# Document Search Using Python

406.426B 데이터관리와 분석

2024.11.18

성민재
alswo5131@snu.ac.kr
서울대학교 산업공학과

Document Search Using Python

# BEFORE ROBOTS.TXT

# Crawling

- 데이터 주권(Data Sovereignty[1] )
  - 정보의 가치가 있는 데이터(Data)에 대해서 언제 어떻게 사용할 것인지에 대해 국가나 개인이 결정할 수 있는 권리
  - ML/AI와 같이 데이터 분석 기법이 발달함에 따라 데이터 수집에 대한 수요 증가
  - 데이터 저작권을 지키기 위한 방법 고도화 및 필요성 대두
    - Ex1) 동일 세션의 반복 접근 차단
    - Ex2) CSS 토큰 랜덤 생성으로 crawling을 위한 check point 지정 불가
- Robots.txt
  - Crawling 가능 여부를 문서화한 site
  - "URL/robots.txt"로 확인 가능

Robots.txt 예시

**User-agent: *** 사이트의 루트 페이지만 수집 허용으로 설정
**Disallow: /**
**Allow : /$**

[1] : https://en.wikipedia.org/wiki/Data_sovereignty
[2] : 두산백과. (2023). 데이터 주권. https://terms.naver.com/entry.naver?docId=6643886&cid=40942&categoryId=32840
[3] : 네이버 웹마스터 가이드. (2023). Robots.txt 설정하기 https://searchadvisor.naver.com/guide/seo-basic-robots

Document Search Using Python

# PART 1. TEXT PROCESSING: NLTK

# NLTK

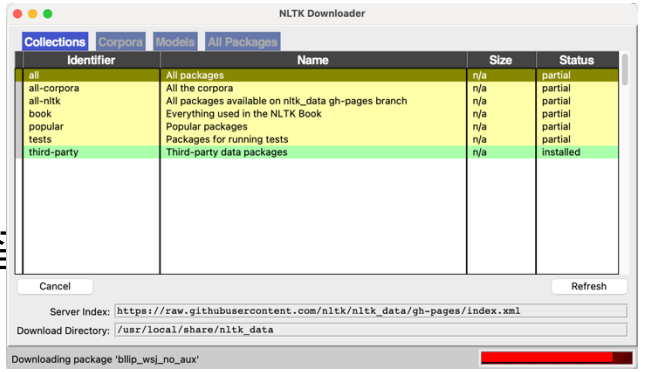
- NLTK(Natural Language Toolkit)
  - 교육용으로 개발된 자연어 처리 및 문서 분석용 python package
  - Tokenization, Stemming, Tagging 등 다양한 텍스트 처리 기능 제공
  - 다양한 말뭉치(corpus: 자연어 분서 작업을 위해 만든 샘플 문서 집합)도 제공
  - Documentation: (**https://www.nltk.org/api/nltk.html)**

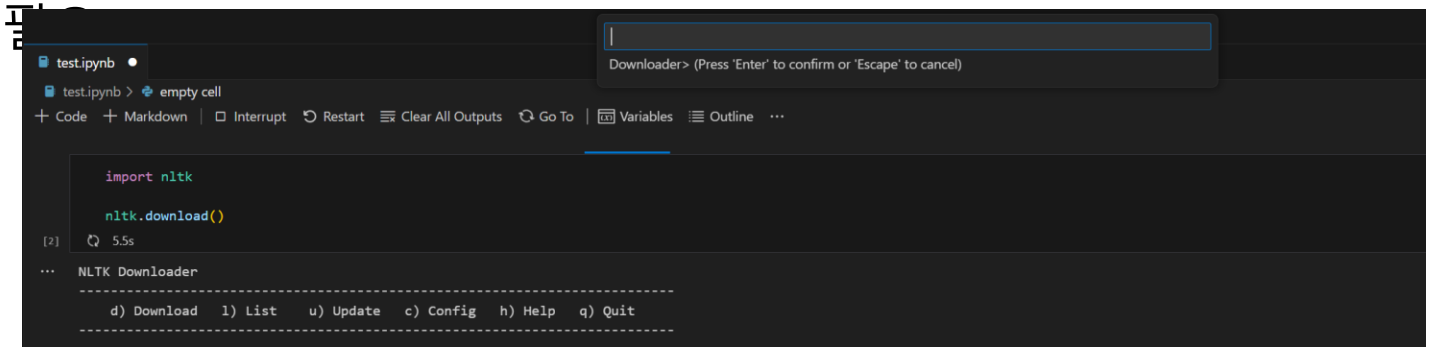
- 설치 및 사용
  - Anaconda 설치 시 기본 설치
  - 설치 안 되어 있을 경우 conda install nltk (https://www.nltk.org/install.html)

# NLTK 데이터 설치

- 별도로 데이터 설치 필요
  - cmd창에서 python 실행 혹은 jupyter notebook 혹은 Pycharm 등
  - import nltk
  - nltk.download()

- NLTK Downloader가 새로운 창으로 pop up
- Download directory
  - Window: C:₩nltk_data 혹은 anaconda3 설치 directory의 하위 nltk_data로 설
  - Mac: /usr/local/share/nltk_data
  - 다른 곳에 설치 시 별도로 환경변수 설정 필

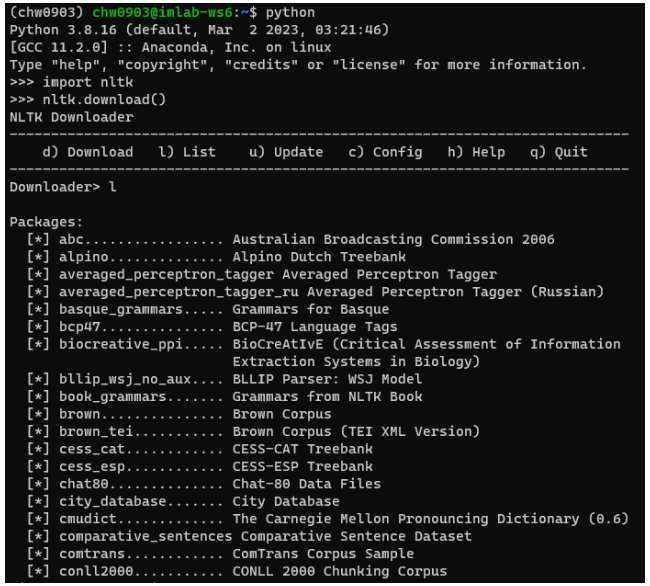


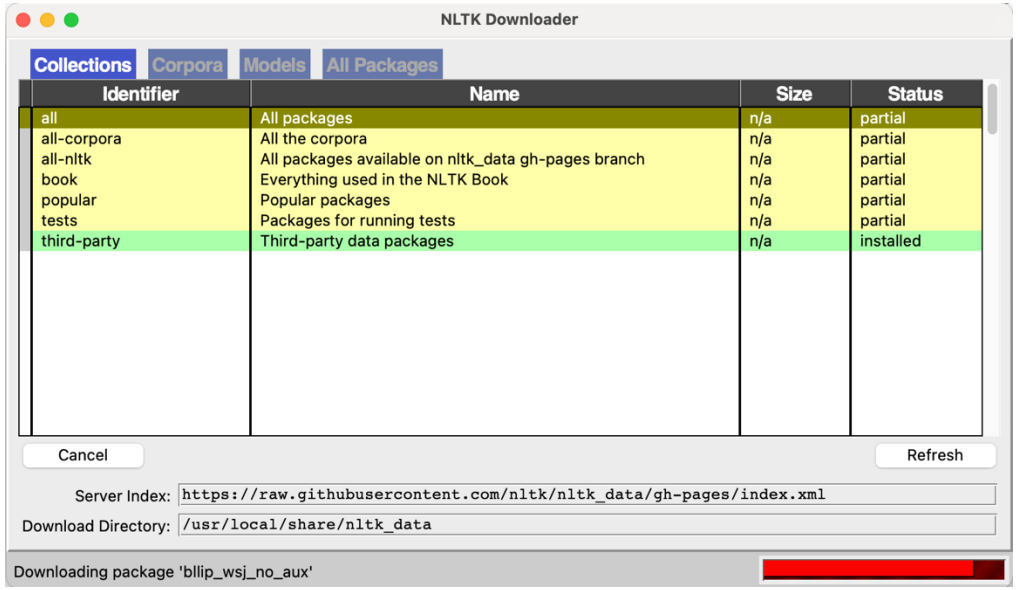



# NLTK 데이터 설치

- Collections에서 'all' 선택해서 전체 다운로드 가능
  - 용량이 커 다운로드에 오랜 시간 소요
  - 여러 시도를 위해 권장

- 실습 수업 및 프로젝트 기본 수행에 필요한 package – All packages에서 선택하여 다운로드 가능
  - averaged_perceptron_tagger
  - gutenberg
  - punkt
  - wordnet
  - tagsets



# Corpus

- gutenberg corpus

```
import nltk
nltk.corpus.gutenberg.fileids()
```

```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

```
emma_raw = nltk.corpus.gutenberg.raw("austen-emma.txt")
text = emma_raw[50:1301]
print(text)
```

```
Emma Woodhouse, handsome, clever, and rich, with a comfortable home
and happy disposition, seemed to unite some of the best blessings
of existence; and had lived nearly twenty-one years in the world
with very little to distress or vex her.

She was the youngest of the two daughters of a most affectionate,
indulgent father; and had, in consequence of her sister's marriage,
been mistress of his house from a very early period.  Her mother
had died too long ago for her to have more than an indistinct
remembrance of her caresses; and her place had been supplied
by an excellent woman as governess, who had fallen little short
of a mother in affection.

Sixteen years had Miss Taylor been in Mr. Woodhouse's family,
less as a governess than a friend, very fond of both daughters,
but particularly of Emma.  Between _them_ it was more the intimacy
of sisters.  Even before Miss Taylor had ceased to hold the nominal
office of governess, the mildness of her temper had hardly allowed
her to impose any restraint; and the shadow of authority being
now long passed away, they had been living together as friend and
friend very mutually attached, and Emma doing just what she liked;
```

# Tokenization

- Sentence tokenize
  - Text를 sentence 단위로 tokenize
  - from nltk.tokenize import sent_tokenize

```
from nltk.tokenize import sent_tokenize
sentence = sent_tokenize(text)[1]
print(sentence)
```

```
She was the youngest of the two daughters of a most affectionate,
indulgent father; and had, in consequence of her sister's marriage,
been mistress of his house from a very early period.
```

- Word tokenize
  - Text를 word 단위로 tokenize
  - from nltk.tokenize import word_tokenize

```
from nltk.tokenize import word_tokenize
print(word_tokenize(sentence))
```

```
['She', 'was', 'the', 'youngest', 'of', 'the', 'two', 'daughters', 'of', 'a', 'most', 'affectionate', ',', 'indulgent', 'f
ather', ';', 'and', 'had', ',', 'in', 'consequence', 'of', 'her', 'sister', "'s", 'marriage', ',', 'been', 'mistress', 'o
f', 'his', 'house', 'from', 'a', 'very', 'early', 'period', '.']
```

# Tokenization

- RegexpTokenizer
  - from nltk.tokenize import RegexpTokenizer
  - () 안에 정규 표현식을 입력해서 토큰화
  - 또는 gaps=True로 지정하여 () 안에 토큰을 나누기 위한 기준을 입력해서 토큰화할 수도 있음

```
from nltk.tokenize import RegexpTokenizer
retokenize = RegexpTokenizer("[\w]+")
retokenize.tokenize(sentence)
```

```
'her',
'sister',
's',
'marriage',
'been',
'mistress',
'of',
'his',
'house',
'from',
'a',
'very',
'early',
'period']
```

```
retokenize = RegexpTokenizer("[\s]+", gaps=True)
retokenize.tokenize(sentence)
```

```
'her',
"sister's",
'marriage,',
'been',
'mistress',
'of',
'his',
'house',
'from',
'a',
'very',
'early',
'period.']
```

# Stemming

- PorterStemmer, SnowballStemmer, LancasterStemmer 등 다양한 stemmer 제공
  - 단어들의 stem 추출
  - from nltk.stem import PorterStemmer

```
from nltk.stem import PorterStemmer

words = ['caresses', 'flies', 'dies', 'mules', 'denied', 'died', 'agreed', 'owned',
         'humbled', 'sized', 'meeting', 'stating', 'siezing', 'itemization',
         'sensational', 'traditional', 'reference', 'colonizer', 'plotted']
st = PorterStemmer()

for w in words:
    print("%s -> %s" % (w, st.stem(w)))
```

```
caresses → caress
flies → fli
dies → die
mules → mule
denied → deni
died → die
agreed → agre
owned → own
humbled → humbl
sized → size
```

# Stemming

- from nltk.stem import LancasterStemmer

```
from nltk.stem import LancasterStemmer

words = ['caresses', 'flies', 'dies', 'mules', 'denied', 'died', 'agreed', 'owned',
         'humbled', 'sized', 'meeting', 'stating', 'siezing', 'itemization',
         'sensational', 'traditional', 'reference', 'colonizer', 'plotted']
st = LancasterStemmer()

for w in words:
    print("%s -> %s" % (w, st.stem(w)))
```

```
caresses → caress
flies → fli
dies → die
mules → mul
denied → deny
died → died
agreed → agree
owned → own
humbled → humbl
sized → siz
meeting → meet
stating → stat
siezing → siez
itemization → item
```

# Lemmatizing

- WordNetLemmatizer
  - stemming은 단순히 어미를 제거, 단어의 원형을 정확히 찾아주지는 않음
  - lemmatizing은 같은 의미를 가지는 여러 단어를 사전형으로 통일
  - 품사(pos)를 지정하는 경우 좀 더 정확한 원형을 찾을 수 있음

```
from nltk.stem import PorterStemmer, LancasterStemmer

st1 = PorterStemmer()
st2 =  LancasterStemmer()

words = ["fly", "flies", "flying", "flew", "flown"]

print("Porter Stemmer   :", [st1.stem(w) for w in words])
print("Lancaster Stemmer:", [st2.stem(w) for w in words])
```

```
Porter Stemmer   : ['fli', 'fli', 'fli', 'flew', 'flown']
Lancaster Stemmer: ['fly', 'fli', 'fly', 'flew', 'flown']
```

```
from nltk.stem import WordNetLemmatizer

lm = WordNetLemmatizer()

print("Word Net Lemmatizer:", [lm.lemmatize(w, pos="v") for w in words])
```

```
Word Net Lemmatizer: ['fly', 'fly', 'fly', 'fly', 'fly']
```

# POS Tagging

- Part-Of-Speech Tagging
  - 단어들의 품사 tagging
  - nltk.help.upenn_tagset()을 이용해 각 태그의 자세한 설명 확
  - from nltk.tag import pos_tag

```
nltk.help.upenn_tagset("VB")
```

```
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
    bank begin behold believe bend benefit bevel beware bless boil bomb
    boost brace break bring broil brush build ...
```

```
from nltk.tag import pos_tag
tagged_list = pos_tag(word_tokenize(sentence))
tagged_list
```

```
[('She', 'PRP'),
 ('was', 'VBD'),
 ('the', 'DT'),
 ('youngest', 'JJS'),
 ('of', 'IN'),
 ('the', 'DT'),
 ('two', 'CD'),
 ('daughters', 'NNS'),
 ('of', 'IN'),
 ('a', 'DT'),
 ('most', 'RBS'),
 ('affectionate', 'JJ'),
 (',', ','),
 ('indulgent', 'JJ'),
 ('father', 'NN'),
 (';', ':'),
 ('and', 'CC'),
 ('had', 'VBD'),
 (',', ','),
 ('in', 'IN'),
 ('consequence', 'NN'),
```

# POS Tagging

- Part-Of-Speech Tagging
  - pos tagging 정보를 사용하여, 특정 품사의 토큰만 선택 가능
  - untag 명령을 사용하면 태그 튜플 제거 가능

```
nouns_list = [t[0] for t in tagged_list if t[1] == "NN"]
nouns_list
```

```
['father', 'consequence', 'sister', 'marriage', 'mistress', 'house', 'period']
```

```
from nltk.tag import untag
untag(tagged_list)
```

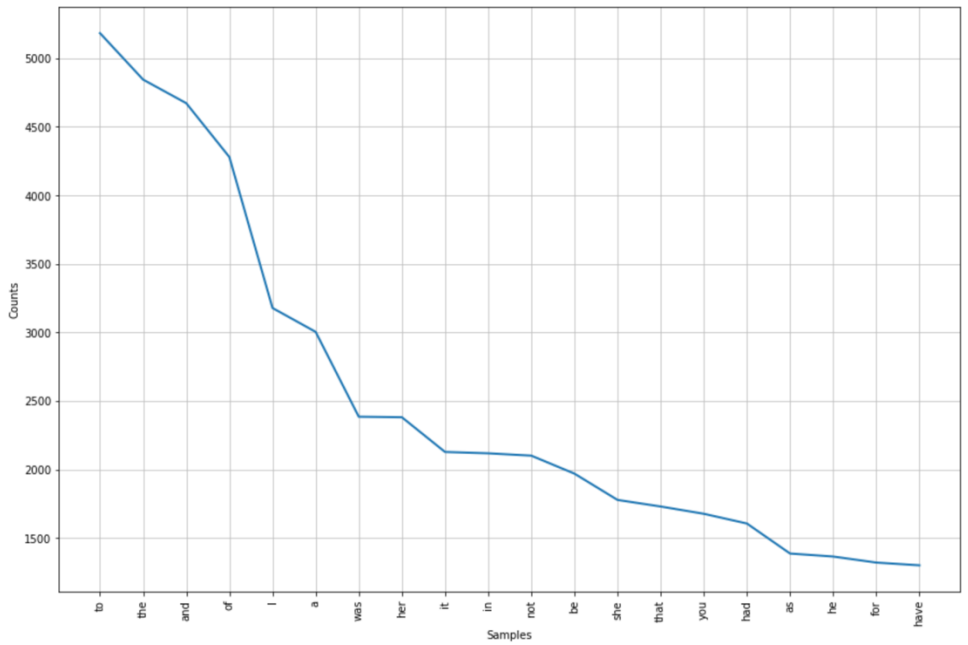
```
['She',
 'was',
 'the',
 'youngest',
 'of',
 'the',
 'two',
 'daughters',
 'of',
 'a',
 'most',
 'affectionate',
 ',',
 'indulgent',
 'father',
 ';',
 'and',
 'had',
```

# Text class

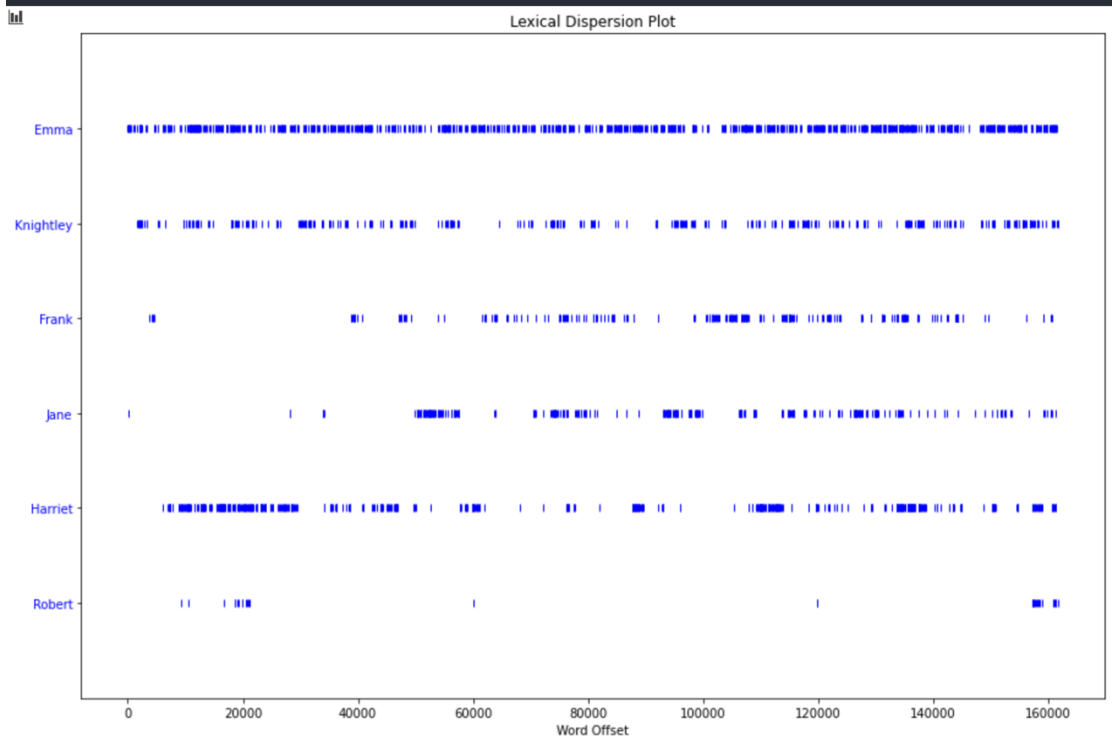
- Text 클래스는 문서 분석에 유용한 여러가지 함수 제공
  - Text 클래스의 객체 생성 시 텍스트를 입력하여
  - plot 함수를 사용하면 각 단어의 사용 빈도를 그
  - dispersion_plot 함수는 텍스트에서 단어가 사용된 위치를 시각화

```
from nltk import Text

text = Text(retokenize.tokenize(emma_raw))
```

```
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (15, 10)
text.plot(20)
plt.show()
```




```
text.dispersion_plot(["Emma", "Knightley", "Frank", "Jane", "Harriet", "Robert"])
```



# FreqDist

- FreqDist 클래스는 문서에 사용된 단어(토큰)의 사용 빈도 정보를 제공
  - 앞서 생성한 Text 클래스의 vocab() 함수를 통해 추출
  - 또는 새로운 단어들의 리스트를 넣어서 직접 생성
  - FreqDist 클래스는 단어를 키(Key), 출현 빈도를 값(value)로 가지는 딕셔너리 자료형과 유사
  - most_common() 함수를 통해 가장 출현 횟수가 높은 단어 추출

```
fd = text.vocab()
type(fd)
```

```
nltk.probability.FreqDist
```

```
from nltk import FreqDist

stopwords = ["Mr.", "Mrs.", "Miss", "Mr", "Mrs", "Dear"]
emma_tokens = pos_tag(retokenize.tokenize(emma_raw))
names_list = [t[0] for t in emma_tokens if t[1] == "NNP" and t[0] not in stopwords]
fd_names = FreqDist(names_list)
print(names_list)
```

```
'Robert', 'Martin', 'Emma', 'No', 'Knightley', 'Robert', 'Martin', 'Emma', 'Harriet', 'Smith', 'Robert', 'Martin', 'Knig
htley', 'God', 'Well', 'How', 'John', 'John', 'Astley', 'Astley', 'Henry', 'John', 'Smith', 'My', 'Robert', 'Harriet',
'Harriet', 'Robert', 'Martin', 'Astley', 'John', 'Knightley', 'John', 'Smith', 'Henry', 'Smith', 'Emma', 'Emma', 'Emma',
'William', 'Larkins', 'Robert', 'Martin', 'Harriet', 'Knightley', 'Emma', 'Knightley', '_accepted_', 'Did', 'Harriet',
```

```
fd_names.N(), fd_names["Emma"], fd_names.freq("Emma")
```

```
(7863, 830, 0.10555767518758744)
```

```
fd_names.most_common(5)
```

```
[('Emma', 830),
 ('Harriet', 491),
 ('Weston', 439),
 ('Knightley', 389),
 ('Elton', 385)]
```

# WordCloud

- 단어의 사용 빈도 수에 따라 워드클라우드 시각화
  - pip install wordcloud / conda install -c conda-forge wordcloud

```
from wordcloud import WordCloud

wc = WordCloud(width=1000, height=600, background_color="white", random_state=0)
plt.imshow(wc.generate_from_frequencies(fd_names))
plt.axis("off")
plt.show()
```

Document Search Using Python

# PART 2. DOCUMENT SEARCH: WHOOSH

# Whoosh

- Python으로 구현된 검색 엔진 library
  - 순수 python으로 구현
  - 빠른 색인 및 검색 기능 제공
  - Documentation: https://whoosh.readthedocs.io/en/latest/index.html

- 설치 및 사용
  - cmd창에 다음 명령어 입력
  - conda install whoosh or pip install whoosh

# Index 생성

- Schema 생성
  - Schema: 정보를 구조화하고 조직화하는데 사용되는 구조적 틀
  - 정보 구조화, 검색 효율성 향상, 데이터 일관성 유지 가능
  - Field가 가질 수 있는 field types
    - **whoosh.fields.TEXT**
    - **whoosh.fields.NUMERIC**
    - whoosh.fields.KEYWORD
    - whoosh.fields.ID
    - whoosh.fields.STORED
    - whoosh.fields.DATETIME
    - whoosh.fields.BOOLEAN

- Index object 생성
  - 생성한 schema의 형식을 갖는 index object 생성

# Index 생성

- Index 생성

```
schema = Schema(docno=NUMERIC(stored=True), contents=TEXT)
index_dir = "index"

if not os.path.exists(index_dir):
    os.makedirs(index_dir)

ix = index.create_in(index_dir, schema)
```

- 생성한 index 불러오기

```
ix = index.open_dir(index_dir)
```

- schema에 field 추가 및 제거

```
writer = ix.writer()
writer.add_field("fieldname", TEXT(stored=True))
writer.remove_field("fieldname")
writer.commit()
```

# Index 생성

- Index object에 add document
  - writer = ix.writer()
  - 각 document에 대해 ID와 내용이 각각 docno, contents filed에 들어가도록 add_document
  - 마지막에 writer.commit()를 통해 index object 저장
  - Index 저장 시에 contents를 가공하여 index를 작성할 수도 있음

```
writer = ix.writer()

writer.add_document(docno=1, contents="apple banana cherry")
writer.add_document(docno=2, contents="banana cherry pear")
writer.add_document(docno=3, contents="pear strawberry melon")
writer.add_document(docno=4, contents="apple melon banana tomato lemon")
writer.add_document(docno=5, contents="apple apple kiwi")

writer.commit()
```

# Searching

- Index object의 searcher() 사용
  - Scoring 함수 선택 가능 ( default는 BM25F )
- QueryParser()
  - Text query를 whoosh query object로 변형
  - Schema와 검색 field 명시
  - Default로는 query에 모든 term에 대해 AND 연산 수행, OR로 바꿔주기 위해서는 whoosh.qparser의 OrGroup 사용
- Results object

```
with ix.searcher(weighting=scoring.BM25F()) as searcher:
    print("BM25F")
    parser = QueryParser("contents", schema=ix.schema, group=OrGroup)
    query = parser.parse("apple banana")
    # query = parser.parse("apple^2 banana^0.5")
    results = searcher.search(query)
    for result in results:
        print(result.fields()['docno'], result.score)
```

```
BM25F
5 3.4787504805438365
1 3.212469720446029
4 2.564213498831359
2 0.6424939440892058
```

# Query handling

```
writer.add_document(docno=1, contents="apple banana cherry")
writer.add_document(docno=2, contents="banana cherry pear")
writer.add_document(docno=3, contents="pear strawberry melon")
writer.add_document(docno=4, contents="apple melon banana tomato lemon")
writer.add_document(docno=5, contents="apple apple kiwi")
```

- OrGroup
  - OrGroup.factory($\alpha$) $0 < \alpha < 1$
    - 단순한 OR($\alpha = 0$)는 질의어 내 서로 다른 단어에 대한 구분이 없음
    - 예) query "apple OR banana"에 대해, 문서 "apple banana"와 "apple apple"은 같은 점수를 받지만 두 단어를 모두 포함하는 문서가 더 선호되는 상황이 일반적
    - Scaling factor $\alpha$ $(0 < \alpha < 1)$

Scaling factor X

```
Custom Scoring tf
1 2.0
4 2.0
5 2.0
2 1.0
```

Scaling factor = 0.9

```
Custom Scoring tf
1 1.4132231404958677
4 1.4132231404958677
5 1.0
2 0.5
```

- OR 외에도 AND, ANDNOT, ANDMAYBE, NOT 등 다양한 syntax 제공

# Query handling

```
writer.add_document(docno=1, contents="apple banana cherry")
writer.add_document(docno=2, contents="banana cherry pear")
writer.add_document(docno=3, contents="pear strawberry melon")
writer.add_document(docno=4, contents="apple melon banana tomato lemon")
writer.add_document(docno=5, contents="apple apple kiwi")
```

- Weighting
  - 중요한, 혹은 덜 중요한 단어에 가중치 부여 가능
  - 예) apple은 2배, banana는 ½배 가중치 부여하고 싶다면
    "apple^2 banana^0.5"
- 그 외의 Query expansion

```
with ix.searcher(weighting=scoring.ScoringFunction()) as searcher:
    print("\nCustom Scoring tf")
    parser = QueryParser("contents", schema=ix.schema, group=OrGroup)
    # query = parser.parse("apple banana")
    query = parser.parse("apple^2 banana^0.5")
    results = searcher.search(query)
    for result in results:
        print(result.fields()['docno'], result.score)
```

```
Custom Scoring tf
5 4.0
1 2.5
4 2.5
2 0.5
```

# CustomScoring

- 첨부한 CustomScoring.py 에서 intappscorer 함수 변경
- 그 후 weighting=scoring.ScoringFunction() 입력

```
with ix.searcher(weighting=scoring.ScoringFunction()) as searcher:
```

- 이용 가능한 기본 정보들
  - tf: term frequency in the current document
  - idf: inverse document frequency
  - cf: term frequency in the collection
  - dc: doc count
  - fl: field length in the current document
  - avgfl: average field length across documents
  - param: free parameter

```
def intappscorer(tf, idf, cf, qf, dc, fl, avgfl, param):
    # tf - term frequency in the current document
    # idf - inverse document frequency
    # cf - term frequency in the collection
    # qf - term frequency in the query
    # dc - doc count
    # fl - field length in the current document
    # avgfl - average field length across documents in collection
    # param - free parameter
    # TODO - Define your own scoring function
    # below is just example
    return tf*idf
```

```
Custom Scoring tf*idf
5 4.892574205256839
1 3.0578588782855243
4 3.0578588782855243
2 0.6115717756571049
```

End of the Document