

Linguistics & Corpora

Monday, February 2, 2015

Plan for Today:

- Character Encodings
- Regular Expressions



<http://www.xkcd.com/1209/>

Last time

Syntax

Word Alignment: Alien Language Activity

Python 3

Today

Working with raw text

- Codecs & Encodings
- Regular Expressions

Issues with Text Data

Somebody gives you a file and says there's text in it.

Issues with obtaining the text?

- text encoding
- language recognition
- formatting (e.g. web, xml, ...)
- misc. information to be removed
 - header information
 - tables, figures
 - footnotes

Character Encoding

How are individual characters represented in actual bits on your computer?

(Short answer: It depends! And the differences end up being a major pain for us!)

Character Encoding

Goal: Represent letters (and other text characters) as 0s and 1s

Competing Factors:

- Compactness
- Size of character set

Hexadecimal

Base-16 representation of numbers

Symbols: 0-9, A-F

Converting binary to hex:

Binary	0000	0001	0010	0011	0100	0101	0110	0111
	0	1	2	3	4	5	6	7
Binary	1000	1001	1010	1011	1100	1101	1110	1111
	8	9	A	B	C	D	E	F

Text encoding: Telegraphs

Baudot encoding

5 bits per character

2 sets of characters; one “shift character” to switch to each.

How many total characters?

WITHOUT LOWERCASE
TODAY PEOPLE WILL THINK
YOU ARE YELLING AT THEM



Text for Teletyping

1960s: ASCII

- 7 bits / character
 - Now how many characters?
- Letters, numbers, punctuation, space, control characters
- Dominated web until ~2007



Accented vowels!

1980s: 8 bit characters

- ISO 8859-1: 191 characters from latin script
- Windows 1252
 - Superset of ISO 8859-1
 - Replaces a range of control characters with displayable characters.
 - Result:
 - “That’s my favorite hat”
Â“ThaÂt’s my favorite hat.Â”

Other languages!

1990s: Unicode

- Key idea: balance
 - Store a lot of characters
1,112,064 valid code points
 - Minimize size of each character
Avoid 21 bits/character, especially if most of our text is ASCII
- Coded character set: Function from int to character
- (Possibly multiple) character encoding forms
 - UTF-8: Blocks of (one or more) 8 bit units
 - UTF-16: Blocks of (one or more) 16 bit units

UTF-8

Currently makes up most of the web.

Dominates standards, etc.

What does it look like?

UTF-8

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

What about other languages?

GB2312: 1980

Official character set of People's Republic of China

GBK: 1990s expansion in response to unicode

GB18030: 2000s update/expansion

Why not just use Unicode/UTF-8?

Characters in Python 3

In Python 3...

- Text **str** objects are Unicode (code points)
 - No need for u"data" like in Python 2.x
- Text encoded as binary data has type **bytes**
 - Create with b"data"
 - Use **decode** to go from **bytes** to **str**
- Files opened as text files need to convert from bytes to code points using some encoding

Pattern Matching in Python

Regular Expressions (“Regex”)

We’ll look at:

- Matching characters
- Metacharacters
- Repetition
- Grouping
- Using regular expressions in Python

Matching Characters

Most characters match themselves.

hmcNLP matches the exact string “hmcNLP”

Case-sensitive (unless you use a special mode)

These are all valid regular expressions:

Regular expressions are very powerful

Look what can we do with regular expressions

These do not seem all that powerful yet

Metacharacters

To do more than match string literals, some characters have special meaning.

.

match any character

[]

match any character inside the brackets

^

make a character set the complement of its contents, AND match beginning of line

\

escapes metacharacters so they can be matched in strings, AND introduces special sequences

\$

match end of line

Special Sequences

There are lots, but most useful may be...

\d	Match any decimal digit
\D	Match any non-digit character
\w	Match any alphanumeric character
\W	Match any non-alphanumeric character
\s	Match any whitespace character
\S	Match any non-whitespace character

Repetition

Some characters modify the thing before them by saying how many times they can appear:

*

Zero or more times

+

One or more times

?

Zero or one time

{m,n}

Anywhere from m to n times

Grouping

Parentheses around characters form a group:

- `(ab)+` matches ab, abab, ababab, abababab, ...

The pipe character | allows alternatives in a group:

- `(ab|cd)+` matches ab, cd, abcd, cdab, ababcdcdab, ...

`\N` matches the contents of group N :

- `(\w)a\1` matches aaa, bab, cac, dad, eae, ...

Group modifiers:

- `(?:sometext)` is a non-capturing group
- `(?P<name>sometext)` is a named group

Using regular expressions in Python

Regex functionality in `re` module

Create regex object with `re.compile()`

Match regex object to string with

- `search()` True if RE matches from start of string
- `match()` True if RE matches anywhere in string
- `findall()` Returns a list of matches
- `finditer()` Returns an iterator of matches

RE groups in python

Match objects can give information about the substrings that matched each group in a regular expression:

- `group(N)` returns the character(s) matched by group N
- `group("name")` returns the group named "name"

A note about raw strings...

In Python, backslash is an escape character.

- `\n` is newline
- `\t` is a tab
- and quite a few others

In regular expressions, backslash is an escape character.

- `\[` matches the character `[`
- `\.` matches the character `.`
- and so forth for characters with special meaning

How do we match the character “\” in a regex?

Avoiding backslash fatigue

In raw strings, backslash is just a backslash.

- `\n` is two characters, not newline
- `\t` is two characters, not tab

If we represent regular expressions as raw strings, we only have to worry about regex escapes:

- `r"\\begin\\{itemize\\}"`

Let's practice!

Make up a regular expression

Exchange with someone next to you