	ℯ	ℰ	ℱ
8520	i	j	?	?	?	?	?	?	?	?	?	⅓	⅔	⅕	⅖	⅗	⅘	⅙
8560	i	ii	iii	iv	v	vi	vii	viii	ix	x	xi	xii	l	c	d	m	?	?
8600	↘	↙	↚	↛	↜	↝	↞	↟	↠	↡	↢	↣	↤	↥	↦	↧	↨	↩
8640	⇀	⇁	⇂	⇃	⇄	⇅	⇆	⇇	⇈	⇉	⇊	⇋	⇌	⇍	⇎	⇏	⇐	⇑
8680	⇨	⇩	⇪	?	?	?	?	?					?	?	?	?	?	?
8720	∐	∑	−	∓	∔	∕	∖						∜	∝	∞	∟	∠	∡
8760	∸	∹	∺	∻	∼	∽	∾	∿	≀	≁	≂	≃	≄	≅	≆	≇	≈	≉
8800	≠	≡	≢	≣	≤	≥	≦	≧	≨	≩	≪	≫	≬	≭	≮	≯	≰	≱
8840	⊈	⊉	⊊	⊋	⊌	⊍	⊎	⊏	⊐	⊑	⊒	⊓	⊔	⊕	⊖	⊗	⊘	⊙

Someone gives you a document. How do you know the encoding?

<u>Email</u>:

Content-Type: text/plain; charset="UTF-8"

<u>HTML</u>:

- ```
  <html><head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  ```
- Often missing, so clever browsers will try to figure out the language and encoding from the frequency distribution of byte patterns

<u>Python scripts</u>: First line special comment, e.g.

```
# -*- coding: utf-8 -*-
```

or

```
# -*- coding: latin-1 -*-
```
```

Python 3.x support for Unicode

- All Python 3 strings (str type) contain Unicode characters
- Default encoding is UTF-8
- In Python 2.5+, the default encoding for scripts was ASCII. (In Python 3, default is Unicode.)
- Specific code points are written using the **\u escape sequence**, which is followed by hex digits giving the code point.

```
>>> s = '\u4e2d\u56fd'
>>> print(s)
中国
```

```
>>> "\N{GREEK CAPITAL LETTER DELTA}"  # Using the character name
'\u0394'
>>> "\u0394"                          # Using a 16-bit hex value
'\u0394'
>>> "\U00000394"                      # Using a 32-bit hex value
'\u0394'
```

Source: http://docs.python.org/3/howto/unicode.html

When and how should you worry about encoding?

- Whenever you are reading or writing potentially unknown external files or other byte streams (e.g. HTTP response)
- To open a text file you know is encoded using UTF-8:

```
f = open("hello.txt", encoding = "utf-8")
x = f.read()
```

- What if you don't know the encoding?
- Use **`encoding = "latin-1"`** if you're not sure what's in the file and want to avoid dreaded encoding errors, so the system will make its "best effort" to map all input bytes to the first 256 Unicode code points (equivalent to old ISO-Latin-1 standard)

Reference: http://docs.python.org/3/howto/unicode.html

Encoding and decoding in Python

```
html_string = html_bytes.decode("utf-8")
```

Decode

String ← Bytes

Encode

Bytes ← String

```
html_bytes = html_string.encode("utf-8")
```

```
>>> import urllib.request
>>> response = urllib.request.urlopen("http://www.umich.edu/~kevynct")
>>> html_page = response.read().decode("utf-8")
>>> type(html_page)
<class 'str'>
```

Unicode summary

- ANSI and Unicode formats for text files
- Python 3.x support for Unicode strings, including encoding and decoding

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. **Normalization (e.g. case conversion)**
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("<u>Barack Obama</u> lived in the <u>White House</u>.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The <u>*President*</u> promised <u>*he*</u> would attend.")
11. Sentiment analysis

Normalization

- you know this one: `str.lower()`
- don't forget to do this!

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. **Tokenization / word-breaking ("what is a word?")**
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("<u>Barack Obama</u> lived in the <u>White House</u>.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The <u>*President*</u> promised <u>*he*</u> would attend.")
11. Sentiment analysis

Tokens and Types

The term *word* can be used in two different ways:

1. To refer to an individual occurrence of a word
2. To refer to an abstract vocabulary item

For example, the sentence *"my dog likes his dog"* contains <u>five</u> occurrences of words, but <u>four</u> vocabulary items.

To avoid confusion use more precise terminology:

1. ***Word token:*** a specific occurrence of a word
2. ***Word type:*** a vocabulary item

Adapted from B. Rosario's UC Berkeley I256 slides

Tokenization

- The simplest way to represent a **text** is with a single string.
- Difficult to process text in this format.
- Often, it is more convenient to work with a list of tokens.
- The task of converting a text from a single string to a list of tokens is known as *tokenization*.

- Two types of tokenization: sentence and word
- sentence tokenization takes a blob of text and splits it into sentences
- word tokenization takes a blob of text (usually a sentence) and splits it into words

Adapted from B. Rosario's UC Berkeley I256 slides

Tokenization

Sentence tokenization:

```
import nltk.data
text = "Hello, world.  How are you, world?"
sent_text = nltk.sent_tokenize(text)
sent_text
Output : ['Hello, world.', 'How are you, world?']
```

Word tokenization:

```
sentence = "The quick brown fox jumped over the lazy dog!"
nltk.word_tokenize(sentence)
  Output: ['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog',
                                        '!']
```

For real-world, messy text:

```
from nltk.tokenize import TweetTokenizer
tokenizer_words = TweetTokenizer()
tokenizer_words.tokenize(sentence)
```

Tokenization

- counting total and unique words is easy and tells a lot about the text
- a useful measure to calculate is the type-token ratio (TTR)
  - what do high and low values of TTR tell you?

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. ~~Spelling correction (if from interactive input)~~
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("<u>Barack Obama</u> lived in the <u>White House</u>.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The <u>*President*</u> promised <u>*he*</u> would attend.")
11. Sentiment analysis

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. **Stopwords ("what is a useful word?")**
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("<u>Barack Obama</u> lived in the <u>White House</u>.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The <u>*President*</u> promised <u>*he*</u> would attend.")
11. Sentiment analysis

Lexical Resources in NLTK: stopwords

- NLTK includes some corpora that are nothing more than **wordlists** (eg the Words Corpus)
- There is also a corpus of **stopwords**, that is, high-frequency words like *the*, *to* and *also* that we sometimes want to filter out of a document before further processing.
  - Stopwords usually have little lexical content, and their presence in a text fails to distinguish it from other texts.

```
>>> from nltk.corpus import stopwords
>>> stopwords.words('english')
['a', "a's", 'able', 'about', 'above', 'according', 'accordingly', 'across',
'actually', 'after', 'afterwards', 'again', 'against', "ain't", 'all', 'allow',
'allows', 'almost', 'alone', 'along', 'already', 'also', 'although', 'always', ...]
```

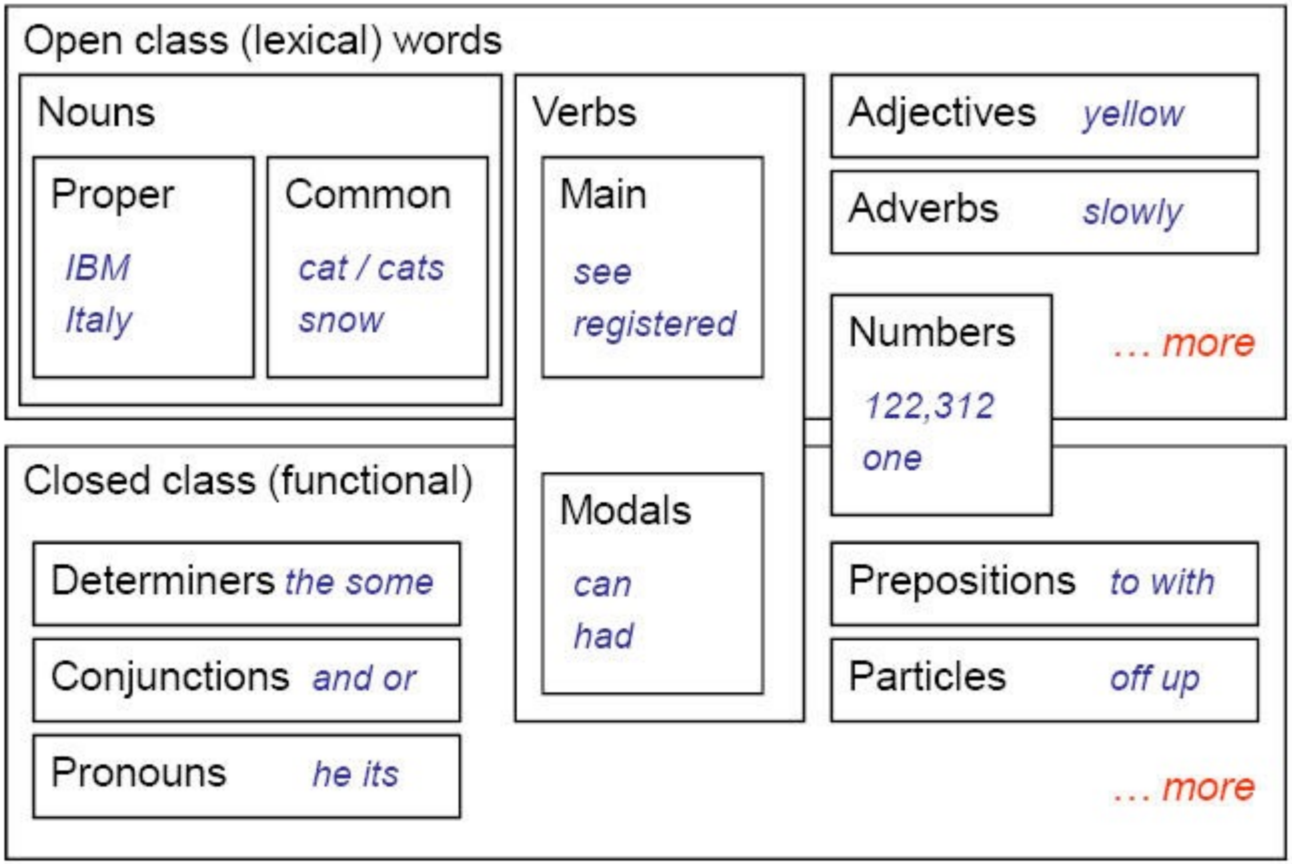
Adapted from B. Rosario's UC Berkeley I256 slides

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. **Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)**
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("Barack Obama lived in the White House.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The ***President*** promised *he* would attend.")
11. Sentiment analysis

Part-of-speech (English)

- One basic kind of linguistic structure: syntactic word classes



From Dan Klein’s UC Berkeley cs 288 slides

Terminology

- **Tagging**
  – associating labels with each token in a text
- **Tags**
  – The labels
  – **Syntactic word classes**
- **Tag Set**
  – The collection of tags used

NLTK reference: http://www.nltk.org/book/ch05.html

Why do Part-of-Speech tagging?

- Useful as a pre-processing step for parsing?
  - Less tag ambiguity means fewer parses
  - However, some tag choices are better decided by parsers

					IN			
DT	NNP	NN	VBD	VBN	RP	NN	NNS	
The	Georgia	branch	had	taken	on	loan	commitments	…

				VDN			
DT	NN	IN	NN	VBD	NNS	VBD	
The	average	of	interbank	offered	rates	plummeted	…

From Dan Klein's UC Berkeley cs 288 slides

Example

- **Typically a tagged text is a sequence of white-space separated base/tag tokens:**

These/DT
findings/NNS
should/MD
be/VB
useful/JJ
for/IN
therapeutic/JJ
strategies/NNS
and/CC
the/DT
development/NN
of/IN
immunosuppressants/NNS
targeting/VBG
the/DT
CD28/NN
costimulatory/NN
pathway/NN
./.

Part-of-speech (English)

CC	conjunction, coordinating	and both but either or
CD	numeral, cardinal	mid-1890 nine-thirty 0.5 one
DT	determiner	a all an every no that the
EX	existential there	there
FW	foreign word	gemeinschaft hund ich jeux
IN	preposition or conjunction, subordinating	among whether out on by if
JJ	adjective or numeral, ordinal	third ill-mannered regrettable
JJR	adjective, comparative	braver cheaper taller
JJS	adjective, superlative	bravest cheapest tallest
MD	modal auxiliary	can may might will would
NN	noun, common, singular or mass	cabbage thermostat investment subhumanity
NNP	noun, proper, singular	Motown Cougar Yvette Liverpool
NNPS	noun, proper, plural	Americans Materials States
NNS	noun, common, plural	undergraduates bric-a-brac averages
POS	genitive marker	' 's
PRP	pronoun, personal	hers himself it we them
PRP$	pronoun, possessive	her his mine my our ours their thy your
RB	adverb	occasionally maddeningly adventurously
RBR	adverb, comparative	further gloomier heavier less-perfectly
RBS	adverb, superlative	best biggest nearest worst
RP	particle	aboard away back by on open through
TO	"to" as preposition or infinitive marker	to
UH	interjection	huh howdy uh whammo shucks heck
VB	verb, base form	ask bring fire see take
VBD	verb, past tense	pleaded swiped registered saw
VBG	verb, present participle or gerund	stirring focusing approaching erasing
VBN	verb, past participle	dilapidated imitated reunifed unsettled
VBP	verb, present tense, not 3rd person singular	twist appear comprise mold postpone
VBZ	verb, present tense, 3rd person singular	bases reconstructs marks uses
WDT	WH-determiner	that what whatever which whichever
WP	WH-pronoun	that what whatever which who whom
WP$	WH-pronoun, possessive	whose
WRB	Wh-adverb	however whenever where why

Part-of-speech tagging

```
>>> sentence = "The quick brown fox jumped over the
lazy dog!"
>>> tokens = nltk.word_tokenize(sentence)
>>> tagged = nltk.pos_tag(tokens)
>>> tagged
[('The', 'DT'), ('quick', 'NN'), ('brown', 'NN'),
('fox', 'NN'), ('jumped', 'VBD'), ('over', 'IN'),
('the', 'DT'), ('lazy', 'NN'), ('dog', 'NN'), ('!',
'.')]
```

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. **Stemming ("what is an underlying word root?")**
8. Named-entity recognition ("Barack Obama lived in the White House.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The *President* promised *he* would attend.")
11. Sentiment analysis

Stemming: merging different inflections of words

thinks → think

thinking → think

thinker → think

argue → argu

argument → argu

arguing → argu

argus → argu

Porter Stemmer: fast but inaccurate

```
>>> stemmer = PorterStemmer()
>>> plurals = ['caresses', 'flies', 'dies', 'mules', 'denied',
...            'died', 'agreed', 'owned', 'humbled', 'sized',
...            'meeting', 'stating', 'siezing', 'itemization',
...            'sensational', 'traditional', 'reference', 'colonizer',
...            'plotted']
>>> singles = [stemmer.stem(plural) for plural in plurals]
>>> print(' '.join(singles))
caress fli die mule deni die agre own humbl size meet
state siez item sensat tradit refer colon plot
```

WordNet Lemmatization: slower, more precise/conservative

```
>>> from nltk.stem.wordnet import WordNetLemmatizer
>>> lmtzr = WordNetLemmatizer()
>>> lmtzr.lemmatize('cars')
'car'
>>> lmtzr.lemmatize('feet')
'foot'
>>> lmtzr.lemmatize('people')
'people'
>>> lmtzr.lemmatize('fantasized','v')
'fantasize'

plurals = ['caresses', 'flies', 'dies', 'mules', 'denied',
...              'died', 'agreed', 'owned', 'humbled', 'sized',
...              'meeting', 'stating', 'siezing', 'itemization',
...              'sensational', 'traditional', 'reference',

>>> singles = [lmtzr.lemmatize(plural) for plural in plurals]
>>> print (' '.join(singles))
```

caresss fly dy mule denied died agreed owned humbled sized meeting stating siezing itemization sensational traditional reference colonizer plotted

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. **Named-entity recognition** ("Barack Obama lived in the White House.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The *President* promised *he* would attend.")
11. Sentiment analysis

Named entity recognition: detecting people, places, things …

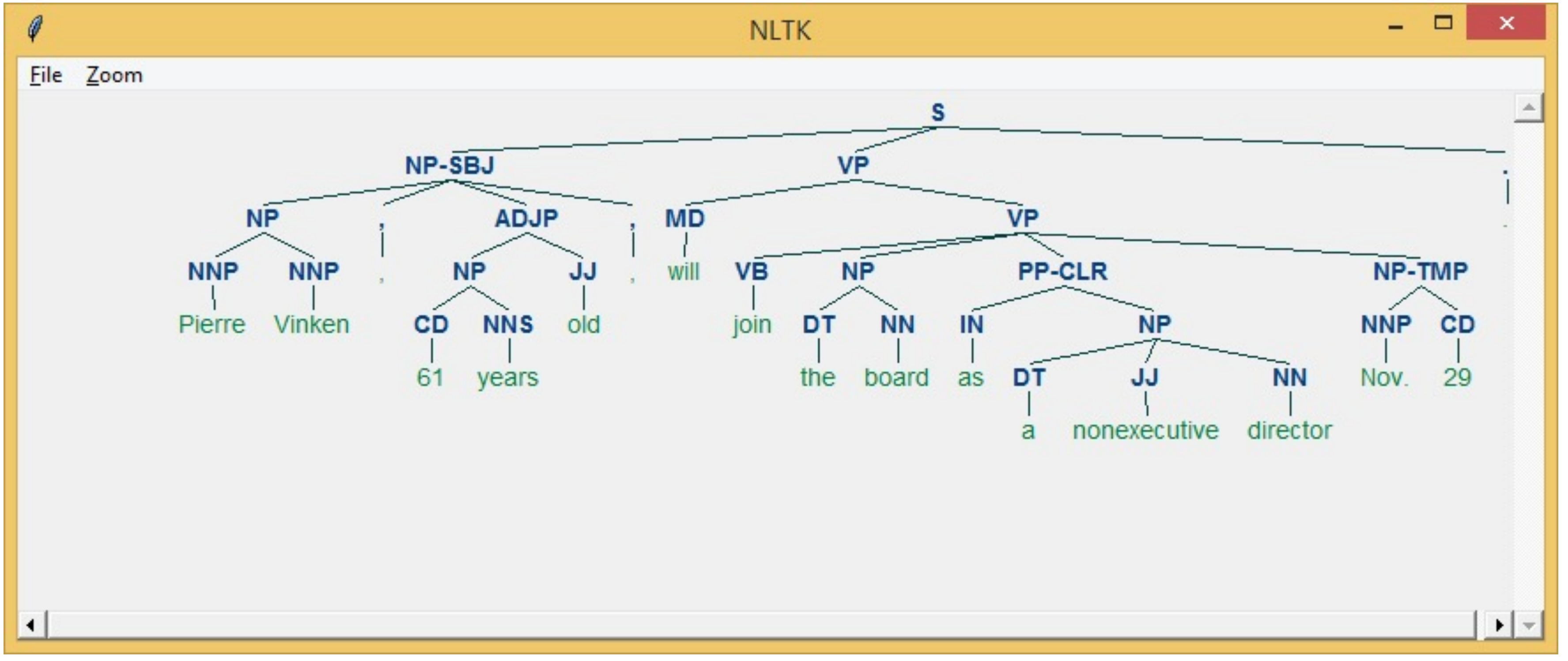
```
>>> entities = nltk.chunk.ne_chunk(tagged)
>>> entities
Tree('S', [('The', 'DT'), ('quick', 'JJ'), ('brown',
'NN'), ('fox', 'NNS'), ('spoke', 'VBD'), ('to', 'TO'),
Tree('PERSON', [('Abraham', 'NNP'), ('Lincoln', 'NNP')]),
('.', '.')])
```

Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("Barack Obama lived in the White House.")
9. **Parsing (sentence structure)**
10. Co-reference resolution ("The *President* promised *he* would attend.")
11. Sentiment analysis

Displaying a parse tree

```
>>> from nltk.corpus import treebank
>>> t = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> t.draw()
```



Layers of text processing

1. Encoding (e.g. ISO-Latin-1 to Unicode)
2. Normalization (e.g. case conversion)
3. Tokenization / word-breaking ("what is a word?")
4. Spelling correction (if from interactive input)
5. Stopwords ("what is a useful word?")
6. Part-of-speech tagging (nouns, verbs, adjectives, adverbs, etc.)
7. Stemming ("what is an underlying word root?")
8. Named-entity recognition ("<u>Barack Obama</u> lived in the <u>White House</u>.")
9. Parsing (sentence structure)
10. Co-reference resolution ("The <u>*President*</u> promised <u>*he*</u> would attend.")
11. Sentiment analysis

Sentiment Analysis

- attempt to identify affective (emotional) state of text based on NLP techniques
- can be extended to other axes (e.g. helpfulness)

Sentiment Analysis

- Which words are associated with positive sentiment? With negative sentiment? Are they different?
- How do we assess sentiment?
- Online demos:
  - http://text-processing.com/demo/sentiment/ (open source)
  - https://app.monkeylearn.com/ (proprietary)

Sentiment Analysis: Limitations

- highly domain-specific
- difficult to assess mixed-sentiment statements (e.g. "The introduction to your essay is good, but the conclusions are weak.")

Sentiment Analysis

- twitter feeds are commonly used to experiment with sentiment analysis

Natural Language Processing in Python with NLTK

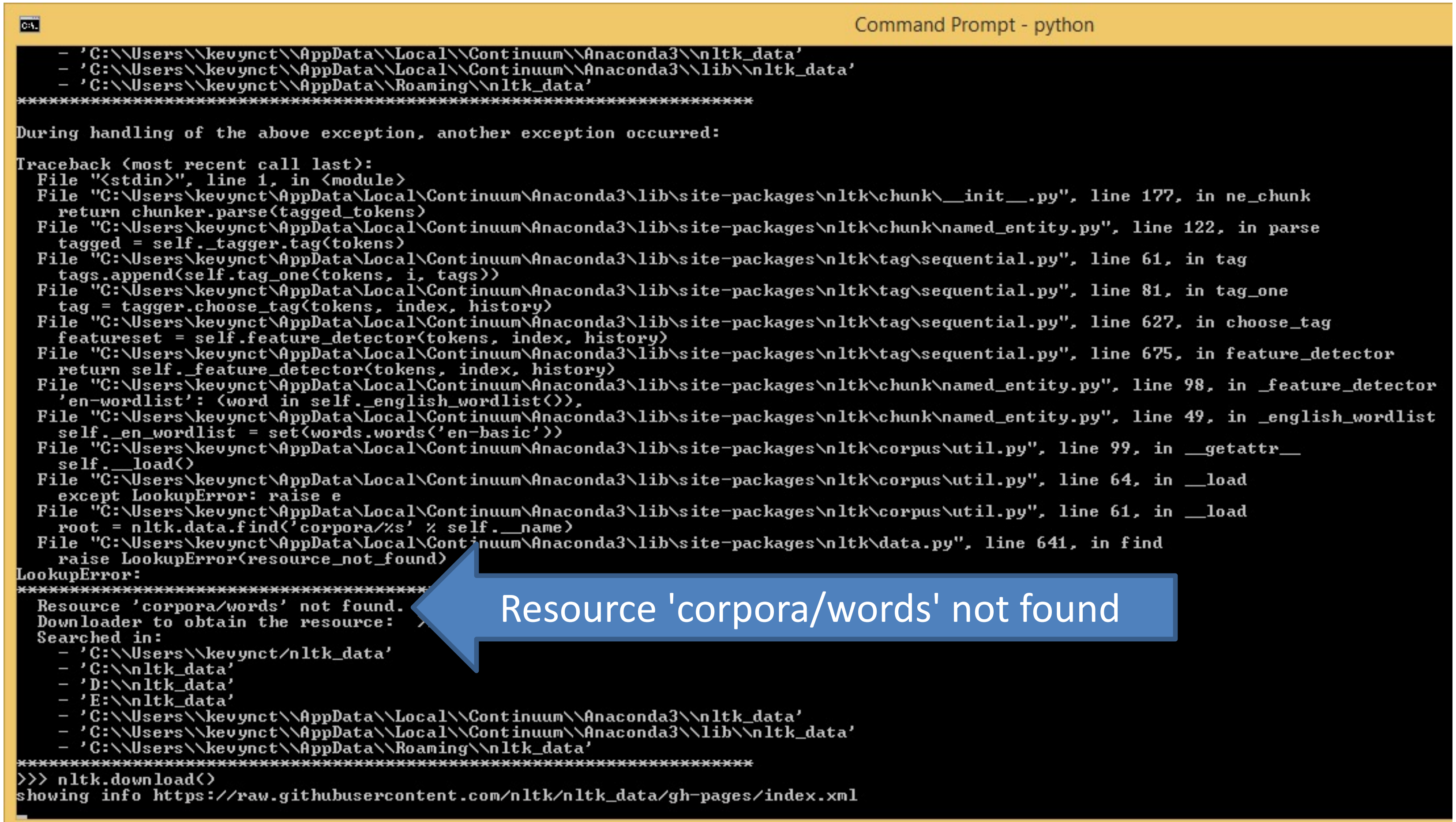
NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, and an active discussion forum.

Introduction to the Natural Language Toolkit (NLTK)

- Basic classes for representing data relevant to natural language processing.
- Standard interfaces for performing NLP tasks, such as tokenization, tagging, and parsing.
- Standard implementations of each task, which can be combined to solve complex problems.

```
import nltk
```

Installing NLTK components



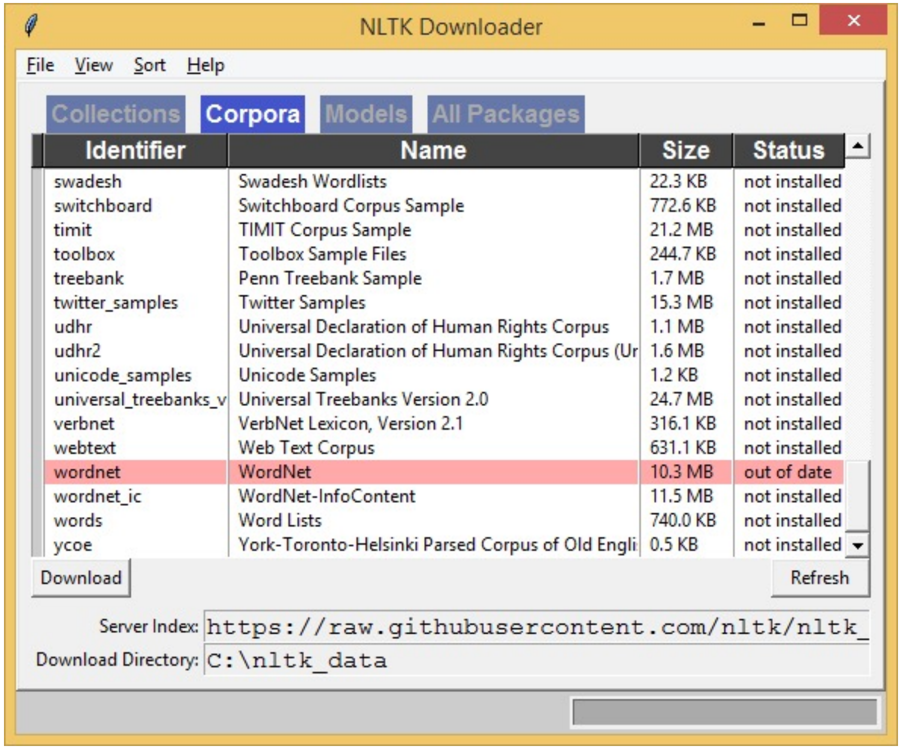

```
    - 'C:\\Users\\kevynct\\AppData\\Local\\Continuum\\Anaconda3\\nltk_data'
    - 'C:\\Users\\kevynct\\AppData\\Local\\Continuum\\Anaconda3\\lib\\nltk_data'
    - 'C:\\Users\\kevynct\\AppData\\Roaming\\nltk_data'
**********************************************************************

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\chunk\__init__.py", line 177, in ne_chunk
    return chunker.parse(tagged_tokens)
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\chunk\named_entity.py", line 122, in parse
    tagged = self._tagger.tag(tokens)
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\tag\sequential.py", line 61, in tag
    tags.append(self.tag_one(tokens, i, tags))
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\tag\sequential.py", line 81, in tag_one
    tag = tagger.choose_tag(tokens, index, history)
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\tag\sequential.py", line 627, in choose_tag
    featureset = self.feature_detector(tokens, index, history)
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\tag\sequential.py", line 675, in feature_detector
    return self._feature_detector(tokens, index, history)
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\chunk\named_entity.py", line 98, in _feature_detector
    'en-wordlist': (word in self._english_wordlist()),
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\chunk\named_entity.py", line 49, in _english_wordlist
    self._en_wordlist = set(words.words('en-basic'))
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\corpus\util.py", line 99, in __getattr__
    self.__load()
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\corpus\util.py", line 64, in __load
    except LookupError: raise e
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\corpus\util.py", line 61, in __load
    root = nltk.data.find('corpora/%s' % self.__name)
  File "C:\Users\kevynct\AppData\Local\Continuum\Anaconda3\lib\site-packages\nltk\data.py", line 641, in find
    raise LookupError(resource_not_found)
LookupError:
***********************************
  Resource 'corpora/words' not found.
  Downloader to obtain the resource:  >
  Searched in:
    - 'C:\\Users\\kevynct/nltk_data'
    - 'C:\\nltk_data'
    - 'D:\\nltk_data'
    - 'E:\\nltk_data'
    - 'C:\\Users\\kevynct\\AppData\\Local\\Continuum\\Anaconda3\\nltk_data'
    - 'C:\\Users\\kevynct\\AppData\\Local\\Continuum\\Anaconda3\\lib\\nltk_data'
    - 'C:\\Users\\kevynct\\AppData\\Roaming\\nltk_data'
**********************************************************************
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Solution: NLTK Downloader

>>> nltk.download()



What you should know

- The existence of text corpora in NLTK and how to access them
- High-level picture of low- to high-level text processing
- General idea of what these text processing steps are, and how to do them from NLTK:
  - Tokenization
  - Stemming
  - Part-of-speech tagging

Resources

http://www.nltk.org/book/