텍스트 전처리

3. Processing Raw Text

conferences

- ACL (general NLP, top-tier)
  - https://acl2018.org/
- SIGDIAL (dialogue)
  - http://workshops.sigdial.org/conference19/
- INLG (generation)
  - https://inlg2018.uvt.nl/
- COLING (computational linguistics)
  - https://coling2018.org/workshop-list/
- ICIDS: storytelling (due on 9th, July)
  - https://icids2018.scss.tcd.ie/
- AIIDE: AI and entertainment (21st, June)
  - https://sites.google.com/ncsu.edu/aiide-2018/
- INT workshop (intelligent storytelling)
  - http://www.aaai.org/Library/Workshops/ws17-20.php
- EACL (유럽 ACL, 3년에 한번)
- EMNLP (Empirical Methods in Natural Language Processing)

Goals

The most important source of texts is undoubtedly the Web. It's convenient to have existing text collections to explore, such as the corpora we saw in the previous chapters. However, you probably have **your own text sources in mind**, and need to learn how to access them.

- 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?
- 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 chapters?
- How can we write programs to produce formatted output and save it in a file?

학습 내용

- Reading from Web, text files
- Small letter conversion, punctuation
- Sentence and word Tokenization
- Regular Expression
- Stemming and Lemmatization

Data source

- Gutenberg
  - 저작권이 만료된 소설, plain text
  - 죄와 벌 (http://www.gutenberg.org/files/2554/2554-0.txt)
- IMSDB
  - 영화 스크립트 저장 사이트, html
  - Lala Land: http://www.imsdb.com/scripts/La-La-Land.html
- Public datasets
  - Kaggle: https://www.kaggle.com/
  - UCI machine learning repository: https://archive.ics.uci.edu/ml/index.php
  - Reddit: https://www.reddit.com/r/datasets/
  - Github that contains public datasets: https://github.com/awesomedata/awesome-public-datasets
  - Google advanced search https://www.google.com/advanced_search

NLP and data science sites

- https://www.datasciencecentral.com/
- https://nlp.stanford.edu/blog/
- Dialogue set: https://nlp.stanford.edu/blog/a-new-multi-turn-multi-domain-task-oriented-dialogue-dataset/
- https://www.kdnuggets.com/
- https://nlpers.blogspot.com/
- https://lingpipe-blog.com/lingpipe-home-page/

Accessing Text from the Web

- A small sample of texts from Project Gutenberg appears in the NLTK corpus collection
- Other texts from Project Gutenberg at http://www.gutenberg.org/catalog/, and obtain a URL to an ASCII text file
- *raw* contains a string with 1,176,893 characters, including many details we are not interested in such as whitespace,
- **Tokenization** breaks up the string into words and punctuation

```
>>> from urllib import request
>>> url =
"http://www.gutenberg.org/files/2554/2554-0.txt"
>>> response = request.urlopen(url)
>>> raw = response.read().decode('utf8')
>>> type(raw)
<class 'str'>
>>> len(raw)
1176893
>>> raw[:75]
'The Project Gutenberg EBook of Crime and
Punishment, by Fyodor Dostoevsky\r\n'
>>> tokens = word_tokenize(raw)
>>> type(tokens)
<class 'list'>
>>> len(tokens)
254354
>>> tokens[:10]
['The', 'Project', 'Gutenberg', 'EBook', 'of', 'Crime', 'and',
'Punishment', ',', 'by']
```

- If we now take the further step of creating an NLTK text from this list, we can carry out all of the other linguistic processing
- Detect where the content begins and end

```
>>> text = nltk.Text(tokens)
>>> type(text)
<class 'nltk.text.Text'>
>>> text[1024:1062]
['CHAPTER', 'I', 'On', 'an', 'exceptionally', 'hot', 'evening',
'early', 'in', 'July', 'a', 'young', 'man', 'came', 'out', 'of', 'the',
'garret', 'in', 'which', 'he', 'lodged', 'in', 'S.', 'Place', 'and',
'walked', 'slowly',',', '.']
>>> text.collocations()
Katerina Ivanovna; Pyotr Petrovitch; Pulcheria
Alexandrovna; Avdotya Romanovna; Rodion
Romanovitch; Marfa Petrovna; Sofya Semyonovna; old
woman; Project Gutenberg-tm; Porfiry Petrovitch; Amalia
Ivanovna; great deal; Nikodim Fomitch; young man; Ilya
Petrovitch; n't know; Project Gutenberg; Dmitri Prokofitch;
Andrey Semyonovitch; Hay Market
>>> raw.find("PART I")
5338
>>> raw.rfind("End of Project Gutenberg's Crime")
1157743
>>> raw = raw[5338:1157743]
>>> raw.find("PART I")
0
```

The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsky

This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.org

Title: Crime and Punishment

Author: Fyodor Dostoevsky

Release Date: March 28, 2006 [EBook #2554]
Last Updated: October 27, 2016

Language: English

Character set encoding: UTF-8

*** START OF THIS PROJECT GUTENBERG EBOOK CRIME AND PUNISHM

Produced by John Bickers; and Dagny

CRIME AND PUNISHMENT

By Fyodor Dostoevsky

```
<html>
<head><title>La La Land Script at IMSDb.</title>
<meta name="description" content="La La Land script at the Internet Movie Script Database.">
<meta name="keywords" content="La La Land script, La La Land movie script, La La Land film script">
<meta name="viewport" content="width=device-width, initial-scale=1" />
<meta name="HandheldFriendly" content="true">
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
<meta http-equiv="Content-Language" content="EN">

<meta name=objecttype CONTENT=Document>
<meta name=ROBOTS CONTENT="INDEX, FOLLOW">
<meta name=Subject CONTENT="Movie scripts, Film scripts">
<meta name=rating CONTENT=General>
<meta name=distribution content=Global>
<meta name=revisit-after CONTENT="2 days">

<link href="/style.css" rel="stylesheet" type="text/css">

 <script async type="text/javascript" src="/postscribe/htmlParser.js"></script>
 <script async type="text/javascript" src="/postscribe/postscribe.js"></script>

<script type="text/javascript">
  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'UA-3785444-3']);
  _gaq.push(['_trackPageview']);

  (function() {
    var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.c
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
  })();
</script>

</head>

<body topmargin="0" bottommargin="0" onLoad="firewhenready();" id="mainbody">

<table width="99%" border="0" cellspacing="0" cellpadding="0" class="body">
  <tr>
    <td valign="bottom" bgcolor="#FF0000"><a href="http://www.imsdb.com" title="The Internet Movie Script D
src="/images/logo_top.gif" border="0"></a></td>
    <td bgcolor="#FF0000">
    <center>
        <font color="#FFFFFF"><h1>The Internet Movie Script Database (IMSDb)</h1></font>
</center>
  <tr>
    <td background="/images/reel.gif" height="13" colspan="2"><a href="http://www.imsdb.com" title="The Int
Database"><img src="/images/logo_middle.gif" border="0"></a></td>
  <tr>
    <td width="170" valign="top" class="smalltxt"> <a href="http://www.imsdb.com" title="The Internet Movie
src="/images/logo_bottom.gif" width="170" border="0"></a>
      <br>
    <center><span class="smalltxt">The web's largest <br>movie script resource!</span></center>
    </td>
    <td>
```

Crime and Punishment

La La Land

Dealing with HTML

- Read BBC News story called *Blondes to die out in 200 years*
- To get text out of HTML, use a Python library called BeautifulSoup, available from http://www.crummy.com/software/BeautifulSoup/
- find the start and end indexes of the content

Your turn: La La Land at http://www.imsdb.com/scripts/La-La-Land.html)

```
>>> from nltk import word_tokenize
>>> url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
>>> html = request.urlopen(url).read().decode('utf8')

>>> from bs4 import BeautifulSoup
>>> raw = BeautifulSoup(html, 'lxml').get_text()
>>> tokens = word_tokenize(raw)
>>> tokens
['BBC', 'NEWS', '|', 'Health', '|', 'Blondes', '"to", 'die', 'out', ...]

>>> tokens = tokens[110:390]
>>> text = nltk.Text(tokens)
>>> text.concordance('gene')
Displaying 5 of 5 matches:
hey say too few people now carry the gene for blondes to
last beyond the next blonde hair is caused by a recessive
gene . In order for a child to have blond have blonde hair , it
must have the gene on both sides of the family in the gere is
a disadvantage of having that gene or by chance . They do
n't disappear des would disappear is if having the gene was
a disadvantage and I do not thin
```

다양한 포맷

- RSS feeds: feedparser 사용
- Local files: 표준 라이브러리의 open, read 사용
  - >>> with open('document.txt', 'r') as f:
    … raw = f.read()

- 사용자 입력: input

Read from pdf

PyPDF2 module 사용

Back to the future script

http://www.imsdb.com/scripts/Back-to-the-Future.pdf

```
from PyPDF2 import PdfFileReader
import os
import nltk
def text_extractor(path):
    with open(path, 'rb') as f:
        pdf = PdfFileReader(f)
        # get the first page
        page = pdf.getPage(1)
        print(page)
        print('Page type: {}'.format(str(type(page))))
        text = page.extractText()
        raw = text
        print(len(text), type(raw))
        tokens = nltk.word_tokenize(raw)
        print(type(tokens), len(tokens))
        text = nltk.Text(tokens)
        print("text length is:",len(text))
        print(raw)
        print(raw.find('Method'))
        print(text)

if __name__ == '__main__':
    path = 'US9152209.pdf'
    #print(os.listdir('.'))
    text_extractor(path)
```

Word file

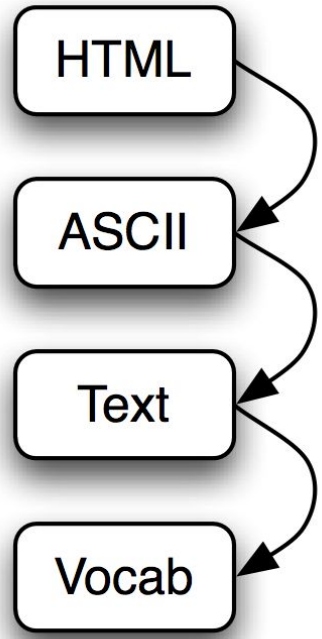
https://automatetheboringstuff.com/chapter13/

pip install python-docx

Python-docx 공식 문서

```
>>> import docx
>>> doc = docx.Document('Synset.docx')
>>> for i in doc.paragraphs:
>>>     print(i.text)

>>> doc = docx.Document()
>>> doc.add_heading('Header 0', 0)
>>> doc.add_heading('Header 1', 1)
>>> doc.add_heading('Header 2', 2)
>>> doc.add_heading('Header 3', 3)
>>> doc.add_heading('Header 4', 4)
>>> doc.save('headings.docx')
```

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

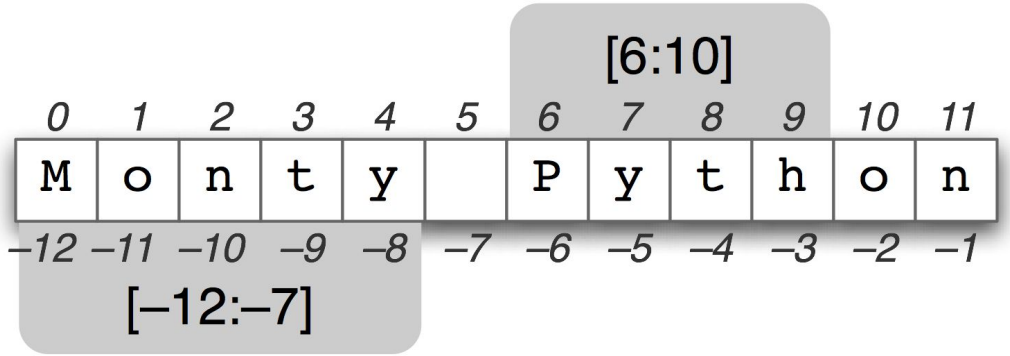
```
words = [w.lower() for w in text]
vocab = sorted(set(words))
```

Normalize the words, build the vocabulary

Strings

- Accessing Substring

```
>>> monty[-12:-7]
'Monty'
```



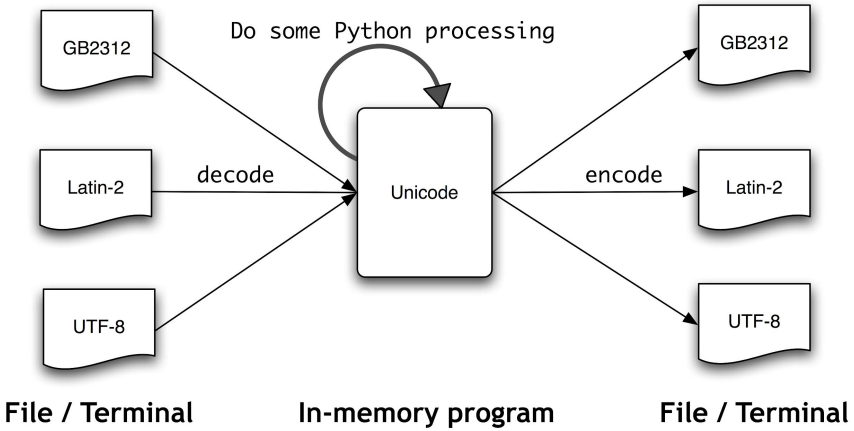
- In

```
>>> phrase = 'And now for something completely different'
>>> if 'thing' in phrase:
...     print('found "thing"')
found "thing"
```

- The Difference between Lists and Strings

Unicode

\u 를 앞에 붙여서 표현




```
path = nltk.data.find('corpora/unicode_samples/polish-lat2.txt')

f = open(path, encoding='latin2')
for line in f:
    line = line.strip()
    print(type(line),line)
    print(line.encode('unicode_escape'))

print(ord('ń'))
nacute = '\u0144'
print(nacute)
print(nacute.encode('utf8'))
print('\xf3')
print('\u00f3')
print('\xc5\x84')
```

Regular Expression
Tokenizer, Stemmer의 동작 원리

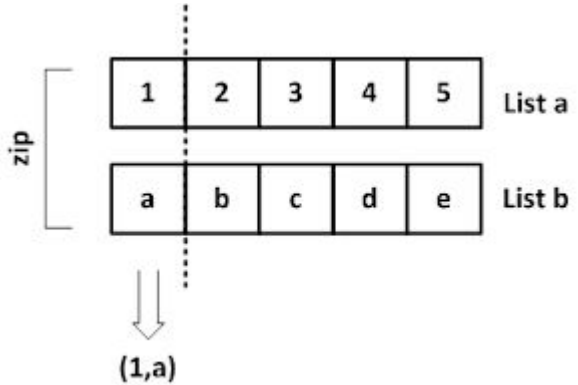
정규식

- Wikipedia: https://en.wikipedia.org/wiki/Regular_expression
- Python: https://docs.python.org/3/library/re.html
- 파이썬 코딩 도장: https://dojang.io/mod/page/view.php?id=1141
- Python Howto: https://docs.python.org/3/howto/regex.html

Zip 함수

zip 함수는 여러 List를 slice 할때 사용
김밥을 말때, 여러가지 재료를 묶어서 자르듯이,
여러 List를 김밥 말아서 짝을 지어준다.
그리고 iterator 등의 함수로 slice

```
a = [1,2,3,4,5]
b = ['a','b','c','d','e']

for x,y in zip (a,b):
  print x,y
```



http://pyengine.blogspot.com/2014/03/python-zip.html

Regular Expression for Detecting Word Patterns

- Many linguistic processing tasks involve pattern matching
  - endswith('ed') finds words ending with ed
- Regular expressions give us a more powerful and flexible method for describing the character patterns we are interested in.
- **import re**
- We use the Words Corpus again and preprocess it to remove any proper names

```
>>> import re
>>> wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()]

>>> [w for w in wordlist if re.search('ed$', w)]
['abaissed', 'abandoned', 'abased', 'abashed', 'abatised', 'abed', 'aborted', ...]

>>> [w for w in wordlist if re.search('^..j..t..$', w)]
['abjectly', 'adjuster', 'dejected', 'dejectly', 'injector', 'majestic', ...]

>>> [w for w in wordlist if re.search('^[ghi][mno][jlk][def]$', w)]
['gold', 'golf', 'hold', 'hole']
```

Regular Expression Meta-Characters

Operator	Behavior
.	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

Useful Applications of Regular Expressions

- We want to ignore word endings, and just deal with word **stems**
- re.**findall**() ("find all") method finds all (non-overlapping) matches of the given regular expression

```
def stem(word):
    regexp = r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'
    stem, suffix = re.findall(regexp, word)[0]
    return stem

>>> raw = """DENNIS: Listen, strange women lying in ponds
distributing swords... is no basis for a system of government.
Supreme executive power derives from
... a mandate from the masses, not from some farcical
aquatic ceremony."""
>>> tokens = word_tokenize(raw)
>>> [stem(t) for t in tokens]
['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'ly', 'in', 'pond',
'distribut', 'sword', 'i', 'no', 'basi', 'for', 'a', 'system', 'of', 'govern',
'.', 'Supreme', 'execut', 'power', 'deriv', 'from', 'a', 'mandate',
'from', 'the', 'mass', ',', 'not', 'from', 'some', 'farcical', 'aquatic',
'ceremony', '.']
```

Searching Tokenized Text

- You can use a special kind of regular expression for searching across **multiple words** in a text (where a text is a list of tokens). For example, "<a> <man>" finds all instances of *a man* in the text
- Using <> produce only the matched word (e.g. monied) and not the matched phrase (e.g. a monied man)

```
>>> from nltk.corpus import gutenberg, nps_chat
>>> moby =
nltk.Text(gutenberg.words('melville-moby_dick.txt'))
>>> moby.findall(r"<a> (<.*>) <man>")
monied; nervous; dangerous; white; pious; queer; good;
mature; white; Cape; great; wise; wise; butterless; white;
fiendish; pale; furious; better; certain; complete; dismasted;
younger; brave; brave; brave; brave
```

Normalizing Text

- By using lower(), we have normalized the text to lowercase so that the distinction between The and the is ignored
    - set(w.lower() for w in text)
- We can strip off any affixes, a task known as **stemming**. A further step is to make sure that the resulting form is a known word in a dictionary, a task known as **lemmatization**

First, we need to define the data we will use

```
>>> raw = """DENNIS: Listen, strange women lying in ponds distributing swords
... is no basis for a system of government.  Supreme executive power derives from
... a mandate from the masses, not from some farcical aquatic ceremony."""
>>> tokens = word_tokenize(raw)
```

Stemmers

- NLTK includes several off-the-shelf stemmers, and if you ever need a stemmer you should use one of these in preference to crafting your own using regular expressions, since these handle a wide range of irregular cases
- The Porter and Lancaster stemmers follow their own rules for stripping affixes
- Observe that the **Porter** stemmer correctly handles the word lying (mapping it to lie), while the Lancaster stemmer does not.

```
>>> porter = nltk.PorterStemmer()
>>> lancaster = nltk.LancasterStemmer()
>>> [porter.stem(t) for t in tokens]
['DENNI', ':', 'Listen', ',', 'strang', 'women', 'lie', 'in', 'pond',
'distribut', 'sword', 'is', 'no', 'basi', 'for', 'a', 'system', 'of',
'govern', '.', 'Suprem', 'execut', 'power', 'deriv', 'from', 'a',
'mandat', 'from', 'the', 'mass', ',', 'not', 'from', 'some',
'farcic', 'aquat', 'ceremoni', '.']
>>> [lancaster.stem(t) for t in tokens]
['den', ':', 'list', ',', 'strange', 'wom', 'lying', 'in', 'pond',
'distribut', 'sword', 'is', 'no', 'bas', 'for', 'a', 'system', 'of',
'govern', '.', 'suprem', 'execut', 'pow', 'der', 'from', 'a',
'mand', 'from', 'the', 'mass', ',', 'not',
'from', 'som', 'farc', 'aqu', 'ceremony', '.']
```

Lemmatization

- WordNet lemmatizer only removes affixes if **the resulting word is in its dictionary**
- This additional checking process makes the lemmatizer **slower** than the above stemmers
- Notice that it doesn't handle lying, but it converts women to woman

```
>>> wnl = nltk.WordNetLemmatizer()
>>> [wnl.lemmatize(t) for t in tokens]
['DENNIS', ':', 'Listen', ',', 'strange', 'woman', 'lying', 'in',
'pond','distributing', 'sword', 'is', 'no', 'basis', 'for', 'a',
'system', 'of', 'government', '.', 'Supreme', 'executive',
'power', 'derives', 'from', 'a', 'mandate', 'from', 'the',
'mass', ',', 'not', 'from', 'some', 'farcical',
'aquatic', 'ceremony', '.']
```

Regular Expressions for Tokenizing Text

- Tokenization is the task of cutting a string into identifiable linguistic units
- By using **re.findall**(), you can have much more control to tokenize text
- nltk.**regexp_tokenize**() is more efficient for this task, and avoids the need for special treatment of parentheses

```
>>> raw = """'When I'M a Duchess,' she said to herself, (not in a very hopeful tone
... though), 'I won't have any pepper in my kitchen AT ALL. Soup does very
... well without--Maybe it's always pepper that makes people hot-tempered,'..."""
>>> print(re.findall(r"\w+(?:[-']\w+)*|'|[-.(]+|\S\w*", raw))
["'", 'When', "I'M", 'a', 'Duchess', ',', "'", 'she', 'said', 'to', 'herself', ',', '(', 'not',
'in', 'a', 'very', 'hopeful', 'tone', 'though', ')', ',', "'", 'I', "won't", 'have', 'any',
'pepper', 'in', 'my', 'kitchen', 'AT', 'ALL', '.', 'Soup', 'does', 'very', 'well',
'without', '--', 'Maybe', "it's", 'always', 'pepper', 'that', 'makes', 'people',
'hot-tempered', ',', "'", '...']

>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Regular Expression symbols

Symbol	Function
\b	Word boundary (zero width)
\d	Any decimal digit (equivalent to [0-9])
\D	Any non-digit character (equivalent to [^0-9])
\s	Any whitespace character (equivalent to [\t\n\r\f\v])
\S	Any non-whitespace character (equivalent to [^ \t\n\r\f\v])
\w	Any alphanumeric character (equivalent to [a-zA-Z0-9_])
\W	Any non-alphanumeric character (equivalent to [^a-zA-Z0-9_])
\t	The tab character
\n	The newline character

ELIZA

nltk의 eliza 소스 코드

http://www.nltk.org/_modules/nltk/chat/eliza.html

```
from nltk.chat.util import Chat, reflections

pairs = (
  (r'I need (.*)',
  ( "Why do you need %1?",
    "Would it really help you to get %1?",
    "Are you sure you need %1?")),

  (r'Why don\'t you (.*)',
  ( "Do you really think I don't %1?",
    "Perhaps eventually I will %1.",
    "Do you really want me to %1?"))]

eliza_chatbot = Chat(pairs, reflections)

def eliza_chat():
    print("Therapist\n---------")
    print("Talk to the program")
    print('='*72)
    print("Hello.  How are you feeling today?")

    eliza_chatbot.converse()
```

Chatbot using NLTK

```
'''# nltk.chat.util
reflections = {
  "i am"       : "you test",
  "i was"      : "you were",
  "i"          : "you",
  "i'm"        : "you are",
  "i'd"        : "you would",
  "i've"       : "you have",
  "i'll"       : "you will",
  "my"         : "your",
  "you are"    : "I am",
  "you were"   : "I was",
  "you've"     : "I have",
  "you'll"     : "I will",
  "your"       : "my",
  "yours"      : "mine",
  "you"        : "me",
  "me"         : "you"
}'''
```

```
from nltk.chat.util import Chat, reflections
pairs = [
    [
        r'hi',
        ['hello', 'kamusta', 'mabuhay',]
    ],
    [
        r'(.*) (hungry|sleepy)',
        [  "%1 %2"   ]
    ],

    [
        r"My name is (.*)",
        ['hello %1',]
    ]
]

print("Hi how can I help you today?")
chat = Chat(pairs, reflections)
chat.converse()
```

https://medium.com/@allanmeriales/a-simple-chatbot-using-nltk-chat-640456dcdf72

Chatbots

- Cleverbot: http://www.cleverbot.com/
  - DB에서 가장 유사한 질문에 대한 답을 생성
  - 1997년에 공개
- Tacobot: 타코벨 주문을 받는 챗봇
- Quartz: 기사를 친구와 대화하듯 전달
- 채팅 데이터
  - Stanford https://nlp.stanford.edu/blog/a-new-multi-turn-multi-domain-task-oriented-dialogue-dataset/

Segmentation

- Tokenization is an instance of a more general problem of segmentation
- We will look at two other instances of this problem, which use radically different techniques to the ones we have seen so far in this chapter.

Sentence Segmentation

- Before tokenizing the text into words, we need to segment it into sentences
- NLTK includes the Punkt sentence segmenter (Kiss & Strunk, 2006)
- the quoted speech contains several sentences, and these have been split into individual strings
- Sentence segmentation is difficult
  - e.g., U.S.A.

```
>>> text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
>>> sents = nltk.sent_tokenize(text)
>>> pprint.pprint(sents[79:89])
['"Nonsense!"',
 'said Gregory, who was very rational when anyone else\nattempted paradox.',
 '"Why do all the clerks and navvies in the\n'
 'railway trains look so sad and tired, so very sad and tired?',
 'I will\ntell you.',
 'It is because they know that the train is going right.',
 'It\n'
 'is because they know that whatever place they have taken a ticket\n'
 'for that place they will reach.',
 'It is because after they have\n'
 'passed Sloane Square they know that the next station must be\n'
 'Victoria, and nothing but Victoria.',
 'Oh, their wild rapture!',
 'oh,\n'
 'their eyes like stars and their souls again in Eden, if the next\n'
 'station were unaccountably Baker Street!"',
 '"It is you who are unpoetical," replied the poet Syme.']
```

Word Segmentation

- We annotate each character with a boolean value to indicate **whether or not a word-break appears after the character**
- Now the segmentation task becomes a **search problem**: find the bit string that causes the text string to be correctly segmented into words
- We assume the learner is acquiring words and storing them in an internal lexicon

```
def segment(text, segs):
    words = []
    last = 0
    for i in range(len(segs)):
        if segs[i] == '1':
            words.append(text[last:i+1])
            last = i+1
    words.append(text[last:])
    return words

>>> text = "doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
>>> seg1 =
"000000000000000100000000001000000000000000100000000000"
>>> seg2 =
"010010010010000100100100001010010001001000010001001000 0"
>>> segment(text, seg1)
['doyouseethekitty', 'seethedoggy', 'doyoulikethekitty', 'likethedoggy']
>>> segment(text, seg2)
['do', 'you', 'see', 'the', 'kitty', 'see', 'the', 'doggy', 'do', 'you',
'like', 'the', 'kitty', 'like', 'the', 'doggy']
```

- Given a suitable lexicon, it is possible to reconstruct the source text as a sequence of lexical items
- **Objective function,** a scoring function whose value we will try to optimize, based on the size of the lexicon (number of characters in the words plus an extra delimiter character to mark the end of each word) and the amount of information needed to reconstruct the source text from the lexicon

```
def evaluate(text, segs):
    words = segment(text, segs)
    text_size = len(words)
    lexicon_size = sum(len(word) + 1 for word in set(words))
    return text_size + lexicon_size
>>> text = "doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
>>> seg1 = "000000000000000100000000001000000000000000000100000000000"
>>> seg3 = "000010010000001100100000011000010001000000110001000000 1"
>>> segment(text, seg3)
['doyou', 'see', 'thekitt', 'y', 'see', 'thedogg', 'y', 'doyou', 'like',
 'thekitt', 'y', 'like', 'thedogg', 'y']
>>> evaluate(text, seg3)
47
>>> evaluate(text, seg1)
64
```

SEGMENTATION

doyou	see	thekitt	y
see	thedogg	y	
doyou	like	thekitt	y
like	thedogg	y	

REPRESENTATION

LEXICON	DERIVATION
1. doyou	1 2 4 6
2. see	2 5 6
3. like	
4. thekitt	1 3 4 6
5. thedogg	
6. y	3 5 6

OBJECTIVE

LEXICON:
6+4+5+8+8+2 = 33

DERIVATION:
4+3+4+3 = 14

TOTAL:
33+14 = 47

Search for the pattern

- search for the pattern of zeros and ones that minimizes this objective function
- Notice that the best segmentation includes "words" like thekitty, since there's not enough evidence in the data to split this any further

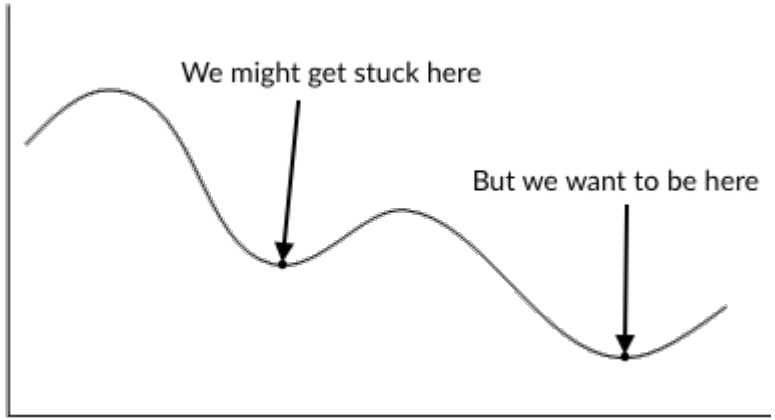
```
>>> text = "doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
>>> seg1 = "0000000000000001000000000010000000000000000100000000000"
>>> anneal(text, seg1, 5000, 1.2)
61 ['doyouseetheki', 'tty', 'see', 'thedoggy', 'doyouliketh', 'ekittylike', 'thedoggy']
59 ['doy', 'ouseetheki', 'ttysee', 'thedoggy', 'doy', 'o', 'ulikethekittylike', 'thedoggy']
57 ['doyou', 'seetheki', 'ttysee', 'thedoggy', 'doyou', 'liketh', 'ekittylike', 'thedoggy']
55 ['doyou', 'seethekit', 'tysee', 'thedoggy', 'doyou', 'likethekittylike', 'thedoggy']
54 ['doyou', 'seethekit', 'tysee', 'thedoggy', 'doyou', 'like', 'thekitty', 'like', 'thedoggy']
52 ['doyou', 'seethekittysee', 'thedoggy', 'doyou', 'like', 'thekitty', 'like', 'thedoggy']
43 ['doyou', 'see', 'thekitty', 'see', 'thedoggy', 'doyou', 'like', 'thekitty', 'like', 'thedoggy']
'0000100100000001001000000010000100010000000100010000000'
```

```
from random import randint

def flip(segs, pos): # pos의 값을 반전
    return segs[:pos] + str(1-int(segs[pos])) +
segs[pos+1:]

def flip_n(segs, n): #n개의 포인트를 랜덤하게 반전
    for i in range(n):
        segs = flip(segs, randint(0, len(segs)-1))
    return segs

def anneal(text, segs, iterations, cooling_rate):
    temperature = float(len(segs))
    while temperature > 0.5:
        best_segs, best = segs, evaluate(text, segs)
        for i in range(iterations):
            guess = flip_n(segs, round(temperature))
            score = evaluate(text, guess)
            if score < best:
                best, best_segs = score, guess
        score, segs = best, best_segs
        temperature = temperature / cooling_rate
        print(evaluate(text, segs), segment(text, segs))
    print()
    return segs
```

Annealing algorithm in the book

1) 초기해를 만든다.
2) 현재의 해를 랜덤하게 변화시킨다.
3) 새로운 해가 원래의 해보다 더 좋다면 바꾸고, 그렇지 않으면 무시한다.
4) 2~3의 과정을 n회 반복하되, 랜덤하게 변화하는 정도를 줄여준다

처음에는 나쁜 해로의 이동을 큰 확률로 주고, 점점 그 확률을 줄인다.



Simulated Annealing

1) 초기해를 만든다.
2) 현재의 해를 랜덤하게 변화시킨다.
3) 새로운 해가 원래의 해보다 더 좋다면 바꾸고, 그렇지 않으면 무시한다.
4) 새로운 해가 원래의 해보다 조금 나쁘고 (cost_delta) 랜덤하게 생성한 0-1 사이의 수가 acceptance probability보다 적으면 나쁜 해로 바꾼다.
5) 2~4의 과정을 n회 반복하되, 랜덤하게 변화하는 정도를 줄여준다

Algorithm 1. Simulated Annealing Algorithm [7]

Data: Cooling ratio r and length L
Result: approximate solution S
Initialize solution S;
Initialize temperature $T > 0$;
while *not yet* frozen **do**
  for $i \leftarrow 1$ **to** L **do**
    Pick a random neighbor S' of S;
    $\Delta \leftarrow (cost(S') - cost(S))$;
    if $\Delta \leqslant 0$ `// downhill move`
    then
      $S \leftarrow S'$;
    end
    if $\Delta \geq 0$ `// uphill move`
    then
      $S \leftarrow S'$ with probability $e^{-\Delta/T}$;
    end
  end
  $T \leftarrow rT$ (reduce temperature);
end

출처:http://blog.pluszero.ca/blog/2016/07/17/using-simulated-annealing-to-solve-logic-puzzles/

From Lists to Strings

```
>>> silly = ['We', 'called', 'him', 'Tortoise', 'because', 'he', 'taught', 'us', '.']
>>> ' '.join(silly)
'We called him Tortoise because he taught us .'

>>> for word in sorted(fdist):
...     print('{}->{};'.format(word, fdist[word]), end=' ')
cat->3; dog->4; snake->1;

>>> '{} wants a {} {}'.format ('Lee', 'sandwich', 'for lunch')
'Lee wants a sandwich for lunch'


>>> 'from {1} to {0}'.format('A', 'B')
'from B to A'
```

Writing Results to a File

```
output_file = open('output.txt', 'w')
words = set(nltk.corpus.genesis.words('english-kjv.txt'))
for word in sorted(words):
    print(word, file=output_file)
```

```
With open('output.txt', 'w') as f:
    f.write( 'TEST')
```

Summary

- We view a text as a list of words. A "raw text" is a potentially long string containing words and whitespace formatting
- Strings can be split into lists: 'Monty Python'.split() gives ['Monty', 'Python']. Lists can be joined into strings: '/'.join(['Monty', 'Python']) gives 'Monty/Python'
- read text from a file input.txt using text = open('input.txt').read(). We can read text from url using text = request.urlopen(url).read().decode('utf8')
- Tokenization is the segmentation of a text into basic units — or tokens — such as words and punctuation. Tokenization based on whitespace is inadequate for many applications because it bundles punctuation together with words. NLTK provides an off-the-shelf tokenizer nltk.word_tokenize().
- Lemmatization is a process that maps the various forms of a word (such as appeared, appears) to the canonical or citation form of the word, also known as the lexeme or lemma (e.g. appear).
- Regular expressions are a powerful and flexible method of specifying patterns. Once we have imported the re module, we can use re.findall() to find all substrings in a string that match a pattern.

Q&A