
# KoNLPy Documentation

*출시 0.5.1*

**Lucy Park**

**2018년 08월 03일**

# Contents

(https://travis-ci.org/konlpy/konlpy) (https://readthedocs.org/projects/konlpy/?badge=latest) KoNLPy("코엔엘파이"라고 읽습니다)는 한국어 정보처리를 위한 파이썬 패키지입니다. 설치법은 *이 곳을* (page 7) 참고해주세요.

NLP를 처음 시작하시는 분들은 *시작하기* (page 5) 에서 가볍게 기본 지식을 습득할 수 있으며, KoNLPy의 사용법 가이드는 *사용하기* (page 7), 각 모듈의 상세사항은 *API* (page 32) 문서에서 보실 수 있습니다.

```
>>> from konlpy.tag import Kkma
>>> from konlpy.utils import pprint
>>> kkma = Kkma()
>>> pprint(kkma.sentences(u'네, 안녕하세요. 반갑습니다.'))
[네, 안녕하세요..,
 반갑습니다.]
>>> pprint(kkma.nouns(u'질문이나 건의사항은 깃헙 이슈 트래커에 남겨주세요.'))
[질문,
 건의,
 건의사항,
 사항,
 깃헙,
 이슈,
 트래커]
>>> pprint(kkma.pos(u'오류보고는 실행환경, 에러메세지와함께 설명을 최대한상세히!^^'))
[(오류, NNG),
 (보고, NNG),
 (는, JX),
 (실행, NNG),
 (환경, NNG),
 (,, SP),
 (에러, NNG),
 (메세지, NNG),
 (와, JKM),
 (함께, MAG),
 (설명, NNG),
 (을, JKO),
 (최대한, NNG),
 (상세히, MAG),
 (!, SF),
 (^^, EMO)]
```

# CHAPTER 1

## 거인의 어깨 위에 서기

아름답지만 다소 복잡하기도한 한국어는 전세계에서 13번째로 많이 사용되는 언어입니다. (http://www.koreatimes.co.kr/www/news/nation/2014/05/116_157214.html) 복잡미묘한 한국어 텍스트에서 유용한 특성을 추출하기 위해 그 동안 *한국어 형태소 분석기* (page 29) 개발되기도 했습니다.

KoNLPy는 같은 기능을 하는 또 하나의 도구를 만들려는 것이 아닙니다. 그보다는, 현존하는 도구 위에 한 층을 쌓아 더 멀리 내다보려는 것입니다. 또한 KoNLPy는 파이썬 (http://python.org) 프로그래밍 언어로 사용할 수 있도록 만들어졌는데, 그것은 파이썬이 간결하고 우아한 문법구조, 강력한 스트링 연산 기능을 가지고 있을 뿐 아니라 크롤링, 웹프로그래밍, 그리고 데이터 분석을 수행할 수 있는 다양한 패키지를 사용할 수 있는 언어이기 때문입니다.

이 프로젝트에는 세 가지 철학이 있습니다:

- 사용법이 간단해야 한다.
- 누구나 쉽게 이용할 수 있어야 한다.
- *"인터넷 민주주의는 효과적이다."* (page 4)

위의 항목 중 하나라도 어긋나는 것이 있다면 제보 부탁드립니다 (https://github.com/konlpy/konlpy/issues).

CHAPTER 2

# 라이센스

KoNLPy는 오픈소스 소프트웨어이며, 아래의 라이센스를 채택하고 있습니다:

- GPL v3 또는 그 이상 (http://gnu.org/licenses/gpl.html)

라이센스에 따라 자유롭게 코드를 이용하실 수 있으며, 연구에 KoNLPy를 사용하신 경우 아래 논문을 인용해 주시기 바랍니다.

- 박은정, 조성준, "KoNLPy: 쉽고 간결한 한국어 정보처리 파이썬 패키지 (http://dmlab.snu.ac.kr/~lucypark/docs/2014-10-10-hclt.pdf)", 제 26회 한글 및 한국어 정보처리 학술대회 논문집, 2014.

BibTeX는 아래의 코드를 사용하시면 됩니다.:

```
@inproceedings{park2014konlpy,
  title={KoNLPy: Korean natural language processing in Python},
  author={Park, Eunjeong L. and Cho, Sungzoon},
  booktitle={Proceedings of the 26th Annual Conference on Human & Cognitive␣
↪Language Technology},
  address={Chuncheon, Korea},
  month={October},
  year={2014}
}
```

CHAPTER 3

# 참여하기

KoNLPy는 완벽하지 않습니다. 하지만 조금씩 지속적으로 발전시킬 예정이며, 누구나 개발 과정에 참여할 수 있습니다.

버그를 찾으셨나요? KoNLPy를 발전시킬 좋은 방법이 떠오르시나요? KoNLPy 깃헙 페이지 (https://github.com/konlpy/konlpy) 를 방문해서 아이디어를 제안해주시거나 (https://github.com/konlpy/konlpy/issues) 풀리퀘스트를 보내주세요. (https://github.com/konlpy/konlpy/pulls)

또, gitter (https://gitter.im/konlpy/konlpy) 의 대화에 참여하시거나, KoNLPy 메일링리스트에 (https://groups.google.com/forum/#!forum/konlpy) 가입해서 관련 정보를 받아보거나 궁금한 것에 대한 질문을 할 수도 있습니다.

무엇보다, 질문하는 것만으로도 엄청난 기여라는 점을 알아주세요! 질문은 개발 커뮤니티에 피드백을 주는 가장 직접적이면서도 쉬운 방법이고, 아이디어의 원천도 됩니다.

CHAPTER 4

# 시작하기

## 4.1 NLP란 무엇인가요?

NLP (Natural Language Processing, 자연어처리)는 텍스트에서 의미있는 정보를 분석, 추출하고 이해하는 일련의 기술집합입니다.

우리 일상에도 다양한 NLP 응용사례가 있습니다. 가령:

- 텍스트 요약 (ex: Summly (http://www.summly.com/index.html))
- 대화 시스템 (ex: Apple Siri (https://www.apple.com/ios/siri/))
- 기계 번역 (ex: Google Translate (http://translate.google.com))

그리고 물론, 검색엔진과 같은 정보검색 시스템 등이 있습니다. NLP의 기초에 대해 더 자세히 알기 위해서는 아래 책들을 참고하시기 바랍니다.

- Jurafsky et al., Speech and Language Processing (https://www.goodreads.com/book/show/908048), 2nd Edition, 2008.
- Manning and Schutze, Foundations of Statistical Natural Language Processing (https://www.goodreads.com/book/show/776349), 1999.

KoNLPy는 여러분이 한국어 텍스트를 이용하여 기초적인 NLP 작업을 수행하는데 도움을 드릴 것입니다. 영어 텍스트를 다루는 것에 관심 있으신 경우, NLTK (http://nltk.org) 를 참고해주시기 바랍니다.

## 4.2 이제 무엇을 준비하면 되나요?

KoNLPy를 사용하기 전에 다음의 몇 가지 준비가 필요합니다.

1. 언어에 대한 깊은 관심과 한국어에 대한 어느 정도의 이해
2. 기본적인 파이썬 프로그래밍 방법[1]

[1] 파이썬을 처음 접하시나요? 이미 다른 프로그래밍 언어를 사용할 줄 아는 경우 http://learnxinyminutes.com/docs/python/ 를 빠르게 훑거나, 프로그래밍이 처음이거나 시간을 조금 더 들일 용의가 있다면 The Hitchhiker's Guide (http://docs.python-guide.org/en/latest/) 또는 Learn Python the hard way (http://learnpythonthehardway.org/book/) 를 살펴보시기 바랍니다. 특히 마지막 링크의 경우 "깐깐하게 배우는 파이썬"이라는 이름으로 국내에 출판되었으니 참고하세요!

3. 좋은 텍스트 에디터와 터미널 (또는 파이썬 IDE)[2]
4. 파이썬이 설치된 컴퓨터 (https://wiki.python.org/moin/BeginnersGuide/Download)
5. 파이썬 패키지 매니저 pip (https://pypi.python.org/pypi/pip)

준비되셨나요? 이제 시작해봅시다.

[2] 많은 분들이 파이썬 프로그래밍을 할 때 Sublime Text 2 (http://www.sublimetext.com/) 을 사용하십니다. 또 다른 분들은 Vim과 터미널을 사용하는 것을 즐깁니다. 그 뿐 아니라 파이썬 프로그래밍이 가능한 수많은 텍스트 에디터 (http://tutorialzine.com/2012/07/battle-of-the-tools-which-is-the-best-code-editor/) 와 파이썬 IDE (http://pedrokroger.net/choosing-best-python-ide/) 가 있으니, 마음에 드는 것을 골라잡으세요!

CHAPTER 5

# 사용하기

## 5.1 설치하기

**주석:** For troubleshooting information, see these pages: Linux (https://github.com/konlpy/konlpy/issues?q=label%3Alinux). Mac OS (https://github.com/konlpy/konlpy/issues?q=label%3A"mac+os"). Windows (https://github.com/konlpy/konlpy/issues?q=label%3Awindows). Please record a "New Issue" (https://github.com/konlpy/konlpy/issues/new) if you have an error that is not listed.

### 5.1.1 우분투

1. Install dependencies

```
# Install Java 1.7 or up
$ sudo apt-get install g++ openjdk-7-jdk python-dev python3-dev
```

2. Install KoNLPy

```
$ pip install konlpy        # Python 2.x
$ pip3 install konlpy       # Python 3.x
```

3. MeCab 설치하기 (*선택사항*)

```
$ sudo apt-get install curl
$ bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/
↪master/scripts/mecab.sh)
```

### 5.1.2 CentOS

1. Install dependencies

```
$ sudo yum install gcc-c++ java-1.7.0-openjdk-devel python-devel

$ wget http://peak.telecommunity.com/dist/ez_setup.py                #␣
↪Python 2.x
```

(continues on next page)

(이전 페이지에서 계속)

```
$ sudo python ez_setup.py
$ sudo easy_install pip

$ wget https://www.python.org/ftp/python/3.4.3/Python-3.4.3.tar.xz  #␣
↪Python 3.x
$ tar xf Python-3.*
$ cd Python-3.*
$ ./configure
$ make # Build
$ sudo make altinstall
```

2. Install KoNLPy

```
$ pip install konlpy         # Python 2.x
$ pip3.4 install konlpy      # Python 3.x
```

3. MeCab 설치하기 (*선택사항*)

```
$ sudo yum install curl
$ bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/
↪master/scripts/mecab.sh)
```

### 5.1.3 맥 OS

1. Install KoNLPy

```
$ pip install konlpy          # Python 2.x
$ pip3 install konlpy         # Python 3.x
```

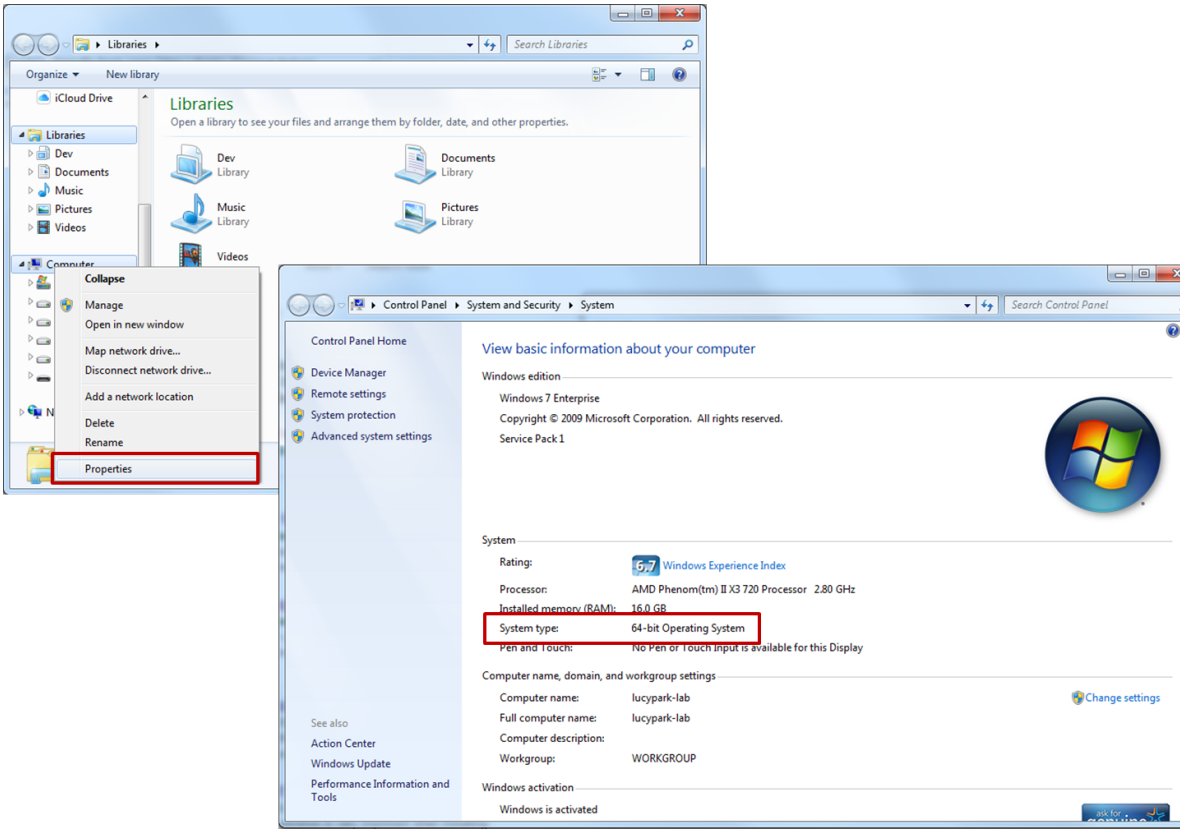
2. MeCab 설치하기 (*선택사항*)

```
$ bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/
↪master/scripts/mecab.sh)
```

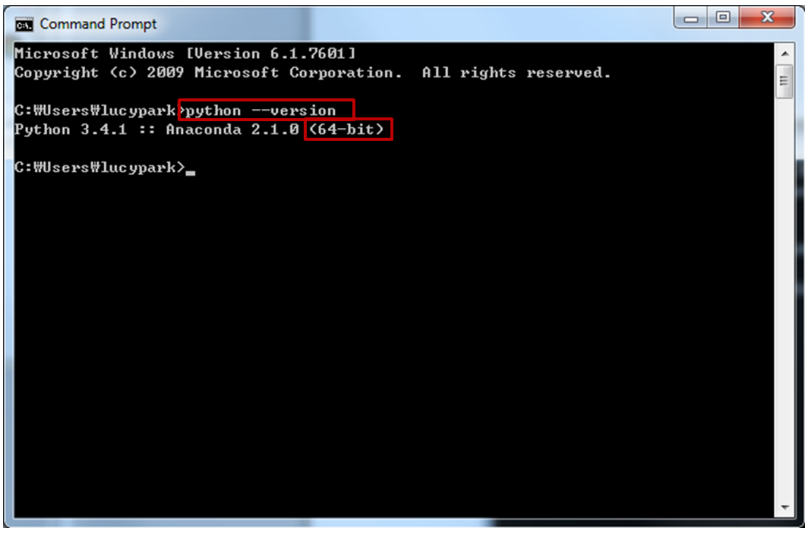
### 5.1.4 윈도우

1. **내 시스템에 설치된 파이썬의 "비트 수"가 OS의 비트 수와 일치하는지 확인해주세요. 예를 들어, 64비트 윈도우를 사용하**

    - 윈도우 비트 수 확인하는 법




- 파이썬 비트 수 확인하는 법




2. OS와 비트 수가 일치하고, 버젼이 1.7 이상인 자바가 설치되어 있나요? 만일 그렇지 않다면 JDK를 설치 (http://www.oracle.com/technetwork/java/javase/downloads/index.html) 합니다. 자바와 OS의 비트 수가 꼭 일치하도록 해주세요.

3. JAVA_HOME을 설정 (http://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/index.html) 합니다.

4. OS의 비트 수와 일치하는 JPype1 (>=0.5.7) (http://www.lfd.uci.edu/~gohlke/pythonlibs/#jpype) 를 설치해주세요. 32비트 OS에는 *win32*, 64비트 OS에는 *win-amd64* 파일을 사용하면 됩니다. whl 파일로 설치하는 경우에는 다음과 같이 명령프롬프트에서 pip을 업그레이드 (https://pip.pypa.io/en/stable/installing.html#upgrade-pip) 해주세요. (명령프롬프트는 Windows + r 을 누른 후 실행창에서 cmd를 입력하면 띄울 수 있습니다.)

```
> pip install --upgrade pip
> pip install JPype1-0.5.7-cp27-none-win_amd64.whl
```

5. 마지막으로, 명령프롬프트에서 KoNLPy를 설치합니다.

```
> pip install konlpy
```

**경고:**

- KoNLPy의 `Mecab()` 클래스는 윈도우에서 지원되지 않습니다.

## 5.2 형태소 분석 및 품사 태깅

**형태소 분석** 이란 형태소를 비롯하여, 어근, 접두사/접미사, 품사(POS, part-of-speech) 등 다양한 언어적 속성의 구조를 파악하는 것입니다.

**품사 태깅** 은 형태소의 뜻과 문맥을 고려하여 그것에 마크업을 하는 일입니다. 예를 들어:

```
가방에 들어가신다 -> 가방/NNG + 에/JKM + 들어가/VV + 시/EPH + ㄴ다/EFN
```

### 5.2.1 KoNLPy로 품사 태깅하기

KoNLPy에는 품사 태깅을 하기 위한 옵션이 여럿 있는데, 이들은 모두 문구(phrase)를 입력받아 태깅된 형태소를 출력하는 동일한 입출력 구조를 가집니다.

더 자세한 설명을 보기 위해서는 *tag Package* (page 32) 를 참고해주시기 바랍니다.

**더 보기:**

한국어 품사 태그 비교표 (https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSXI8/edit#gid=0)

각종 한국어 형태소 분석기의 품사 태그를 비교해 보세요.

### 5.2.2 품사 태깅 클래스 간 비교

이제 *tag Package* (page 32) 에 있는 태거들의 성능을 확인해볼까요? 실험은 4개의 코어가 있는 인텔 i7 CPU, 파이썬 2.7, KoNLPy 0.4.1을 이용해 수행되었습니다.

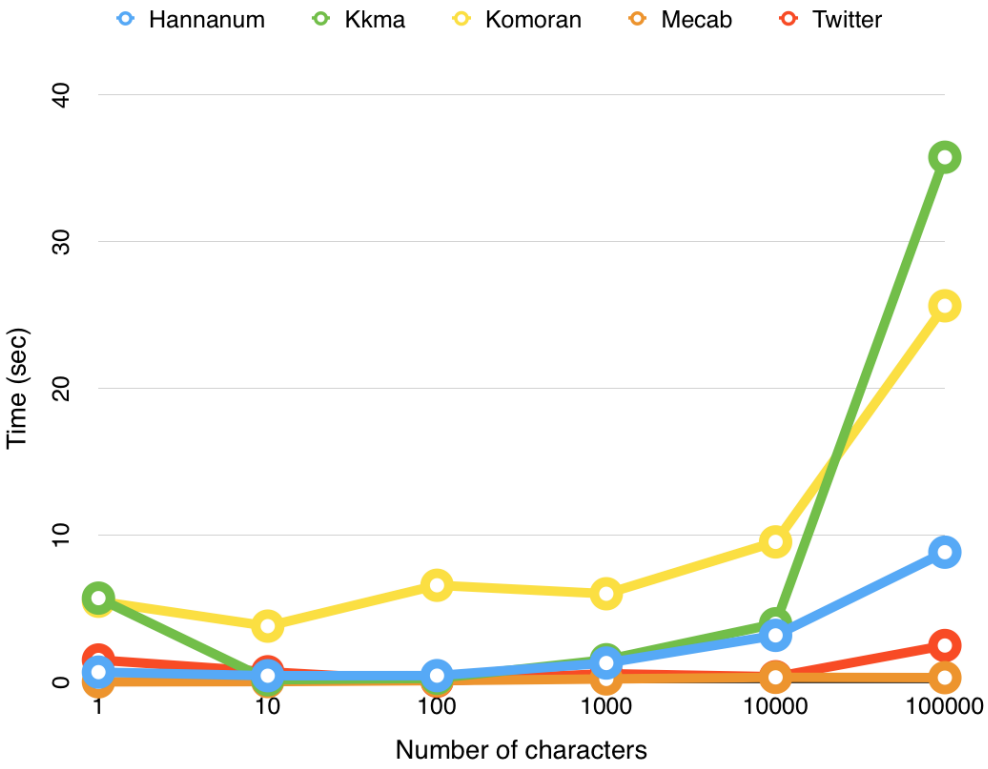
#### Time analysis[1]

1. 로딩 시간: 사전 로딩을 포함하여 클래스를 로딩하는 시간.
   - *Kkma* (page 33): 5.6988 *secs*
   - *Komoran* (page 34): 5.4866 *secs*
   - *Hannanum* (page 32): 0.6591 *secs*
   - *Okt* (page 36) (previous `Twitter`): 1.4870 *secs*
   - *Mecab* (page 35): 0.0007 *secs*
2. 실행시간: 10만 문자의 문서를 대상으로 각 클래스의 `pos` 메소드를 실행하는데 소요되는 시간.
   - *Kkma* (page 33): 35.7163 *secs*

[1] 각 형태소 분석기 배포판이 아닌, KoNLPy 내부 모듈 간 비교임에 유의해주시기 바랍니다.

- *Komoran* (page 34): 25.6008 *secs*
- *Hannanum* (page 32): 8.8251 *secs*
- *Okt* (page 36) (previous Twitter): 2.4714 *secs*
- *Mecab* (page 35): 0.2838 *secs*

문자의 개수를 늘려감에 따라 모든 클래스의 실행 시간은 기하급수적으로 증가합니다.




### 성능 분석

성능 검증은 몇 개의 샘플 문장을 비교하는 것으로 대체합니다.

1. *"아버지가방에들어가신다"*

   이 예시를 통해 띄어쓰기 알고리즘의 성능을 확인해볼 수 있습니다. 이상적인 경우, 이 예시에 대해서는 아버지 + 가방에 + 들어가신다 보다는 아버지가 + 방에 + 들어가신다 로 해석하는 것이 더 바람직하겠지요.

| Hannanum | Kkma | Komoran | Mecab | Twitter |
|---|---|---|---|---|
| 아버지가방에들어가 / N | 아버지 / NNG | 아버지가방에들어가신다 / NNP | 아버지 / NNG | 아버지 / Noun |
| 이 / J | 가방 / NNG | | 가 / JKS | 가방 / Noun |
| 시ㄴ다 / E | 에 / JKM | | 방 / NNG | 에 / Josa |
| | 들어가 / VV | | 에 / JKB | 들어가신 / Verb |
| | 시 / EPH | | 들어가 / VV | 다 / Eomi |
| | ㄴ다 / EFN | | 신다 / EP+EC | |

2. *"나는 밥을 먹는다" vs "하늘을 나는 자동차"*

   두 문장에서 "나는"에 집중해서 본다면, 각 분석기가 태깅할 때 단어의 의미와 주변부를 잘 살피는지 확인해볼 수 있습니다. 첫번째 문장에서 "나는"은 나/N + 는/J, 두번째 문장에서는 나(-ㄹ다)/V +

는/E 이 되는 것이 바람직합니다.

| Hannanum | Kkma | Komoran | Mecab | Twitter |
|---|---|---|---|---|
| 나 / N | 나 / NP | 나 / NP | 나 / NP | 나 / Noun |
| 는 / J | 는 / JX | 는 / JX | 는 / JX | 는 / Josa |
| 밥 / N | 밥 / NNG | 밥 / NNG | 밥 / NNG | 밥 / Noun |
| 을 / J | 을 / JKO | 을 / JKO | 을 / JKO | 을 / Josa |
| 먹 / P | 먹 / VV | 먹 / VV | 먹 / VV | 먹는 / Verb |
| 는다 / E | 는 / EPT | 는다 / EC | 는다 / EC | 다 / Eomi |
| | 다 / EFN | | | |

| Hannanum | Kkma | Komoran | Mecab | Twitter |
|---|---|---|---|---|
| 하늘 / N | 하늘 / NNG | 하늘 / NNG | 하늘 / NNG | 하늘 / Noun |
| 을 / J | 을 / JKO | 을 / JKO | 을 / JKO | 을 / Josa |
| 나 / N | 날 / VV | 나 / NP | 나 / NP | 나 / Noun |
| 는 / J | 는 / ETD | 는 / JX | 는 / JX | 는 / Josa |
| 자동차 / N | 자동차 / NNG | 자동차 / NNG | 자동차 / NNG | 자동차 / Noun |

3. *"아이폰 기다리다 지쳐 애플공홈에서 언락폰질러버렸다 6+ 128기가실버ㅋ"*

   각 분석기가 사전에 포함되지 않은 단어를 어떻게 해결하는지 확인해볼까요?

| Hannanum | Kkma | Komoran | Mecab | Twitter |
|---|---|---|---|---|
| 아이폰 / N | 아이 / NNG | 아이폰 / NNP | 아이폰 / NNP | 아이폰 / Noun |
| 기다리 / P | 폰 / NNG | 기다리 / VV | 기다리 / VV | 기다리 / Verb |
| 다 / E | 기다리 / VV | 다 / EC | 다 / EC | 다 / Eomi |
| 지치 / P | 다 / ECS | 지치 / VV | 지쳐 / VV+EC | 지쳐 / Verb |
| 어 / E | 지치 / VV | 어 / EC | 애플 / NNP | 애플 / Noun |
| 애플공홈 / N | 어 / ECS | 애플 / NNP | 공 / NNG | 공홈 / Noun |
| 에서 / J | 애플 / NNP | 공 / NNG | 홈 / NNG | 에서 / Josa |
| 언락폰질러버렸다 / N | 공 / NNG | 홈 / NNG | 에서 / JKB | 언락폰 / Noun |
| 6+ / N | 홈 / NNG | 에서 / JKB | 언락 / NNG | 질 / Verb |
| 128기가실벜 / N | 에서 / JKM | 언 / NNG | 폰 / NNG | 러 / Eomi |
| | 언락 / NNG | 락 / NNG | 질러버렸 / VV+EC+VX+EP | 버렸 / Verb |
| | 폰 / NNG | 폰 / NNG | 다 / EC | 다 / Eomi |
| | 질르 / VV | 지르 / VV | 6 / SN | 6 / Number |
| | 어 / ECS | 어 / EC | + / SY | + / Punctuation |
| | 버리 / VXV | 버리 / VX | 128 / SN | 128 / Number |
| | 었 / EPT | 었 / EP | 기 / NNG | 기 / Noun |
| | 다 / ECS | 다 / EC | 가 / JKS | 가 / Josa |
| | 6 / NR | 6 / SN | 실버 / NNP | 실버 / Noun |
| | + / SW | + / SW | ㅋ / UNKNOWN | ㅋ / KoreanParticle |
| | 128 / NR | 128기가실벜 / NA | | |
| | 기가 / NNG | | | |
| | 실버 / NNG | | | |
| | ㅋ / UN | | | |

**주석:** 이 코드를 (https://github.com/konlpy/konlpy/blob/master/docs/morph.py) 컴퓨터에서 실행하시면 위 비교 실험을 직접 수행하실 수 있습니다.

## 5.3 데이터

### 5.3.1 말뭉치

다음의 말뭉치(corpus)를 사용할 수 있습니다:

1. **kolaw: 한국 법률 말뭉치.**
    - constitution.txt
2. **kobill: 대한민국 국회 의안 말뭉치. 파일 ID는 의안 번호를 의미합니다.**
    - 1809890.txt - 1809899.txt

KoNLPy에 포함된 말뭉치의 사용은 *corpus Package* (page 37) 에서 더 자세하게 확인해볼 수 있습니다.

```
>>> from konlpy.corpus import kolaw
>>> c = kolaw.open('constitution.txt').read()
>>> print c[:10]
대한민국 헌법

유구한 역사와
>>> from konlpy.corpus import kobill
>>> d = kobill.open('1809890.txt').read()
>>> print d[:15]
지방공무원법 일부개정법률안
```

### 5.3.2 사전

사전은 대부분 *말뭉치* (page 30) 를 이용해 구축되었으며 *형태소 분석 및 품사 태깅* (page 10) 를 하는데 사용됩니다.

#### Hannanum 시스템 사전

KAIST 말뭉치를 이용해 생성된 사전. (4.7MB)

`./konlpy/java/data/kE/dic_system.txt` 에 위치해있으며, 아래에서 파일의 일부를 보실 수 있습니다.:

```
...
나라경제        ncn
나라기획        nqq
나라기획회장    ncn
나라꽃  ncn
나라님  ncn
나라도둑        ncn
나라따르        pvg
나라링링프로덕션        ncn
나라말  ncn
나라망신        ncn
나라박물관      ncn
나라발전        ncpa
나라별  ncn
나라부동산      nqq
나라사랑        ncn
나라살림        ncpa
나라시  nqq
나라시마        ncn
...
```

사용자 사전에 새로운 항목을 추가하기 위해서는 `./konlpy/java/data/kE/dic_user.txt` 를 수정하시면 됩니다.

#### Kkma 시스템 사전

세종 말뭉치를 이용해 생성된 사전. (32MB)

꼬꼬마 형태소 분석기의 `.jar` 파일 안에 위치해 있습니다. 사전 파일을 직접 보기 위해서는 꼬꼬마 미러 (https://github.com/e9t/kkma/tree/master/dic) 를 확인해보시기 바랍니다. `kcc.dic` 는 다음과 같은 형태를 가집니다.:

```
아니/IC
후우/IC
그래서/MAC
그러나/MAC
그러니까/MAC
그러면/MAC
그러므로/MAC
그런데/MAC
그리고/MAC
따라서/MAC
하지만/MAC
...
```

#### Mecab 시스템 사전

세종 말뭉치로 만들어진 CSV 형태의 사전. (346MB)

컴파일 된 사전은 `/usr/local/lib/mecab/dic/mecab-ko-dic` (또는 MeCab 설치 시 지정한 경로)에 있으며, 원본 사전은 소스코드 (https://bitbucket.org/eunjeon/mecab-ko-dic/src/ce04f82ab0083fb24e4e542e69d9e88a672c3325/seed/?at=master) 에서 확인하실 수 있습니다. `CoinedWord.csv` 파일의 일부를 아래에서 보실 수 있습니다.:

```
가오티,0,0,0,NNG,*,F,가오티,*,*,*,*,*
갑툭튀,0,0,0,NNG,*,F,갑툭튀,*,*,*,*,*
강퇴,0,0,0,NNG,*,F,강퇴,*,*,*,*,*
개드립,0,0,0,NNG,*,T,개드립,*,*,*,*,*
갠소,0,0,0,NNG,*,F,갠소,*,*,*,*,*
고퀄,0,0,0,NNG,*,T,고퀄,*,*,*,*,*
광삭,0,0,0,NNG,*,T,광삭,*,*,*,*,*
광탈,0,0,0,NNG,*,T,광탈,*,*,*,*,*
굉천,0,0,0,NNG,*,T,굉천,*,*,*,*,*
국을,0,0,0,NNG,*,T,국을,*,*,*,*,*
귀요미,0,0,0,NNG,*,F,귀요미,*,*,*,*,*
...
```

사용자 사전을 추가하기 위해서는 이 곳 (https://bitbucket.org/eunjeon/mecab-ko-dic/src/ce04f82ab0083fb24e4e542e69d9e88a672c3325/final/user-dic/?at=master) 을 참고해주시기 바랍니다.

---

**주석:** 시스템 사전과 사용자 사전 모두에 새로운 항목을 추가할 수 있지만, 두 경우에는 약간의 차이가 있습니다.

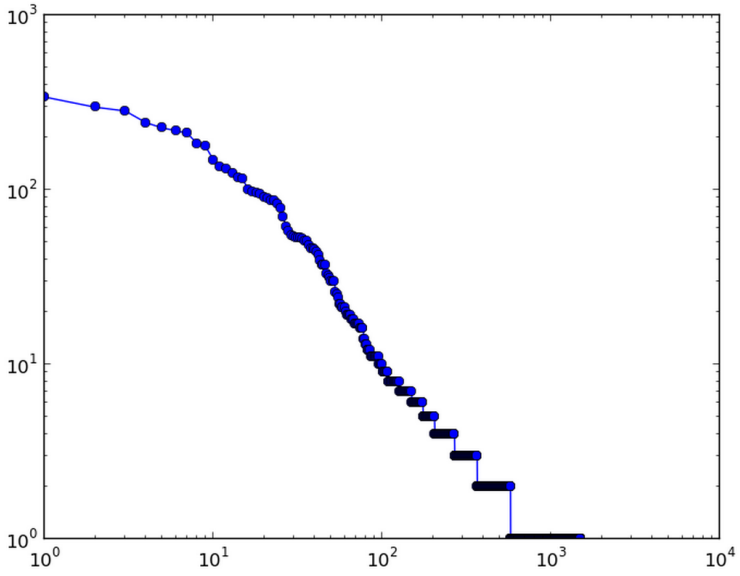
- 시스템 사전에 항목 추가하기: 사전 업데이트가 잦지 않은 경우, 속도 저하를 원하지 않는 경우.
- 사용자 사전에 항목 추가하기: 사전 업데이트가 잦은 경우, 관리자(root) 권한이 없는 경우.

---

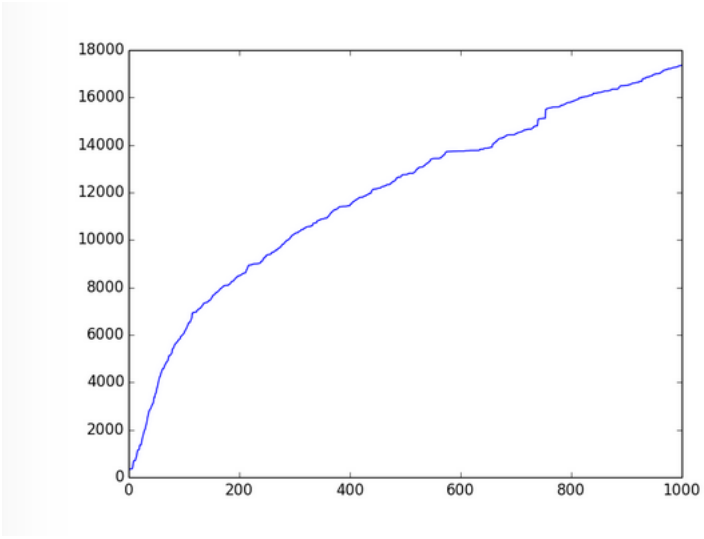
## 5.4 사용 예시

다음은 KoNLPy를 이용해 수행할 수 있는 몇몇 사용 예시입니다.

*문서 탐색하기* (page 18)




*말뭉치 탐색하기* (page 26)




*연어(collocation) 찾기* (page 19)

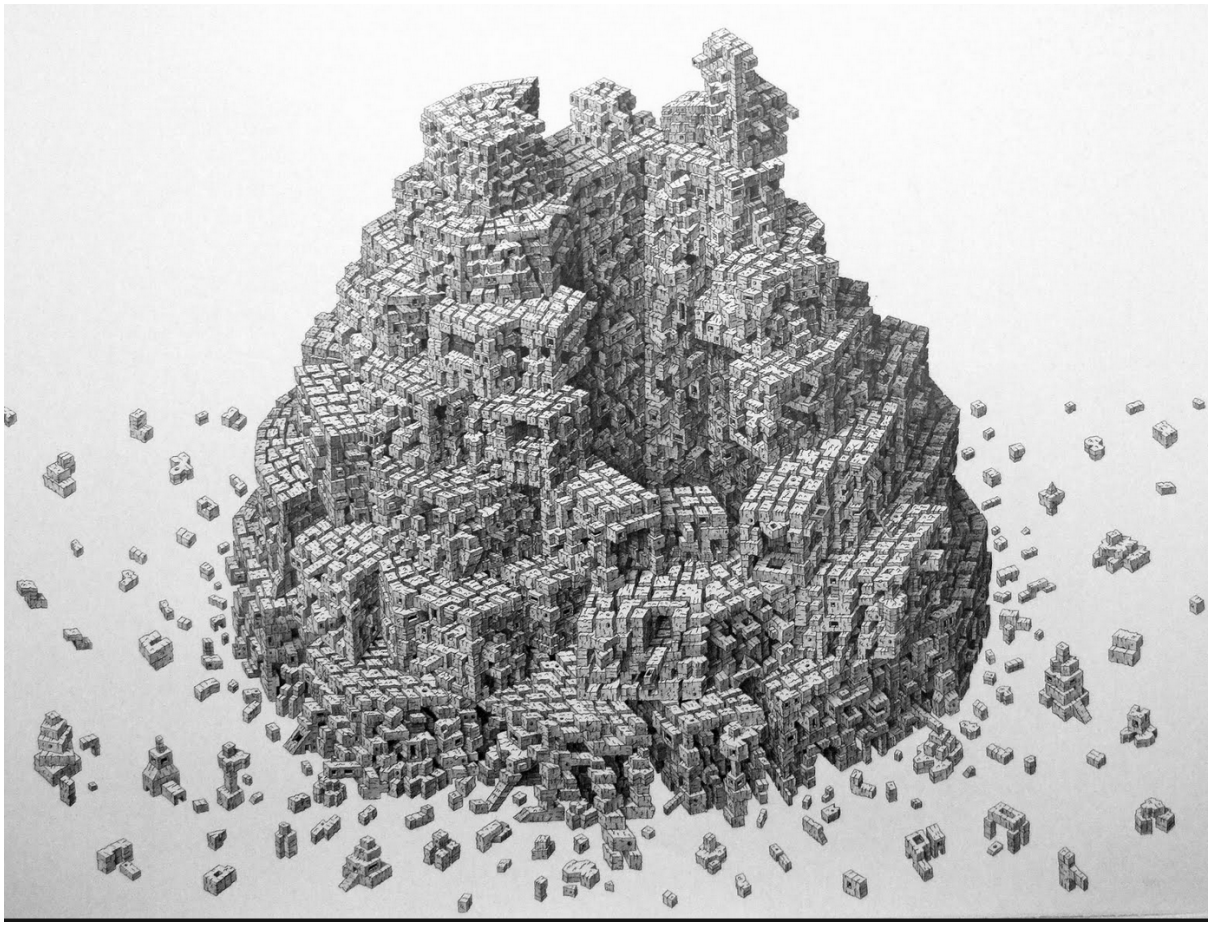
*구문 분석* (page 21)

*랜덤 텍스트 생성하기* (page 22)

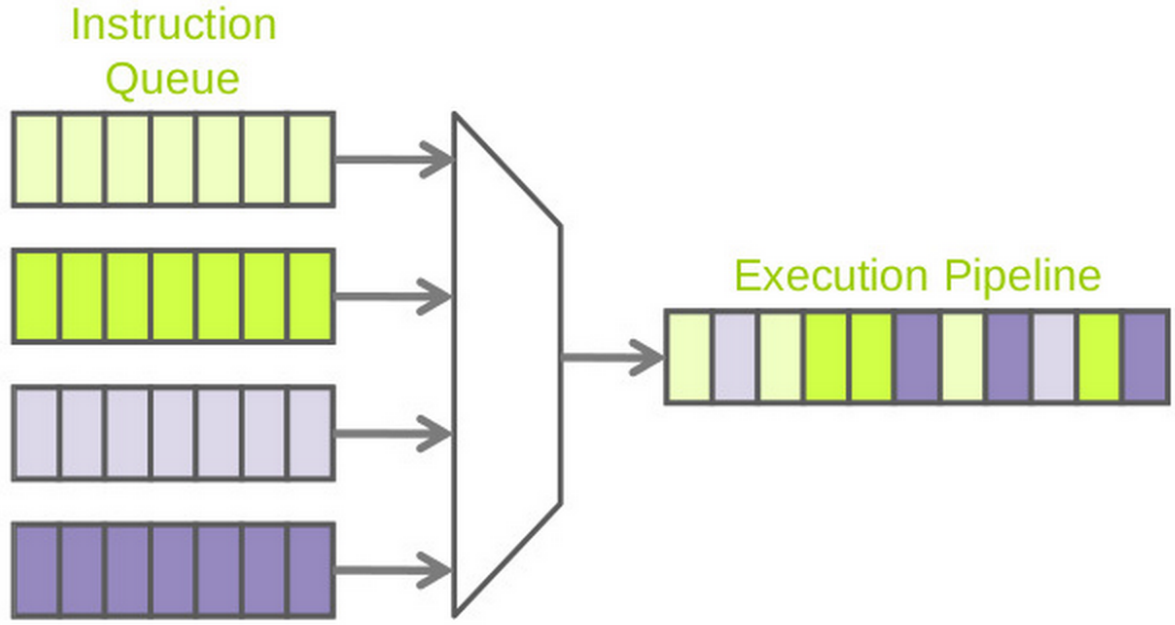
*워드클라우드 그리기* (page 24)

*KoNLPy를 이용한 멀티쓰레딩* (page 25)


S
NP
의 J
NP
6세 N
이하 N
초등학교 N
취학 N




행위
동물
인터넷
판매 · 전시 · 전달 · 상영
학대
게재
영상물
신설
금지
다음
고통
목적
누구
개정법률안
굶주림
정당
현행법
제1항부터
과태료
활동
적절
질병
표시
의식
동물보호
제외
일부
말
방치
신체
일체
위반
홍보


Instruction
Queue
Execution Pipeline

### 5.4.1 목록보기

#### 문서 탐색하기

처음 문서를 탐색할 때는 다양한 것을 확인해볼 수 있습니다:

- 개수 세기 (문자, 단어 등)
- 지프의 법칙(Zipf's laws) 확인해보기: $fr = k$
- 용례 찾기

```
#! /usr/bin/python2.7
# -*- coding: utf-8 -*-

from collections import Counter

from konlpy.corpus import kolaw
from konlpy.tag import Hannanum
from konlpy.utils import concordance, pprint
from matplotlib import pyplot


def draw_zipf(count_list, filename, color='blue', marker='o'):
    sorted_list = sorted(count_list, reverse=True)
    pyplot.plot(sorted_list, color=color, marker=marker)
    pyplot.xscale('log')
    pyplot.yscale('log')
    pyplot.savefig(filename)


doc = kolaw.open('constitution.txt').read()
pos = Hannanum().pos(doc)
cnt = Counter(pos)

print('nchars  :', len(doc))
print('ntokens :', len(doc.split()))
print('nmorphs :', len(set(pos)))
print('\nTop 20 frequent morphemes:'); pprint(cnt.most_common(20))
print('\nLocations of "대한민국" in the document:')
concordance(u'대한민국', doc, show=True)

draw_zipf(cnt.values(), 'zipf.png')
```

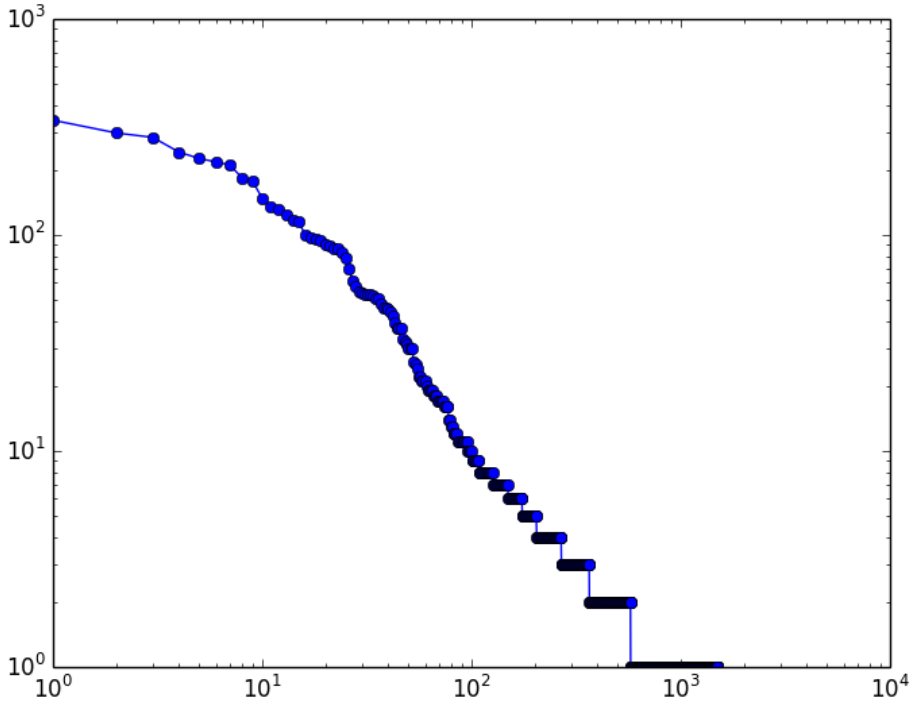
- 출력 결과:

```
nchars  : 19240
ntokens : 4178
nmorphs : 1501

Top 20 frequent morphemes:
[((의, J), 398),
 ((., S), 340),
 ((하, X), 297),
 ((에, J), 283),
 ((ㄴ다, E), 242),
 ((ㄴ, E), 226),
 ((이, J), 218),
 ((을, J), 211),
 ((은, J), 184),
 ((어, E), 177),
 ((를, J), 148),
 ((ㄹ, E), 135),
```

(continues on next page)

(이전 페이지에서 계속)

```
 ((/, S), 131),
 ((하, P), 124),
 ((는, J), 117),
 ((법률, N), 115),
 ((,, S), 100),
 ((는, E), 97),
 ((있, P), 96),
 ((되, X), 95)]

Locations of "대한민국" in the document:
0 대한민국헌법 유구한 역사와
9 대한국민은 3·1운동으로 건립된 대한민국임시정부의 법통과 불의에
98 총강 제1조 ① 대한민국은 민주공화국이다. ②대한민국의
100 ① 대한민국은 민주공화국이다. ②대한민국의 주권은 국민에게
110 나온다. 제2조 ① 대한민국의 국민이 되는
126 의무를 진다. 제3조 대한민국의 영토는 한반도와
133 부속도서로 한다. 제4조 대한민국은 통일을 지향하며,
147 추진한다. 제5조 ① 대한민국은 국제평화의 유지에
787 군무원이 아닌 국민은 대한민국의 영역안에서는 중대한
1836 파견 또는 외국군대의 대한민국 영역안에서의 주류에
3620 경제 제119조 ① 대한민국의 경제질서는 개인과
```




- **zipf.png:**

### 연어(collocation) 찾기

NLTK (http://nltk.org) 를 같이 활용하여 연어(collocation)을 찾을 수 있습니다.

3 음절 연어를 찾기 위해서는 *BigramAssocMeasures* 를 *TrigramAssocMeasures* 로 바꾸고, *BigramCollocationFinder* 를 *TrigramCollocationFinder* 로 바꾸시면 됩니다.

```
#! /usr/bin/python2.7
# -*- coding: utf-8 -*-
```

(continues on next page)

(이전 페이지에서 계속)

```
from konlpy.tag import Kkma
from konlpy.corpus import kolaw
from konlpy.utils import pprint
from nltk import collocations


measures = collocations.BigramAssocMeasures()
doc = kolaw.open('constitution.txt').read()

print('\nCollocations among tagged words:')
tagged_words = Kkma().pos(doc)
finder = collocations.BigramCollocationFinder.from_words(tagged_words)
pprint(finder.nbest(measures.pmi, 10)) # top 5 n-grams with highest PMI

print('\nCollocations among words:')
words = [w for w, t in tagged_words]
ignored_words = [u'안녕']
finder = collocations.BigramCollocationFinder.from_words(words)
finder.apply_word_filter(lambda w: len(w) < 2 or w in ignored_words)
finder.apply_freq_filter(3) # only bigrams that appear 3+ times
pprint(finder.nbest(measures.pmi, 10))

print('\nCollocations among tags:')
tags = [t for w, t in tagged_words]
finder = collocations.BigramCollocationFinder.from_words(tags)
pprint(finder.nbest(measures.pmi, 5))
```

- 출력 결과:

```
Collocations among tagged words:
[((가부, NNG), (동수, NNG)),
 ((강제, NNG), (노역, NNG)),
 ((경자, NNG), (유전, NNG)),
 ((고, ECS), (채취, NNG)),
 ((공무, NNG), (담임, NNG)),
 ((공중, NNG), (도덕, NNG)),
 ((과반, NNG), (수가, NNG)),
 ((교전, NNG), (상태, NNG)),
 ((그러, VV), (나, ECE)),
 ((기본적, NNG), (인권, NNG))]

Collocations among words:
[(현행, 범인),
 (형의, 선고),
 (내부, 규율),
 (정치적, 중립성),
 (누구, 든지),
 (회계, 연도),
 (지체, 없이),
 (평화적, 통일),
 (형사, 피고인),
 (지방, 자치)]

Collocations among tags:
[(XR, XSA),
 (JKC, VCN),
 (VCN, ECD),
 (ECD, VX),
 (ECD, VXV)]
```

### 구문 분석

*문장에 품사를 부착한 후에는* (page 10) 품사들을 조금 더 큰 단위의 묶음, 즉 구문으로 묶을 수 있습니다.

이곳에서는 형태소 분석된 결과와 `nltk.chunk.regexp.RegexpParser` (http://www.nltk.org/api/nltk.chunk.html#nltk.chunk.regexp.RegexpParser) 를 이용하여 간단하게 한국어 문장에서 명사구, 동사구, 형용사구를 찾는 법을 살펴보도록 하겠습니다.

```
#! /usr/bin/python2.7
# -*- coding: utf-8 -*-

import konlpy
import nltk

# POS tag a sentence
sentence = u'만 6세 이하의 초등학교 취학 전 자녀를 양육하기 위해서는'
words = konlpy.tag.Twitter().pos(sentence)

# Define a chunk grammar, or chunking rules, then chunk
grammar = """
NP: {<N.*>*<Suffix>?}   # Noun phrase
VP: {<V.*>*}            # Verb phrase
AP: {<A.*>*}            # Adjective phrase
"""
parser = nltk.RegexpParser(grammar)
chunks = parser.parse(words)
print("# Print whole tree")
print(chunks.pprint())

print("\n# Print noun phrases only")
for subtree in chunks.subtrees():
    if subtree.label()=='NP':
        print(' '.join((e[0] for e in list(subtree))))
        print(subtree.pprint())

# Display the chunk tree
chunks.draw()
```

여기에서는 세 가지 구문 문법(chunk grammar)을 정의해보았습니다. 먼저, 명사가 연속적으로 등장한 후 접미사(suffix)가 선택적으로 붙은 경우를 명사구(NP)로 정의하였습니다. (목적에 따라, 또 어떤 형태소 분석기를 사용하느냐에 따라 이 문법 규칙은 바뀌어야겠죠.) 마찬가지 방식으로 동사구(VP)와 형용사구(AP)를 정의하였습니다.

결과물은 트리(tree) 형태로 콘솔에 출력하거나, 이미지로 출력할 수 있습니다.

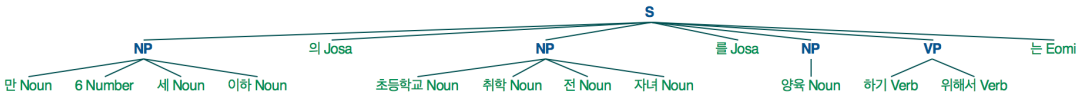
- 출력 결과:

```
# Print whole tree
(S
  (NP 만/Noun 6/Number 세/Noun 이하/Noun)
  의/Josa
  (NP 초등학교/Noun 취학/Noun 전/Noun 자녀/Noun)
  를/Josa
  (NP 양육/Noun)
  (VP 하기/Verb 위해서/Verb)
  는/Eomi)

# Print noun phrases only
만 6 세 이하
(NP 만/Noun 6/Number 세/Noun 이하/Noun)
초등학교 취학 전 자녀
(NP 초등학교/Noun 취학/Noun 전/Noun 자녀/Noun)
```

(continues on next page)

(이전 페이지에서 계속)

```
양육
(NP 양육/Noun)
```




- **chunking.png**

### 랜덤 텍스트 생성하기

한국어로 랜덤 텍스트를 생성해봅시다. 어떻게 할 수 있을까요?

가장 간단하게는 고양이 한 마리가 키보드를 밟고 지나가게 할 수 있습니다. 그렇게 하면 이와 유사한 결과가 나오겠죠[1]:

```
ㅕ키ㅖ츄ㅑㅋ응아 ㄲㄴㅇ 쿄ㅛㅍㄹㅎ튜허ㅓㅎㅠㅅㄴ마
```

하지만 이렇게 글자를 무작위로 나열하는 것은 아무런 의미를 가지지 않습니다. 일반적으로 한국어 텍스트는 초성, 중성, 종성으로 구성된 낱개의 음절들이 모여 단어를 이루고[2] 이 단어들이 다시 모여 문장을 이루는데, 글자들을 랜덤으로 타이핑해서는 음절이나 단어를 이룰 수 있는 확률은 극히 희박하겠죠. 그래서 이번에는 고양이에게 음절을 타이핑하는 방법을 (어떻게든) 가르쳐봅니다:

```
로켁불됴약 니앙굷 췡바이꿻노기
```

이렇게 하고 보니, 한국어에서 일반적으로 보기 힘든 음절들이 자주 등장했다는 사실을 알게 됩니다. 보통 한국어에서는 '이'가 '앙'보다는, 특히 '꿻'이나 '굷' 같은 음절보다 훨씬 빈번하게 등장하죠. 이번에는 일반적인 사용 빈도가 높은 음절들을 더 자주 등장시키도록 고양이를 다시 교육시킵니다:

```
다이는가 고다하에지 요그이데습
```

그렇게 해도 얻은 문자열이 온전한 문장이 되지는 않습니다. 그렇다면 각의 음절을 독립적으로(independently) 생성하지 말고, 선행하는 음절이 무엇이냐에 따라 현재의 음절을 생성해보면 어떨까요? 그러면 '하' 다음에는 '다'가 자주 등장할 것이고, '그' 다음에는 '리'와 '고'가 연속적으로 등장할 가능성이 높아질테니 우리가 실제로 쓰는 단어도 나올 수 있지 않을까요? 이런 과정을 수학적 용어로는 마코프 체인 (http://en.wikipedia.org/wiki/Markov_chain) 이라고 합니다:

```
국의의하되고인정부가는 요구한 대통령은 2조 사면 기밀과 헌법률로 위하의 위하며
```

직전의 음절 하나가 무엇이냐에 따라 현재의 음절을 생성한 위의 "문장"은 "bigram" 또는 "2-그램"으로 생성된 것이지만, 만일 더 말이 되는 문장을 만들고 싶다면 직전 음절 여러개를 보는 "3-그램" 또는 "4-그램"을 시도해볼 수도 있습니다. 또는, 음절 단위보다 한 단계 높은 형태소 단위에서 문장을 생성해보는 것도 가능합니다. 실제 코드로 한 번 해보죠.

> **경고:** 아래 코드는 파이썬3로 작동하고 파이썬2에서는 작동하지 않습니다. 터미널에서 *python3 generate.py* 를 입력하면 실행할 수 있습니다.

```
#! /usr/bin/python3
# -*- coding: utf-8 -*-

import bisect
import itertools
import random

import nltk
```

(continues on next page)

[1] This story would actually feature a monkey instead of a cat. Namely by the Infinite monkey theorem (http://en.wikipedia.org/wiki/Infinite_monkey_theorem).

[2] 유니코드 문자 코드표 (http://www.unicode.org/charts/)

(이전 페이지에서 계속)

```
from konlpy.corpus import kolaw
from konlpy.tag import Mecab # MeCab tends to reserve the original form of␣
↪morphemes


def generate_sentence(cfdist, word, num=15):
    sentence = []

    # Generate words until we meet a period
    while word!='.':
        sentence.append(word)

        # Generate the next word based on probability
        choices, weights = zip(*cfdist[word].items())
        cumdist = list(itertools.accumulate(weights))
        x = random.random() * cumdist[-1]
        word = choices[bisect.bisect(cumdist, x)]

    return ' '.join(sentence)


def calc_cfd(doc):
    # Calculate conditional frequency distribution of bigrams
    words = [w for w, t in Mecab().pos(doc)]
    bigrams = nltk.bigrams(words)
    return nltk.ConditionalFreqDist(bigrams)


if __name__=='__main__':
    nsents = 5 # Number of sentences
    initstr = u'국가' # Try replacing with u'국가', u'대통령', etc

    doc = kolaw.open('constitution.txt').read()
    cfd = calc_cfd(doc)

    for i in range(nsents):
        print('%d. %s' % (i, generate_sentence(cfd, initstr)))
```

- 출력 결과:

```
0. 국민 은 법률 로 인한 배상 은 특별 한 영장 을 청구 할 수 있 어서 최고 득표자 가 제출 한 유
일 한 때 에 의하 여 는 경우 를 선거 관리 할 수 없 을 포함 한 사항 은 청구 할 수 있 다
1. 국민 투표 의 범죄 에 의하 여 발언 하 지 아니한 회의 는 요건 은 1988 년 으로 대통령 이 의
결 한다
2. 국민 경제 자문 기구 를 타파 하 기 위하 여 긴급 한 형태 로 정한다
3. 국민 은 이 정하 는 헌법 시행 당시 의 심사 할 수 있 다
4. 국민 의 기본 질서 를 진다
```

아직 많이 부족하지만 처음에 고양이가 타이핑했던 것보다는 훨씬 나아졌네요! 특히 형태소와 형태소 사이에 공백이 있어서 덜 예뻐보이지만, 그 정도는 충분히 개선할 수 있을 것 같아요. 게다가 이 모델은 단 하나의 문서를 기반으로 구축한 것이지만, 보다 큰 말뭉치를 이용하는 경우에는 굳이 형태소 분석도 필요하지 않을 테고요. 그 외에도 이 모델을 발전시킬 수 있는 여지는 무궁무진하게 많습니다! 상상력을 발휘하여 다양하게 실험해보세요.

텍스트 생성에 대한 더 자세한 설명은 존 벤틀리의 생각하는 프로그래밍 (15장 3절) (http://www.cs.bell-labs.com/cm/cs/pearls/sec153.html) 의 내용을 참고해주세요.

한 걸음 더 나아가기 위해서는 언어 모델 (http://en.wikipedia.org/wiki/Language_model) 을 이용해 생성한 랜덤 텍스트가 통계적인 관점에서 얼마나 말이 되는지 평가해볼 수 있습니다.

### 워드클라우드 그리기

다음은 파이썬만으로 웹에서 국회 의안의 내용을 수집해서 이용해 명사를 추출한 후, 워드클라우드를 그리는 예시입니다.

의안 번호(i.e., `bill_num`)를 바꿔가며 워드클라우드가 어떻게 달라지는지 확인할수도 있습니다. (ex: '1904882', '1904883', 'ZZ19098', etc)

```
#! /usr/bin/python2.7
# -*- coding: utf-8 -*-

from collections import Counter
import urllib
import random
import webbrowser

from konlpy.tag import Hannanum
from lxml import html
import pytagcloud # requires Korean font support
import sys

if sys.version_info[0] >= 3:
    urlopen = urllib.request.urlopen
else:
    urlopen = urllib.urlopen


r = lambda: random.randint(0,255)
color = lambda: (r(), r(), r())

def get_bill_text(billnum):
    url = 'http://pokr.kr/bill/%s/text' % billnum
    response = urlopen(url).read().decode('utf-8')
    page = html.fromstring(response)
    text = page.xpath(".//div[@id='bill-sections']/pre/text()")[0]
    return text

def get_tags(text, ntags=50, multiplier=10):
    h = Hannanum()
    nouns = h.nouns(text)
    count = Counter(nouns)
    return [{ 'color': color(), 'tag': n, 'size': c*multiplier }\
                for n, c in count.most_common(ntags)]

def draw_cloud(tags, filename, fontname='Noto Sans CJK', size=(800, 600)):
    pytagcloud.create_tag_image(tags, filename, fontname=fontname, size=size)
    webbrowser.open(filename)


bill_num = '1904882'
text = get_bill_text(bill_num)
tags = get_tags(text)
print(tags)
draw_cloud(tags, 'wordcloud.png')
```

**주석:** PyPI에 업로드되어 있는 PyTagCloud (https://pypi.python.org/pypi/pytagcloud) 를 이용하면 워드클라우드에 한글이 제대로 표시되지 않을 수 있습니다. 이 경우, 직접 소스코드에 한글 폰트를 추가하거나, 이 곳 (https://github.com/e9t/PyTagCloud) 에서 미리 폰트가 추가된 버전을 다운받아 사용하시면 됩니다.


### KoNLPy를 이용한 멀티쓰레딩

시간이 오래 걸리는 태깅 작업을 수행하다보면 지루해지곤합니다. 그럴 때는 병렬처리 기법을 적용해보면 어떨까요? 파이썬은 멀티쓰레딩과 멀티프로세싱을 지원하며, 물론 그를 KoNLPy에 적용할 수 있습니다. 다음은 멀티쓰레딩을 활용한 예시입니다.

```
#! /usr/bin/python2.7
# -*- coding: utf-8 -*-

from konlpy.tag import Kkma
from konlpy.corpus import kolaw
from threading import Thread
import jpype

def do_concurrent_tagging(start, end, lines, result):
    jpype.attachThreadToJVM()
    l = [k.pos(lines[i]) for i in range(start, end)]
    result.append(l)
    return

if __name__=="__main__":
    import time

    print('Number of lines in document:')
    k = Kkma()
    lines = kolaw.open('constitution.txt').read().splitlines()
    nlines = len(lines)
    print(nlines)

    print('Batch tagging:')
    s = time.clock()
    result = []
    l = [k.pos(line) for line in lines]
    result.append(l)
    t = time.clock()
    print(t - s)
```

(continues on next page)

(이전 페이지에서 계속)

```
    print('Concurrent tagging:')
    result = []
    t1 = Thread(target=do_concurrent_tagging, args=(0, int(nlines/2), lines,␣
↪result))
    t2 = Thread(target=do_concurrent_tagging, args=(int(nlines/2), nlines, lines,␣
↪result))
    t1.start(); t2.start()
    t1.join(); t2.join()

    m = sum(result, []) # Merge results
    print(time.clock() - t)
```

- 출력 결과:

```
Number of lines in document:
356
Batch tagging:
37.758173
Concurrent tagging:
8.037602
```

얼마나 빨라지는지 보세요!

---

**주석:**

- **파이썬 병렬처리와 관련된 좋은 레퍼런스 몇 개:**
  - 장혜식, "파이썬은 멀티코어 줘도 쓰잘데기가 없나요?"에 대한 파이썬 2.6의 대답 (http://highthroughput.org/wp/python-multiprocessing/), 2008.
  - 하용호, 파이썬으로 클라우드 하고 싶어요 (http://www.slideshare.net/devparan/h3-2011-c6-python-and-cloud), 2011.

---

### 말뭉치 탐색하기

말뭉치(corpus)는 여러 문서의 집합체입니다.

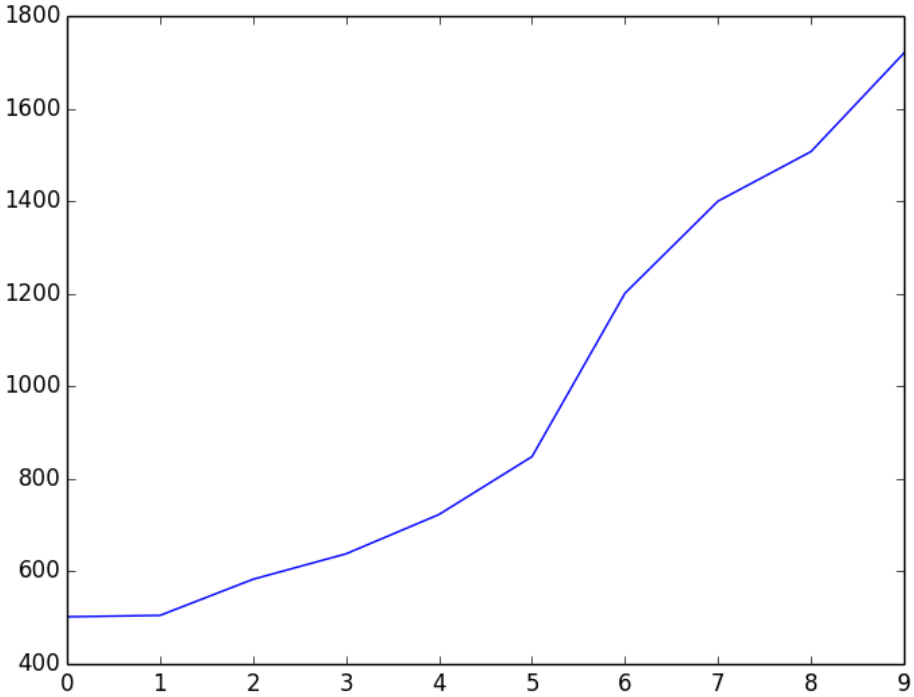
다음은 말뭉치의 크기를 늘려갈수록 등장하는 토큰의 개수가 로그함수적으로 늘어간다는 힙의 법칙(Heap's Law) (http://en.wikipedia.org/wiki/Heaps%27_law) 을 관찰하는 방법입니다.

```
#! /usr/bin/python
# -*- coding: utf-8 -*-

from konlpy.corpus import kobill
from konlpy.tag import Twitter; t = Twitter()
from matplotlib import pyplot as plt

pos = lambda x: ['/'.join(p) for p in t.pos(x)]
docs = [kobill.open(i).read() for i in kobill.fileids()]

# get global unique token counts
global_unique = []
global_unique_cnt = []
for doc in docs:
    tokens = pos(doc)
    unique = set(tokens)
    global_unique += list(unique)
    global_unique = list(set(global_unique))
    global_unique_cnt.append(len(global_unique))
```

(continues on next page)

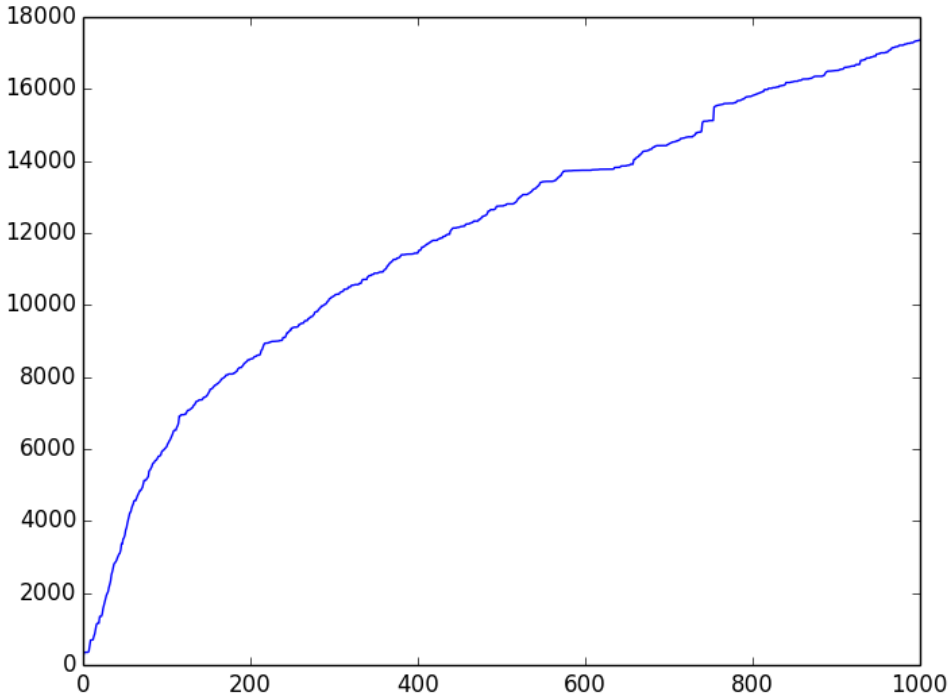
(이전 페이지에서 계속)

```
    print(len(unique), len(global_unique))

# draw heap
plt.plot(global_unique_cnt)
plt.savefig('heap.png')
```




- **heap.png:**

But why is our image not log-function shaped, as generally known? That is because the corpus we used is very small, and contains only 10 documents. To observe the Heap's law's log-function formatted curve, try experimenting with a larger corpus. Below is an image drawn from 1,000 Korean news articles. Of course, the curve will become smoother with a much larger corpus.




- **heap-1000.png:**

## 5.5 테스트하기

KoNLPy 결과물의 정합성을 확인하기 위한 테스트가 몇 가지 있습니다. 테스트를 수행하기 위해서는 다음의 코드를 실행해주세요.

```
$ pip install pytest
$ cd konlpy
$ python -m pytest test/* # for Python 2.x
$ python3 -m pytest test/* # for Python 3.x
```

**주석:** To see testing logs on KoNLPy, see here (https://docs.google.com/spreadsheets/d/1Ii_L9NF9gSLbsJOGqsf-zfqTtyhhthmJWNC2kgUDIsU/edit?usp=sharing).

**주석:** To see known bugs/issues, see here (https://github.com/konlpy/konlpy/labels/bug).

## 5.6 참고문헌

**주석:** Please modify this document (https://github.com/konlpy/konlpy/blob/master/docs/references.rst) if anything is erroneous or not included. Last updated at 2018년 08월 03일.

### 5.6.1 한국어 형태소 분석기

한국어 텍스트를 분석할 때 가장 기본적으로 행해야하는 것은 형태소 분석입니다. 이를 위해 다양한 프로그래밍 언어로 된 여러 라이브러리가 있습니다:

#### C/C++

- MeCab-ko (https://bitbucket.org/eunjeon/mecab-ko/) (2013) - By Yong-woon Lee and Youngho Yoo GPL LGPL BSD
- **UTagger (http://nlplab.ulsan.ac.kr/Demo/ProjectDemo.html) (2012) - By Joon-Choul Shin, Cheol-Young Ock* (Ulsan**
  - 신준철, 옥철영, 기분석 부분 어절 사전을 활용한 한국어 형태소 분석기 (A Korean Morphological Analyzer using a Pre-analyzed Partial Word-phrase Dictionary) (http://www.dbpia.co.kr/Journal/ArticleDetail/NODE01873335), 정보과학회논문지: 소프트웨어 및 응용, 제39권 제5호, 2012.
  - 신준철, 옥철영, 한국어 품사 및 동형이의어 태깅을 위한 단계별 전이모델 (A Stage Transition Model for Korean Part-of-Speech and Homograph Tagging) (http://www.dbpia.co.kr/Journal/ArticleDetail/NODE02033338), 정보과학회논문지: 소프트웨어 및 응용, 제39권 제11호, 2012.
  - slides (http://www.slideserve.com/mills/u-tagger-2013)
- **MACH (http://cs.sungshin.ac.kr/~shim/demo/mach.html) (2002) - By Kwangseob Shim (성신여대) custom**
  - Kwangseob Shim, Jaehyung Yang, MACH: A Supersonic Korean Morphological Analyzer (http://www.aclweb.org/anthology/C02-1092), ACL, 2002.
- **KTS (http://wiki.kldp.org/wiki.php/KTS) (1995) - By 이상호, 서정연, 오영환 (KAIST) GPL v2**
  - 이상호, KTS: Korean Tagging System Manual (Version 0.9) (https://wiki.kldp.org/wiki.php/KTS?action=download&value=ktsmanual.pdf)
  - 김재훈, 서정연, 자연언어 처리를 위한 한국어 품사 태그 (A Korean part-of-speech tag set for natural language processing) (https://wiki.kldp.org/wiki.php/KTS?action=download&value=tagset.pdf), 1993.
  - Created at 1995, released at 2002.[1]

#### Java/Scala

- twitter-korean-text (https://github.com/twitter/twitter-korean-text/) (2014) - By Will Hohyon Ryu (Twitter) Apache v2
- KOMORAN (http://shineware.tistory.com/tag/KOMORAN) (2013) - By 신준수 (shineware) Apache v2
- **KKMA (http://kkma.snu.ac.kr) (2010) - By Sang-goo Lee*, Dongjoo Lee (http://therocks.tistory.com), et al. (Seoul Nat**
  - 이동주, 연종흠, 황인범, 이상구, 꼬꼬마: 관계형 데이터베이스를 활용한 세종 말뭉치 활용 도구 (http://ids.snu.ac.kr/w/images/f/f8/CPL2010-therocks.pdf), 정보과학회논문지: 컴퓨팅의 실제 및 레터, Volume 16, No.11, 2010.
- **Arirang (http://cafe.naver.com/korlucene) (2009) - By SooMyung Lee Apache v2**
  - code (http://sourceforge.net/projects/lucenekorean)
- **HanNanum (http://semanticweb.kaist.ac.kr/home/index.php/HanNanum) (1999) - By Key-Sun Choi* et al. (KAIST) G**
  - code (http://kldp.net/projects/hannanum/src), docs (http://semanticweb.kaist.ac.kr/research/hannanum/j/javadoc/)

[1] https://wiki.kldp.org/wiki.php/KTS

#### 파이썬

- **KoNLPy (http://konlpy.org) (2014) GPL v3+**
  - By Lucy Park (Seoul National University)
  - Wrapper for Hannanum, KKMA, KOMORAN, twitter-korean-text, MeCab-ko
  - Tools for Hangul/Korean manipulation
- **UMorpheme (https://pypi.python.org/pypi/UMorpheme) (2014) MIT**
  - 김경훈 (UNIST)
  - Wrapper for MeCab-ko for online usage

#### R

- **KoNLP (https://github.com/haven-jeon/KoNLP) (2011) GPL v3**
  - 전희원
  - Wrapper for Hannaum

#### 그 외

- K-LIWC (http://k-liwc.ajou.ac.kr/) (아주대)
- **KRISTAL-IRMS (http://www.kristalinfo.com/) (KISTI)**
  - 개발 후기 (http://spasis.egloos.com/9507)
- Korean XTAG (http://www.cis.upenn.edu/~xtag/koreantag/) (UPenn)
- HAM (http://nlp.kookmin.ac.kr/HAM/kor/ham-intr.html) (국민대)
- POSTAG/K (http://nlp.postech.ac.kr/~project/DownLoad/k_api.html) (POSTECH)

### 5.6.2 말뭉치

- Korean Universal Dependency (UD) Treebank (https://github.com/UniversalDependencies/UD_Korean), 2013.
- **고려대학교 한국어 말뭉치, 1995**
  - 1970-90년대 한국어에 대한 1000만 어절
- **HANTEC 2.0 (http://www.kristalinfo.com/download/#hantec), KISTI & 충남대, 1998-2003.**
  - 12만 개의 테스트 문서 (237MB)
  - QA를 위한 50개의 TREC 형태 질의
- **HKIB-40075 (http://www.kristalinfo.com/TestCollections/readme_hkib.html), KISTI & 한국일보, 2002.**
  - 텍스트 분류를 위한 40,075 테스트 문서 (88MB)
- KAIST Corpus (http://semanticweb.kaist.ac.kr/home/index.php/KAIST_Corpus), KAIST, 1997-2005.
- Sejong Corpus (http://www.sejong.or.kr/), National Institute of the Korean Language, 1998-2007.
- **연세 말뭉치, 연세대, 1987.**
  - 1960년 이후 한국어에 대한 4200만 어절
- BoRA 언어자원은행 (http://semanticweb.kaist.ac.kr/org/bora/), KAIST

### 5.6.3 다른 NLP 도구

- **Hangulize (http://www.hangulize.org/) - By Heungsub Lee Python**
  - Hangul transcription tool to 38+ languages
- **Hanja (https://github.com/suminb/hanja) - By Sumin Byeon Python**
  - Hanja to hangul transcriptor
- **Jamo (http://github.com/JDong820/python-jamo) - By Joshua Dong Python**
  - Hangul syllable decomposition and synthesis
- **KoreanParser (http://semanticweb.kaist.ac.kr/home/index.php/KoreanParser) - By DongHyun Choi, Jungyeul Park, K**
  - 언어 파서
- **Korean (http://pythonhosted.org/korean) - By Heungsub Lee Python**
  - Package for attaching particles (josa) in sentences
- **go_hangul (https://github.com/suapapa/go_hangul) (2012) - By Homin Lee Go BSD**
  - Tools for Hangul manipulation [docs] (https://godoc.org/github.com/suapapa/go_hangul)
- Speller (http://speller.cs.pusan.ac.kr/) (부산대)

# CHAPTER 6

# API

## 6.1 konlpy Package

### 6.1.1 Subpackages

#### tag Package

**주석:** Initial runs of each class method may require some time to load dictionaries (< 1 *min*). Second runs should be faster.

##### `Hannanum` Class

**class** konlpy.tag._hannanum.**Hannanum**(*jvmpath=None*, *max_heap_size=1024*)

Wrapper for JHannanum (http://semanticweb.kaist.ac.kr/home/index.php/HanNanum).

JHannanum is a morphological analyzer and POS tagger written in Java, and developed by the Semantic Web Research Center (SWRC) (http://semanticweb.kaist.ac.kr/) at KAIST since 1999.

```
>>> from konlpy.tag import Hannanum
>>> hannanum = Hannanum()
>>> print(hannanum.analyze(u'롯데마트의 흑마늘 양념 치킨이 논란이 되고 있다.'))
[[[('롯데마트', 'ncn'), ('의', 'jcm')], [('롯데마트의', 'ncn')], [('롯데마트', 'nqq
↪'), ('의', 'jcm')], [('롯데마트의', 'nqq')]], [[('흑마늘', 'ncn')], [('흑마늘',
↪'nqq')]], [[('양념', 'ncn')]], [[('치킨', 'ncn'), ('이', 'jcc')], [('치킨', 'ncn
↪'), ('이', 'jcs')], [('치킨', 'ncn'), ('이', 'ncn')]], [[('논란', 'ncpa'), ('이
↪', 'jcc')], [('논란', 'ncpa'), ('이', 'jcs')], [('논란', 'ncpa'), ('이', 'ncn
↪')]], [[('되', 'nbu'), ('고', 'jcj')], [('되', 'nbu'), ('이', 'jp'), ('고',
↪'ecc')], [('되', 'nbu'), ('이', 'jp'), ('고', 'ecs')], [('되', 'nbu'), ('이',
↪'jp'), ('고', 'ecx')], [('되', 'paa'), ('고', 'ecc')], [('되', 'paa'), ('고',
↪'ecs')], [('되', 'paa'), ('고', 'ecx')], [('되', 'pvg'), ('고', 'ecc')], [('되
↪', 'pvg'), ('고', 'ecs')], [('되', 'pvg'), ('고', 'ecx')], [('되', 'px'), ('고
↪', 'ecc')], [('되', 'px'), ('고', 'ecs')], [('되', 'px'), ('고', 'ecx')]], [[(
↪'있', 'paa'), ('다', 'ef')], [('있', 'px'), ('다', 'ef')]], [[('.', 'sf')], [(
↪'.', 'sy')]]]
```

(continues on next page)

(이전 페이지에서 계속)

```
>>> print(hannanum.morphs(u'롯데마트의 흑마늘 양념 치킨이 논란이 되고 있다.'))
['롯데마트', '의', '흑마늘', '양념', '치킨', '이', '논란', '이', '되', '고', '있', '다
↪', '.']
>>> print(hannanum.nouns(u'다람쥐 헌 쳇바퀴에 타고파'))
['다람쥐', '쳇바퀴', '타고파']
>>> print(hannanum.pos(u'웃으면 더 행복합니다!'))
[('웃', 'P'), ('으면', 'E'), ('더', 'M'), ('행복', 'N'), ('하', 'X'), ('ㅂ니다', 'E
↪'), ('!', 'S')]
```

**매개 변수**

- **jvmpath** – The path of the JVM passed to *init_jvm()* (page 39).
- **max_heap_size** – Maximum memory usage limitation (Megabyte) *init_jvm()* (page 39).

**analyze** (*phrase*)
Phrase analyzer.

This analyzer returns various morphological candidates for each token. It consists of two parts: 1) Dictionary search (chart), 2) Unclassified term segmentation.

**morphs** (*phrase*)
Parse phrase to morphemes.

**nouns** (*phrase*)
Noun extractor.

**pos** (*phrase*, *ntags=9*, *flatten=True*, *join=False*)
POS tagger.

This tagger is HMM based, and calculates the probability of tags.

**매개 변수**

- **ntags** – The number of tags. It can be either 9 or 22.
- **flatten** – If False, preserves eojeols.
- **join** – If True, returns joined sets of morph and tag.

### Kkma Class

**class** konlpy.tag._kkma.**Kkma** (*jvmpath=None*, *max_heap_size=1024*)
Wrapper for Kkma (http://kkma.snu.ac.kr).

Kkma is a morphological analyzer and natural language processing system written in Java, developed by the Intelligent Data Systems (IDS) Laboratory at SNU (http://snu.ac.kr).

```
>>> from konlpy.tag import Kkma
>>> kkma = Kkma()
>>> print(kkma.morphs(u'공부를 하면할수록 모르는게 많다는 것을 알게 됩니다.'))
['공부', '를', '하', '면', '하', 'ㄹ수록', '모르', '는', '것', '이', '많', '다는', '것
↪', '을', '알', '게', '되', 'ㅂ니다', '.']
>>> print(kkma.nouns(u'대학에서 DB, 통계학, 이산수학 등을 배웠지만...'))
['대학', '통계학', '이산', '이산수학', '수학', '등']
>>> print(kkma.pos(u'다 까먹어버렸네요?ㅋㅋ'))
[('다', 'MAG'), ('까먹', 'VV'), ('어', 'ECD'), ('버리', 'VXV'), ('었', 'EPT'), (
↪'네요', 'EFN'), ('?', 'SF'), ('ㅋㅋ', 'EMO')]
>>> print(kkma.sentences(u'그래도 계속 공부합니다. 재밌으니까!'))
['그래도 계속 공부합니다.', '재밌으니까!']
```

**경고:** There are reports that `Kkma()` is weak for long strings with no spaces between words. See issue #73 (https://github.com/konlpy/konlpy/issues/73) for details.

**매개 변수**

- **jvmpath** – The path of the JVM passed to *init_jvm()* (page 39).
- **max_heap_size** – Maximum memory usage limitation (Megabyte) *init_jvm()* (page 39).

**morphs** (*phrase*)
Parse phrase to morphemes.

**nouns** (*phrase*)
Noun extractor.

**pos** (*phrase*, *flatten=True*, *join=False*)
POS tagger.

**매개 변수**

- **flatten** – If False, preserves eojeols.
- **join** – If True, returns joined sets of morph and tag.

**sentences** (*phrase*)
Sentence detection.

### Komoran Class

**class** konlpy.tag._komoran.**Komoran**(*jvmpath=None*, *userdic=None*, *modelpath=None*, *max_heap_size=1024*)
Wrapper for KOMORAN (https://github.com/shin285/KOMORAN).

KOMORAN is a relatively new open source Korean morphological analyzer written in Java, developed by Shineware (http://shineware.co.kr), since 2013.

```
>>> cat /tmp/dic.txt  # Place a file in a location of your choice
코모란      NNP
오픈소스     NNG
바람과 함께 사라지다      NNP
>>> from konlpy.tag import Komoran
>>> komoran = Komoran(userdic='/tmp/dic.txt')
>>> print(komoran.morphs(u'우왕 코모란도 오픈소스가 되었어요'))
['우왕', '코모란', '도', '오픈소스', '가', '되', '었', '어요']
>>> print(komoran.nouns(u'오픈소스에 관심 많은 멋진 개발자님들!'))
['오픈소스', '관심', '개발자']
>>> print(komoran.pos(u'혹시 바람과 함께 사라지다 봤어?'))
[('혹시', 'MAG'), ('바람과 함께 사라지다', 'NNP'), ('보', 'VV'), ('았', 'EP'), ('어',
↪ 'EF'), ('?', 'SF')]
```

**매개 변수**

- **jvmpath** – The path of the JVM passed to *init_jvm()* (page 39).
- **userdic** – The path to the user dictionary.

  This enables the user to enter custom tokens or phrases, that are mandatorily assigned to tagged as a particular POS. Each line of the dictionary file should consist of a token or phrase, followed by a POS tag, which are delimited with a <*tab*> character.

  An example of the file format is as follows:

```
바람과 함께 사라지다 NNG
바람과 함께 NNP
자연어 NNG
```

If a particular POS is not assigned for a token or phrase, it will be tagged as NNP.

- **modelpath** – The path to the Komoran HMM model.
- **max_heap_size** – Maximum memory usage limitation (Megabyte) *init_jvm()* (page 39).

**morphs** (*phrase*)
Parse phrase to morphemes.

**nouns** (*phrase*)
Noun extractor.

**pos** (*phrase*, *flatten=True*, *join=False*)
POS tagger.

매개 변수

- **flatten** – If False, preserves eojeols.
- **join** – If True, returns joined sets of morph and tag.

### Mecab Class

**경고:** `Mecab()` is not supported on `Windows`.

**class** konlpy.tag._mecab.**Mecab**(*dicpath='/usr/local/lib/mecab/dic/mecab-ko-dic'*)
Wrapper for MeCab-ko morphological analyzer.

MeCab (https://code.google.com/p/mecab/), originally a Japanese morphological analyzer and POS tagger developed by the Graduate School of Informatics in Kyoto University, was modified to MeCab-ko by the Eunjeon Project (http://eunjeon.blogspot.kr/) to adapt to the Korean language.

In order to use MeCab-ko within KoNLPy, follow the directions in optional-installations.

```
>>> # MeCab installation needed
>>> from konlpy.tag import Mecab
>>> mecab = Mecab()
>>> print(mecab.morphs(u'영등포구청역에 있는 맛집 좀 알려주세요.'))
['영등포구', '청역', '에', '있', '는', '맛집', '좀', '알려', '주', '세요', '.']
>>> print(mecab.nouns(u'우리나라에는 무릎 치료를 잘하는 정형외과가 없는가!'))
['우리', '나라', '무릎', '치료', '정형외과']
>>> print(mecab.pos(u'자연주의 쇼핑몰은 어떤 곳인가?'))
[('자연', 'NNG'), ('주', 'NNG'), ('의', 'JKG'), ('쇼핑몰', 'NNG'), ('은', 'JX'), (
↪'어떤', 'MM'), ('곳', 'NNG'), ('인가', 'VCP+EF'), ('?', 'SF')]
```

매개 변수 **dicpath** – The path of the MeCab-ko dictionary.

**morphs** (*phrase*)
Parse phrase to morphemes.

**nouns** (*phrase*)
Noun extractor.

**pos** (*phrase*, *flatten=True*, *join=False*)
POS tagger.

매개 변수

- **flatten** – If False, preserves eojeols.
- **join** – If True, returns joined sets of morph and tag.

### Okt Class

**경고:** `Twitter()` has changed to `Okt()` since v0.5.0.

**class** konlpy.tag._okt.**Okt**(*jvmpath=None*, *max_heap_size=1024*)

Wrapper for Open Korean Text (https://github.com/open-korean-text/open-korean-text).

Open Korean Text is an open source Korean tokenizer written in Scala, developed by Will Hohyon Ryu.

```
>>> from konlpy.tag import Okt
>>> okt = Okt()
>>> print(okt.morphs(u'단독입찰보다 복수입찰의 경우'))
['단독', '입찰', '보다', '복수', '입찰', '의', '경우']
>>> print(okt.nouns(u'유일하게 항공기 체계 종합개발 경험을 갖고 있는 KAI는'))
['항공기', '체계', '종합', '개발', '경험']
>>> print(okt.phrases(u'날카로운 분석과 신뢰감 있는 진행으로'))
['날카로운 분석', '날카로운 분석과 신뢰감', '날카로운 분석과 신뢰감 있는 진행', '분석', '신뢰
↪', '진행']
>>> print(okt.pos(u'이것도 되나욬ㅋㅋ'))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되나욬', 'Noun'), ('ㅋㅋ
↪', 'KoreanParticle')]
>>> print(okt.pos(u'이것도 되나욬ㅋㅋ', norm=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되나요', 'Verb'), ('ㅋㅋ
↪', 'KoreanParticle')]
>>> print(okt.pos(u'이것도 되나욬ㅋㅋ', norm=True, stem=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되다', 'Verb'), ('ㅋㅋ',
↪'KoreanParticle')]
```

매개 변수

- **jvmpath** – The path of the JVM passed to *init_jvm()* (page 39).
- **max_heap_size** – Maximum memory usage limitation (Megabyte) *init_jvm()* (page 39).

**morphs**(*phrase*, *norm=False*, *stem=False*)

Parse phrase to morphemes.

**nouns**(*phrase*)

Noun extractor.

**phrases**(*phrase*)

Phrase extractor.

**pos**(*phrase*, *norm=False*, *stem=False*, *join=False*)

POS tagger. In contrast to other classes in this subpackage, this POS tagger doesn't have a *flatten* option, but has *norm* and *stem* options. Check the parameter list below.

매개 변수

- **norm** – If True, normalize tokens.
- **stem** – If True, stem tokens.
- **join** – If True, returns joined sets of morph and tag.

konlpy.tag._okt.**Twitter**(*jvmpath=None*)
The `Twitter()` backend has changed to `Okt()` since KoNLPy v0.5.0. See #141 (https://github.com/konlpy/konlpy/issues/141) for details.

**더 보기:**

Korean POS tags comparison chart (https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSXI8/edit#gid=0)

Compare POS tags between several Korean analytic projects. (In Korean)

### corpus Package

**class** konlpy.corpus.**CorpusLoader**(*name=None*)
Loader for corpora. For a complete list of corpora available in KoNLPy, refer to *말뭉치* (page 30).

```
>>> from konlpy.corpus import kolaw
>>> fids = kolaw.fileids()
>>> fobj = kolaw.open(fids[0])
>>> print fobj.read(140)
대한민국헌법

유구한 역사와 전통에 빛나는 우리 대한국민은 3·1운동으로 건립된 대한민국임시정부의 법통과 불의에
항거한 4·19민주이념을 계승하고, 조국의 민주개혁과 평화적 통일의 사명에 입각하여 정의·인도와 동포
애로써 민족의 단결을 공고히 하고, 모든 사회적 폐습과 불의를 타파하며, 자율과 조화를 바 바
```

**abspath**(*filename=None*)
Absolute path of corpus file. If `filename` is *None*, returns absolute path of corpus.

**매개 변수** **filename** – Name of a particular file in the corpus.

**fileids**()
List of file IDs in the corpus.

**open**(*filename*)
Method to open a file in the corpus. Returns a file object.

**매개 변수** **filename** – Name of a particular file in the corpus.

## 6.1.2 data Module

konlpy.data.**find**(*resource_url*)
Find the path of a given resource URL by searching through directories in `konlpy.data.path`. If the given resource is not found, raise a `LookupError`, whose message gives a pointer to the installation instructions for `konlpy.download()`.

**매개 변수** **resource_url** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – The URL of the resource to search for. URLs are posix-style relative path names, such as `corpora/kolaw`. In particular, directory names should always be separated by the forward slash character (i.e., '/'), which will be automatically converted to a platform-appropriate path separator by KoNLPy.

konlpy.data.**load**(*resource_url*, *format='auto'*)
Load a given resource from the KoNLPy data package. If no format is specified, `load()` will attempt to determine a format based on the resource name's file extension. If that fails, `load()` will raise a `ValueError` exception.

**매개 변수**

- **resource_url** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – A URL specifying where the resource should be loaded from.
- **format** – Format type of resource.

konlpy.data.**path = ['/home/docs/konlpy_data', '/usr/share/konlpy_data', '/usr/local/shar**
A list of directories where the KoNLPy data package might reside. These directories will be checked in order when looking for a resource. Note that this allows users to substitute their own versions of resources.

**class** konlpy.data.**FileSystemPathPointer**(*path*)
Bases: *konlpy.data.PathPointer* (page 38), str (https://docs.python.org/3/library/stdtypes.html#str)

A path pointer that identifies a file by an absolute path.

**file_size**()

**open**(*encoding='utf-8'*)

**class** konlpy.data.**PathPointer**
Bases: object (https://docs.python.org/3/library/functions.html#object)

An abstract base class for path pointers. One subclass exists: 1. FileSystemPathPointer: Identifies a file by an absolute path.

**file_size**()

**open**(*encoding='utf-8'*)

### 6.1.3 downloader Module

**class** konlpy.downloader.**Downloader**(*download_dir=None*)
Bases: object (https://docs.python.org/3/library/functions.html#object)

A class used to access the KoNLPy data server, which can be used to download packages.

**INDEX_URL = 'http://konlpy.github.io/konlpy-data/index.json'**

**INSTALLED = 'installed'**

**NOT_INSTALLED = 'not installed'**

**PACKAGE_URL = 'http://konlpy.github.io/konlpy-data/packages/%s.%s'**

**SCRIPT_URL = 'http://konlpy.github.io/konlpy-data/packages/%s.sh'**

**STALE = 'corrupt or out of date'**

**download**(*id=None*, *download_dir=None*)
The KoNLPy data downloader. With this module you can download corpora, models and other data packages that can be used with KoNLPy.

Individual packages can be downloaded by passing a single argument, the package identifier for the package that should be downloaded:

```
>>> download('corpus/kobill')
[konlpy_data] Downloading package 'kobill'...
[konlpy_data]   Unzipping corpora/kobill.zip.
```

To download all packages, simply call download with the argument 'all':

```
>>> download('all')
[konlpy_data] Downloading package 'kobill'...
[konlpy_data]   Unzipping corpora/kobill.zip.
...
```

**status**(*info_or_id=None*, *download_dir=None*)

konlpy.downloader.**default_download_dir**()
Returns the directory to which packages will be downloaded by default. This value can be overriden using the constructor, or on a case-by-case basis using the download_dir argument when calling download().

On Windows, the default download directory is `PYTHONHOME/lib/konlpy`, where *PYTHONHOME* is the directory containing Python e.g., `C:\Python27`.

On all other platforms, the default directory is the first of the following which exists or which can be created with write permission: `/usr/share/konlpy_data`, `/usr/local/share/konlpy_data`, `/usr/lib/konlpy_data`, `/usr/local/lib/konlpy_data`, `~/konlpy_data`.

### 6.1.4 `jvm` Module

konlpy.jvm.**init_jvm**(*jvmpath=None*, *max_heap_size=1024*)
Initializes the Java virtual machine (JVM).

**매개 변수**

- **jvmpath** – The path of the JVM. If left empty, inferred by `jpype.getDefaultJVMPath()`.
- **max_heap_size** – Maximum memory usage limitation (Megabyte). Default is 1024 (1GB). If you set this value too small, you may got out of memory. We recommend that you set it 1024 ~ 2048 or more at least. However, if this value is too large, you may see inefficient memory usage.

### 6.1.5 `utils` Module

**class** konlpy.utils.**UnicodePrinter**(*indent=1*, *width=80*, *depth=None*, *stream=None*)
Bases: `pprint.PrettyPrinter` (https://docs.python.org/3/library/pprint.html#pprint.PrettyPrinter)

**format**(*object*, *context*, *maxlevels*, *level*)
Overrided method to enable Unicode pretty print.

konlpy.utils.**char2hex**(*c*)
Converts a unicode character to hex.

```
>>> char2hex(u'음')
'0xc74c'
```

konlpy.utils.**concordance**(*phrase*, *text*, *show=False*)
Find concordances of a phrase in a text.

The farmost left numbers are indices, that indicate the location of the phrase in the text (by means of tokens). The following string, is part of the text surrounding the phrase for the given index.

**매개 변수**

- **phrase** – Phrase to search in the document.
- **text** – Target document.
- **show** – If `True`, shows locations of the phrase on the console.

```
>>> from konlpy.corpus import kolaw
>>> from konlpy.tag import Mecab
>>> from konlpy import utils
>>> constitution = kolaw.open('constitution.txt').read()
>>> idx = utils.concordance(u'대한민국', constitution, show=True)
0       대한민국헌법 유구한 역사와
9       대한국민은 3·1운동으로 건립된 대한민국임시정부의 법통과 불의에
98      총강 제1조 ① 대한민국은 민주공화국이다. ②대한민국의
100     ① 대한민국은 민주공화국이다. ②대한민국의 주권은 국민에게
110     나온다. 제2조 ① 대한민국의 국민이 되는
126     의무를 진다. 제3조 대한민국의 영토는 한반도와
133     부속도서로 한다. 제4조 대한민국은 통일을 지향하며,
147     추진한다. 제5조 ① 대한민국은 국제평화의 유지에
```

(continues on next page)

(이전 페이지에서 계속)

```
787     군무원이 아닌 국민은 대한민국의 영역안에서는 중대한
1836    파견 또는 외국군대의 대한민국 영역안에서의 주류에
3620    경제 제119조 ① 대한민국의 경제질서는 개인과
>>> idx
[0, 9, 98, 100, 110, 126, 133, 147, 787, 1836, 3620]
```

konlpy.utils.**csvread**(*f*, *encoding=u'utf-8'*)
Reads a csv file.

**매개 변수** **f** – File object.

```
>>> from konlpy.utils import csvread
>>> with open('some.csv', 'r') as f:
        print csvread(f)
[[u'이 / NR', u'차 / NNB'], [u'나가 / VV', u'네 / EFN']]
```

konlpy.utils.**csvwrite**(*data*, *f*)
Writes a csv file.

**매개 변수** **data** – A list of list.

```
>>> from konlpy.utils import csvwrite
>>> d = [[u'이 / NR', u'차 / NNB'], [u'나가 / VV', u'네 / EFN']]
>>> with open('some.csv', 'w') as f:
        csvwrite(d, f)
```

konlpy.utils.**hex2char**(*h*)
Converts a hex character to unicode.

```
>>> print hex2char('c74c')
음
>>> print hex2char('0xc74c')
음
```

konlpy.utils.**load_txt**(*filename*, *encoding=u'utf-8'*)
Text file loader. To read a file, use ‘‘read_txt()‘‘instead.

konlpy.utils.**partition**(*list_*, *indices*)
Partitions a list to several parts using indices.

**매개 변수**

- **list** – The target list.
- **indices** – Indices to partition the target list.

konlpy.utils.**pprint**(*obj*, ***kwargs*)
Unicode pretty printer.

```
>>> import pprint, konlpy
>>> pprint.pprint([u"Print", u"유니코드", u"easily"])
[u'Print', u'유니코드', u'easily']
>>> konlpy.utils.pprint([u"Print", u"유니코드", u"easily"])
['Print', '유니코드', 'easily']
```

**매개 변수** **stream** – Option to stream to a particular destination. Can be either sys.stdout (default) or sys.stderr. See #179 for details.

konlpy.utils.**read_json**(*filename*, *encoding=u'utf-8'*)
JSON file reader.

konlpy.utils.**read_txt**(*filename*, *encoding=u'utf-8'*)
Text file reader.

konlpy.utils.**select**(*phrase*)
Replaces some ambiguous punctuation marks to simpler ones.

CHAPTER 7

# 인덱스와 표

- genindex
- modindex
- search
- changelog

# Python 모듈 목록

## k

# 색인

## R



## S



## T



## U