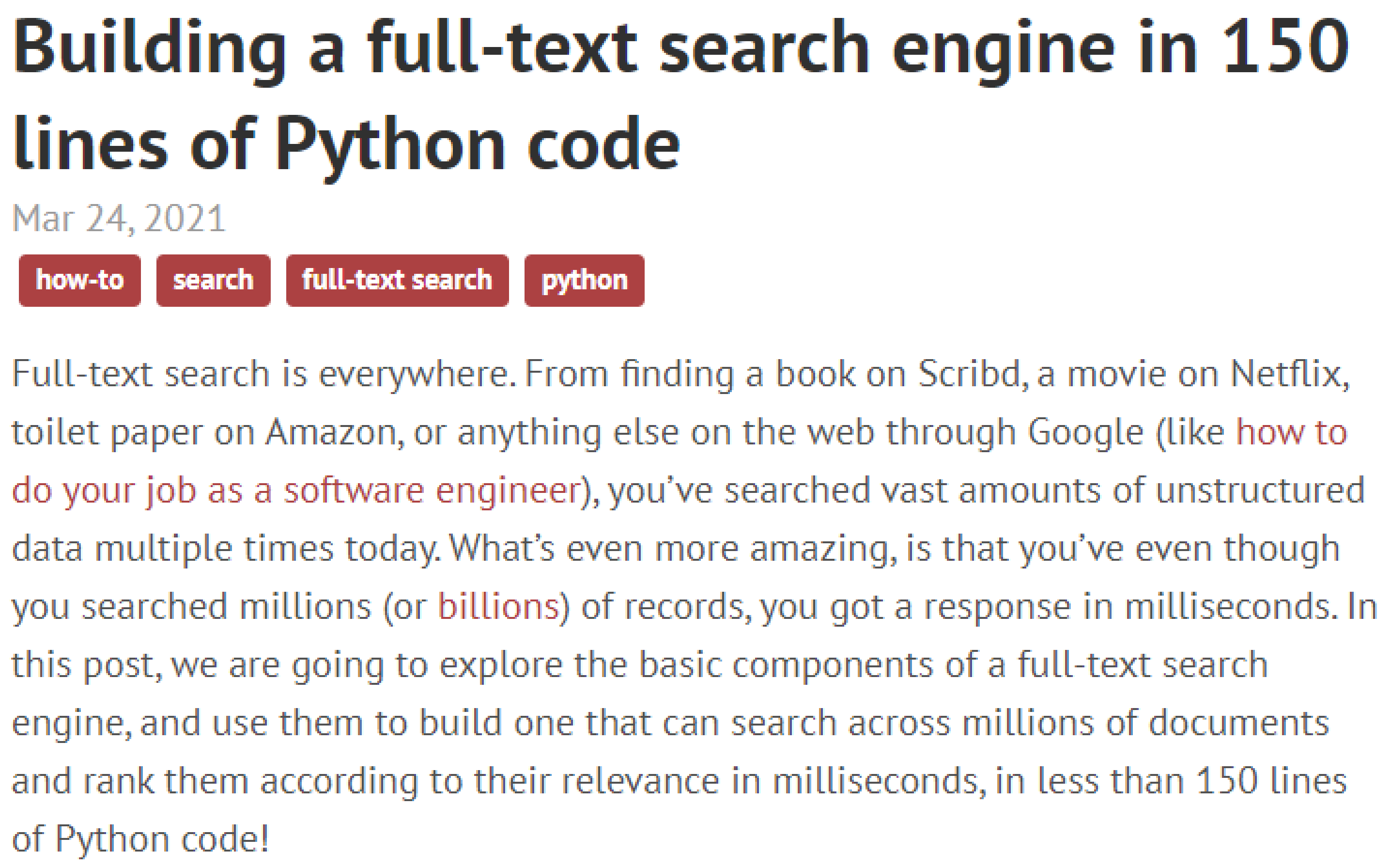
FULL TEXT SEARCH

박경규

Introduction

https://bart.degoe.de/building-a-full-text-search-engine-150-lines-of-code/



Building a full-text search engine in 150 lines of Python code

Mar 24, 2021

how-to | search | full-text search | python

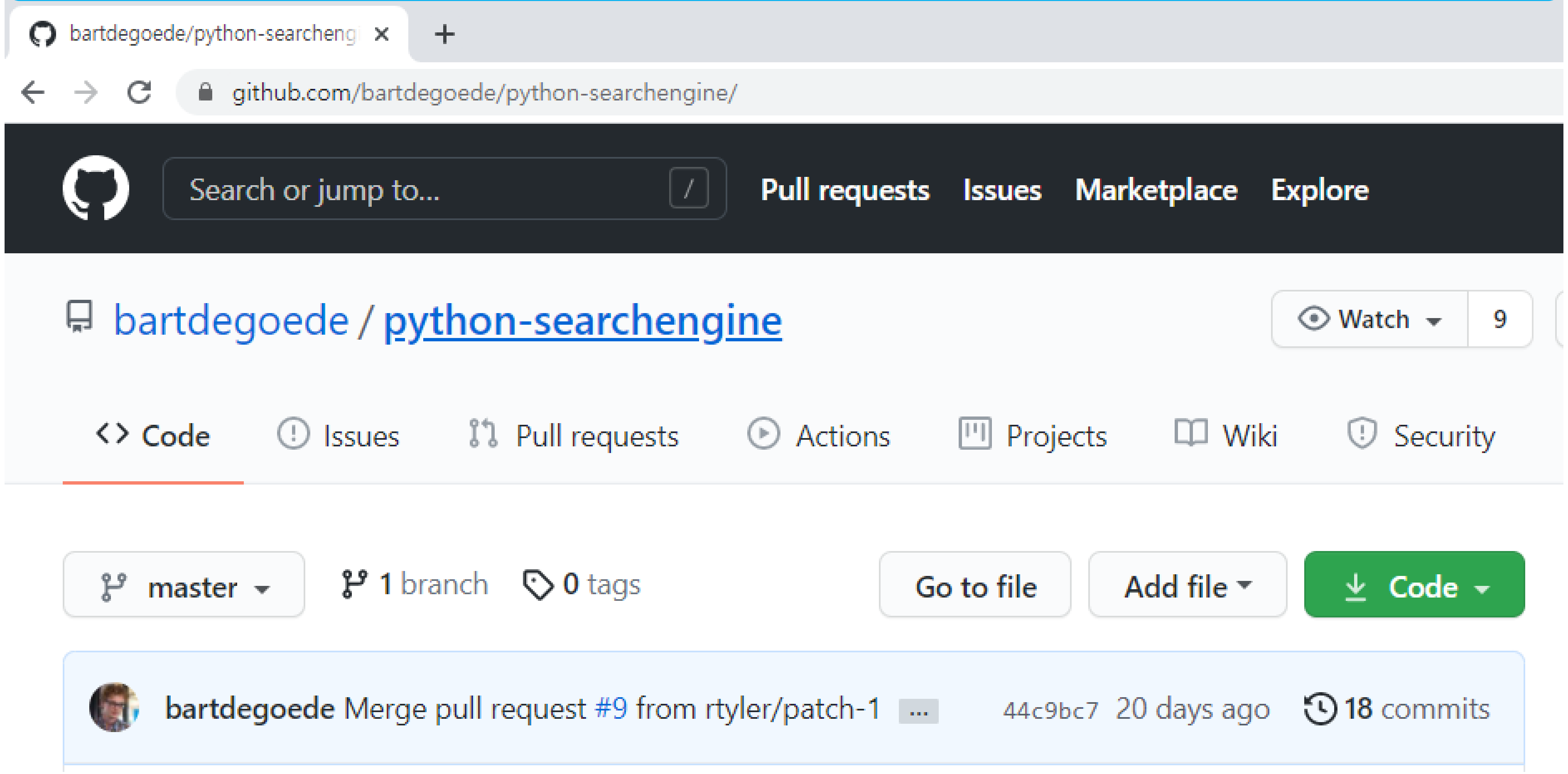
Full-text search is everywhere. From finding a book on Scribd, a movie on Netflix, toilet paper on Amazon, or anything else on the web through Google (like how to do your job as a software engineer), you've searched vast amounts of unstructured data multiple times today. What's even more amazing, is that you've even though you searched millions (or billions) of records, you got a response in milliseconds. In this post, we are going to explore the basic components of a full-text search engine, and use them to build one that can search across millions of documents and rank them according to their relevance in milliseconds, in less than 150 lines of Python code!

https://bart.degoe.de/building-a-full-text-search-engine-150-lines-of-code/

Data

■ 소스 다운로드

https://github.com/bartdegoede/python-searchengine/



bartdegoede/python-searcheng ×

github.com/bartdegoede/python-searchengine/

bartdegoede / python-searchengine

bartdegoede Merge pull request #9 from rtyler/patch-1 … 44c9bc7 20 days ago 18 commits

https://bart.degoe.de/building-a-full-text-search-engine-150-lines-of-code/

Data

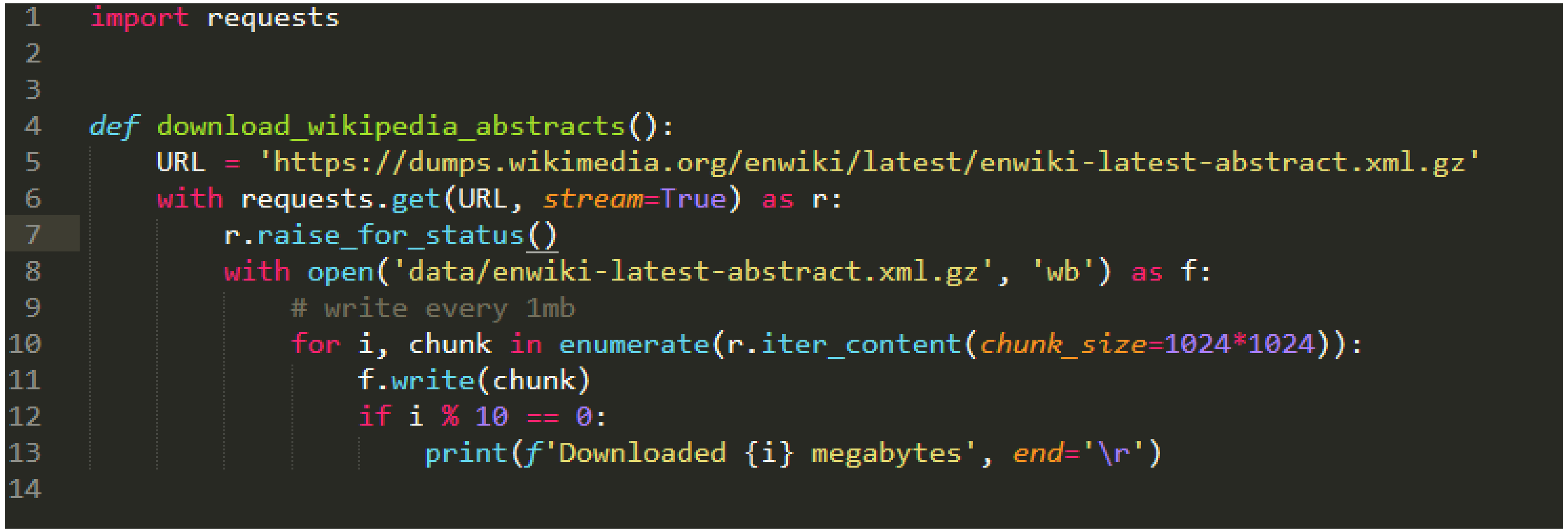
■ 파이썬 패키지 설치

pip install -r requirements.txt

■ 프로그램 실행

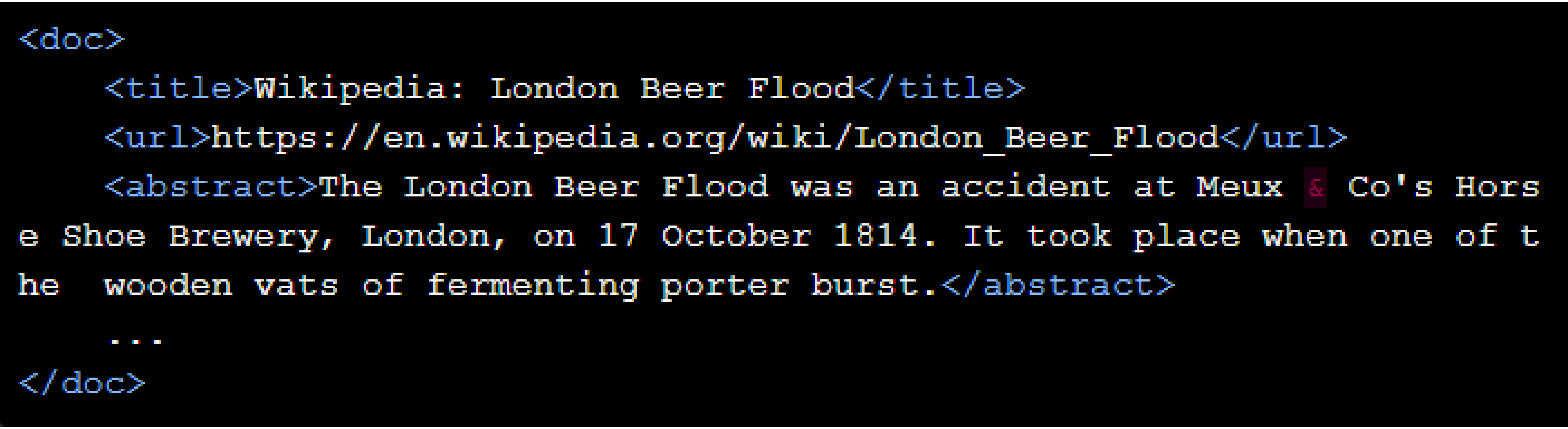
python run.py

download.py



```python
import requests


def download_wikipedia_abstracts():
    URL = 'https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-abstract.xml.gz'
    with requests.get(URL, stream=True) as r:
        r.raise_for_status()
        with open('data/enwiki-latest-abstract.xml.gz', 'wb') as f:
            # write every 1mb
            for i, chunk in enumerate(r.iter_content(chunk_size=1024*1024)):
                f.write(chunk)
                if i % 10 == 0:
                    print(f'Downloaded {i} megabytes', end='\r')

```

Data preparation

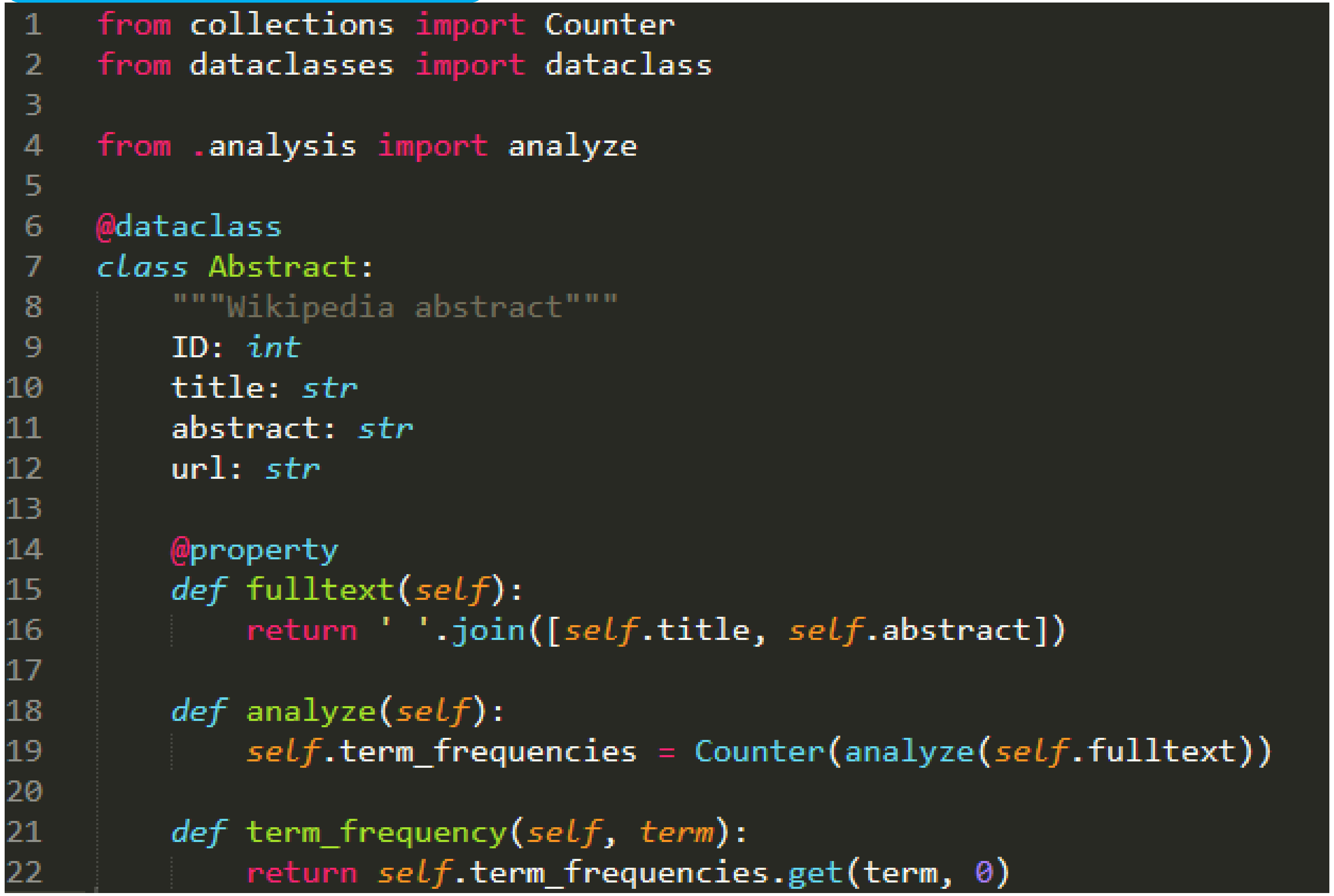


```
<doc>
    <title>Wikipedia: London Beer Flood</title>
    <url>https://en.wikipedia.org/wiki/London_Beer_Flood</url>
    <abstract>The London Beer Flood was an accident at Meux & Co's Hors
e Shoe Brewery, London, on 17 October 1814. It took place when one of t
he  wooden vats of fermenting porter burst.</abstract>
    ...
</doc>
```

https://bart.degoe.de/building-a-full-text-search-engine-150-lines-of-code/

Data preparation

documents.py



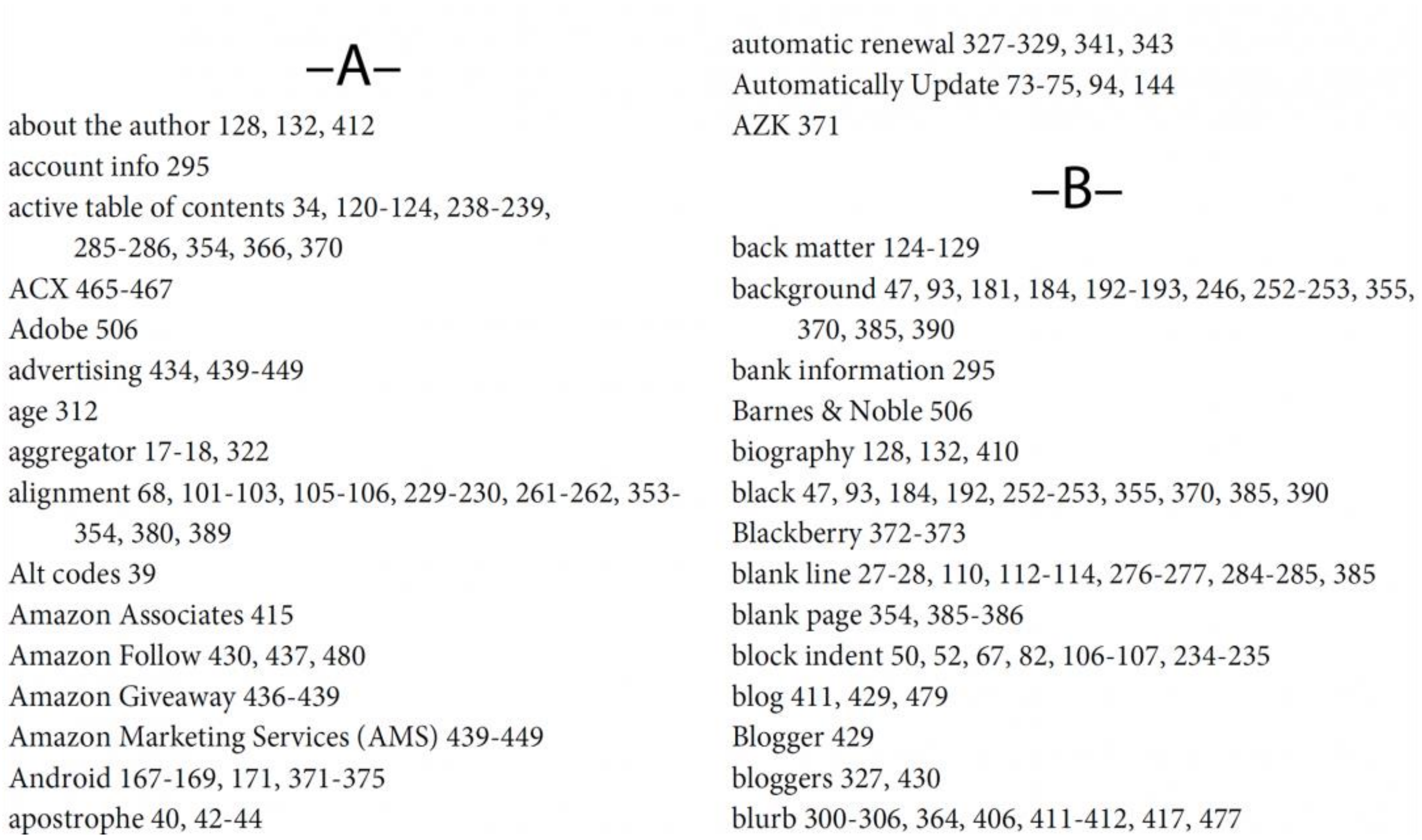
```python
from collections import Counter
from dataclasses import dataclass

from .analysis import analyze

@dataclass
class Abstract:
    """Wikipedia abstract"""
    ID: int
    title: str
    abstract: str
    url: str

    @property
    def fulltext(self):
        return ' '.join([self.title, self.abstract])

    def analyze(self):
        self.term_frequencies = Counter(analyze(self.fulltext))

    def term_frequency(self, term):
        return self.term_frequencies.get(term, 0)
```

Data preparation

load.py

```python
import gzip
from lxml import etree
import time

from search.documents import Abstract

def load_documents():
    start = time.time()
    with gzip.open('data/enwiki-latest-abstract.xml.gz', 'rb') as f:
        doc_id = 0
        for _, element in etree.iterparse(f, events=('end',), tag='doc'):
            title = element.findtext('./title')
            url = element.findtext('./url')
            abstract = element.findtext('./abstract')

            yield Abstract(ID=doc_id, title=title, url=url, abstract=abstract)

            doc_id += 1
            element.clear()
    end = time.time()
    print(f'Parsing XML took {end - start} seconds')
```

Indexing



–A–

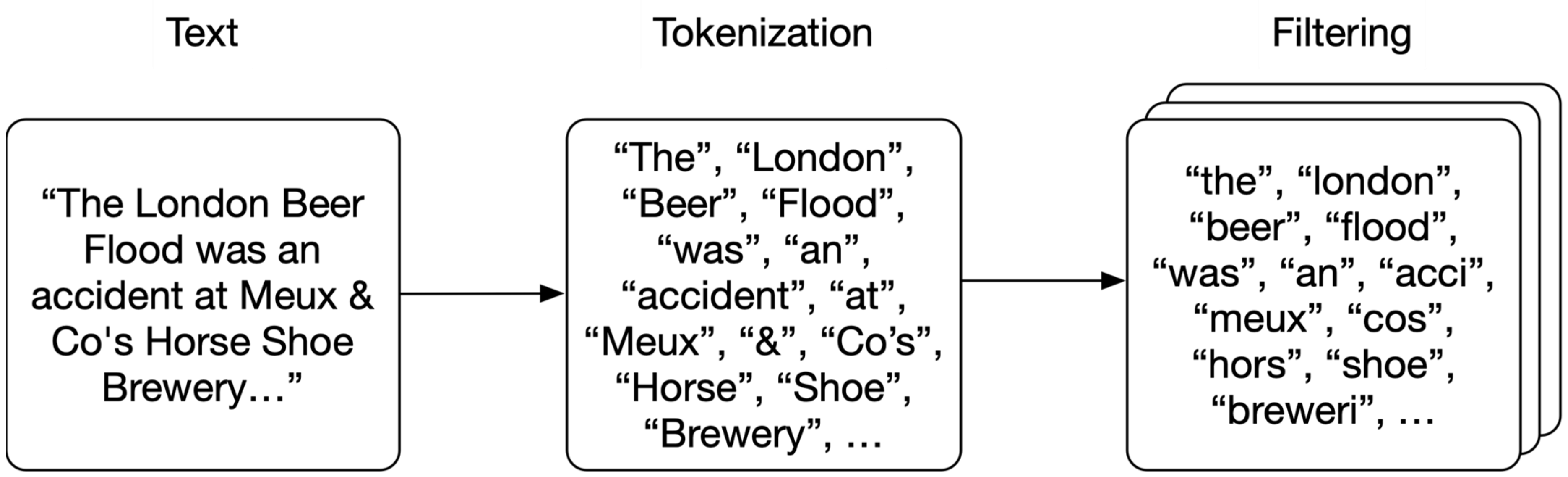
about the author 128, 132, 412
account info 295
active table of contents 34, 120-124, 238-239, 285-286, 354, 366, 370
ACX 465-467
Adobe 506
advertising 434, 439-449
age 312
aggregator 17-18, 322
alignment 68, 101-103, 105-106, 229-230, 261-262, 353-354, 380, 389
Alt codes 39
Amazon Associates 415
Amazon Follow 430, 437, 480
Amazon Giveaway 436-439
Amazon Marketing Services (AMS) 439-449
Android 167-169, 171, 371-375
apostrophe 40, 42-44
automatic renewal 327-329, 341, 343
Automatically Update 73-75, 94, 144
AZK 371

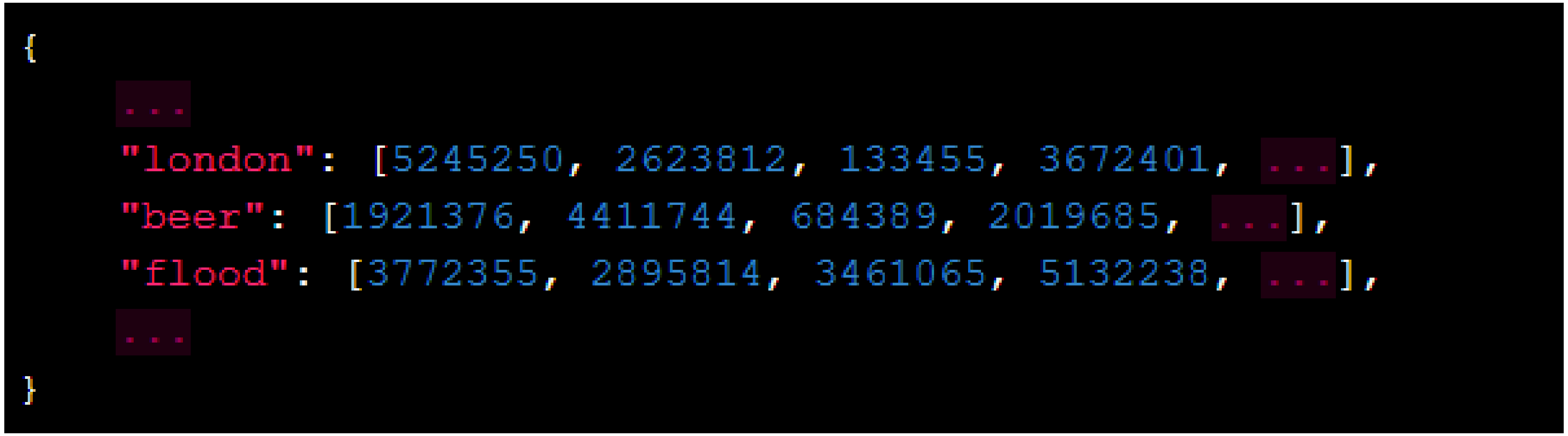
–B–

back matter 124-129
background 47, 93, 181, 184, 192-193, 246, 252-253, 355, 370, 385, 390
bank information 295
Barnes & Noble 506
biography 128, 132, 410
black 47, 93, 184, 192, 252-253, 355, 370, 385, 390
Blackberry 372-373
blank line 27-28, 110, 112-114, 276-277, 284-285, 385
blank page 354, 385-386
block indent 50, 52, 67, 82, 106-107, 234-235
blog 411, 429, 479
Blogger 429
bloggers 327, 430
blurb 300-306, 364, 406, 411-412, 417, 477

https://bart.degoe.de/building-a-full-text-search-engine-150-lines-of-code/

Indexing


Text
Tokenization
Filtering
“The London Beer Flood was an accident at Meux & Co's Horse Shoe Brewery…”
“The”, “London”, “Beer”, “Flood”, “was”, “an”, “accident”, “at”, “Meux”, “&”, “Co’s”, “Horse”, “Shoe”, “Brewery”, …
“the”, “london”, “beer”, “flood”, “was”, “an”, “acci”, “meux”, “cos”, “hors”, “shoe”, “breweri”, …

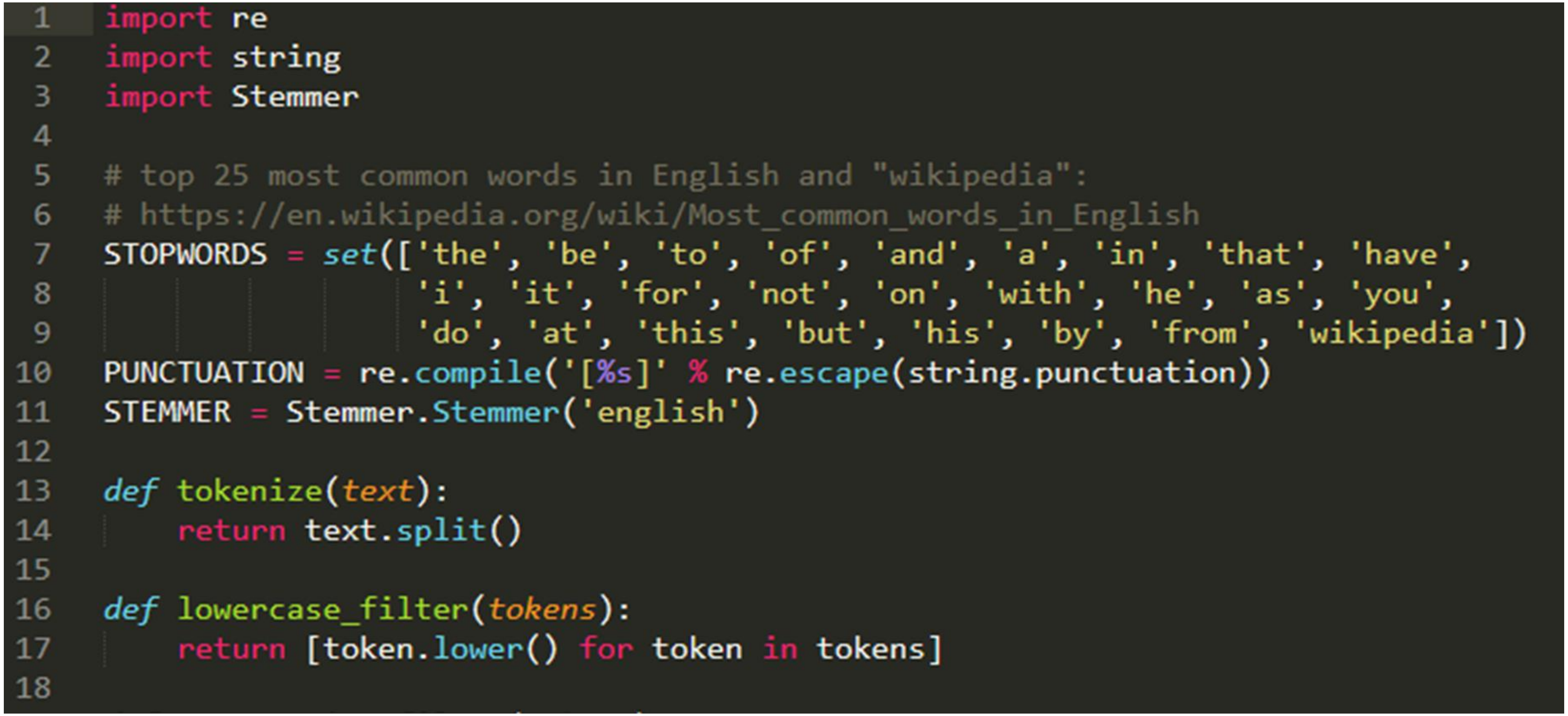


```
{
    ...
    "london": [5245250, 2623812, 133455, 3672401, ...],
    "beer": [1921376, 4411744, 684389, 2019685, ...],
    "flood": [3772355, 2895814, 3461065, 5132238, ...],
    ...
}
```

https://bart.degoe.de/building-a-full-text-search-engine-150-lines-of-code/

Indexing

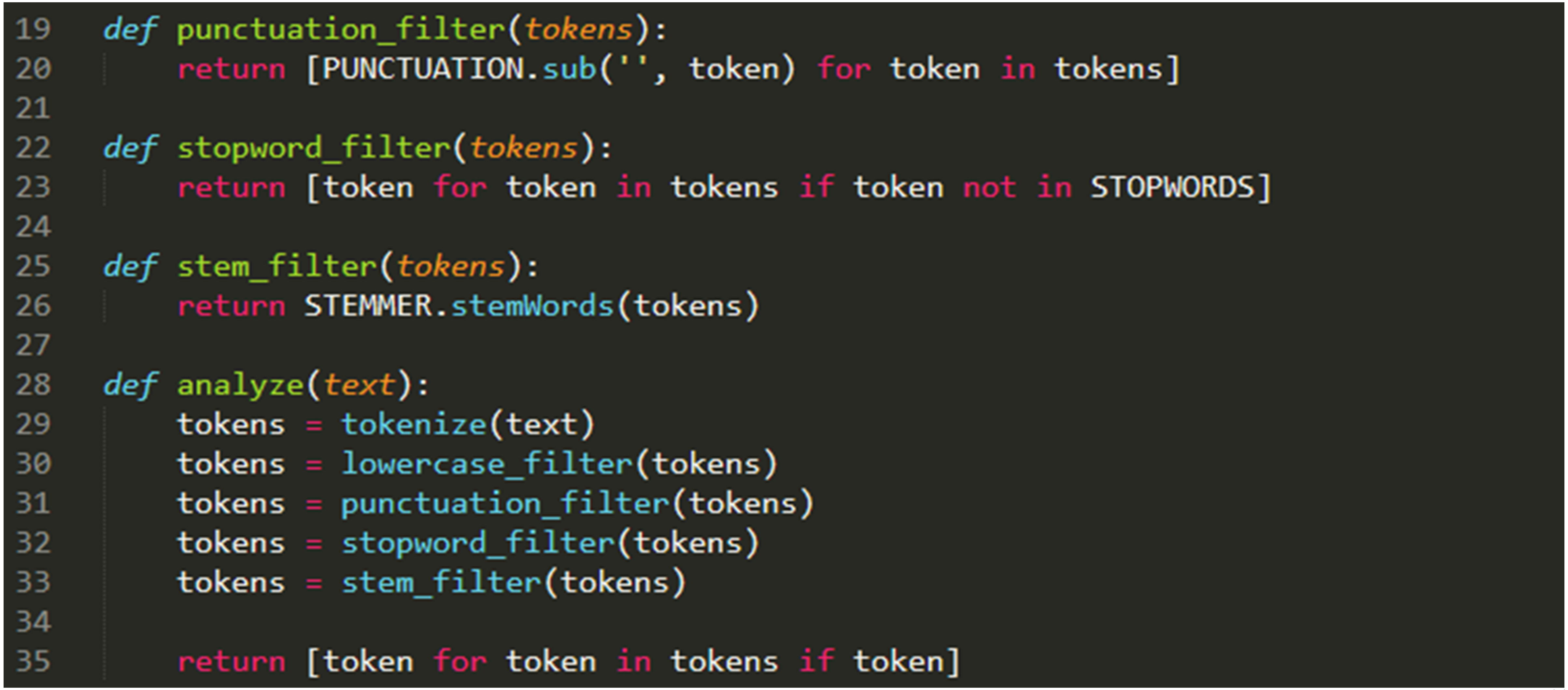
analysis.py



```python
import re
import string
import Stemmer

# top 25 most common words in English and "wikipedia":
# https://en.wikipedia.org/wiki/Most_common_words_in_English
STOPWORDS = set(['the', 'be', 'to', 'of', 'and', 'a', 'in', 'that', 'have',
                 'i', 'it', 'for', 'not', 'on', 'with', 'he', 'as', 'you',
                 'do', 'at', 'this', 'but', 'his', 'by', 'from', 'wikipedia'])
PUNCTUATION = re.compile('[%s]' % re.escape(string.punctuation))
STEMMER = Stemmer.Stemmer('english')

def tokenize(text):
    return text.split()

def lowercase_filter(tokens):
    return [token.lower() for token in tokens]

```

Indexing

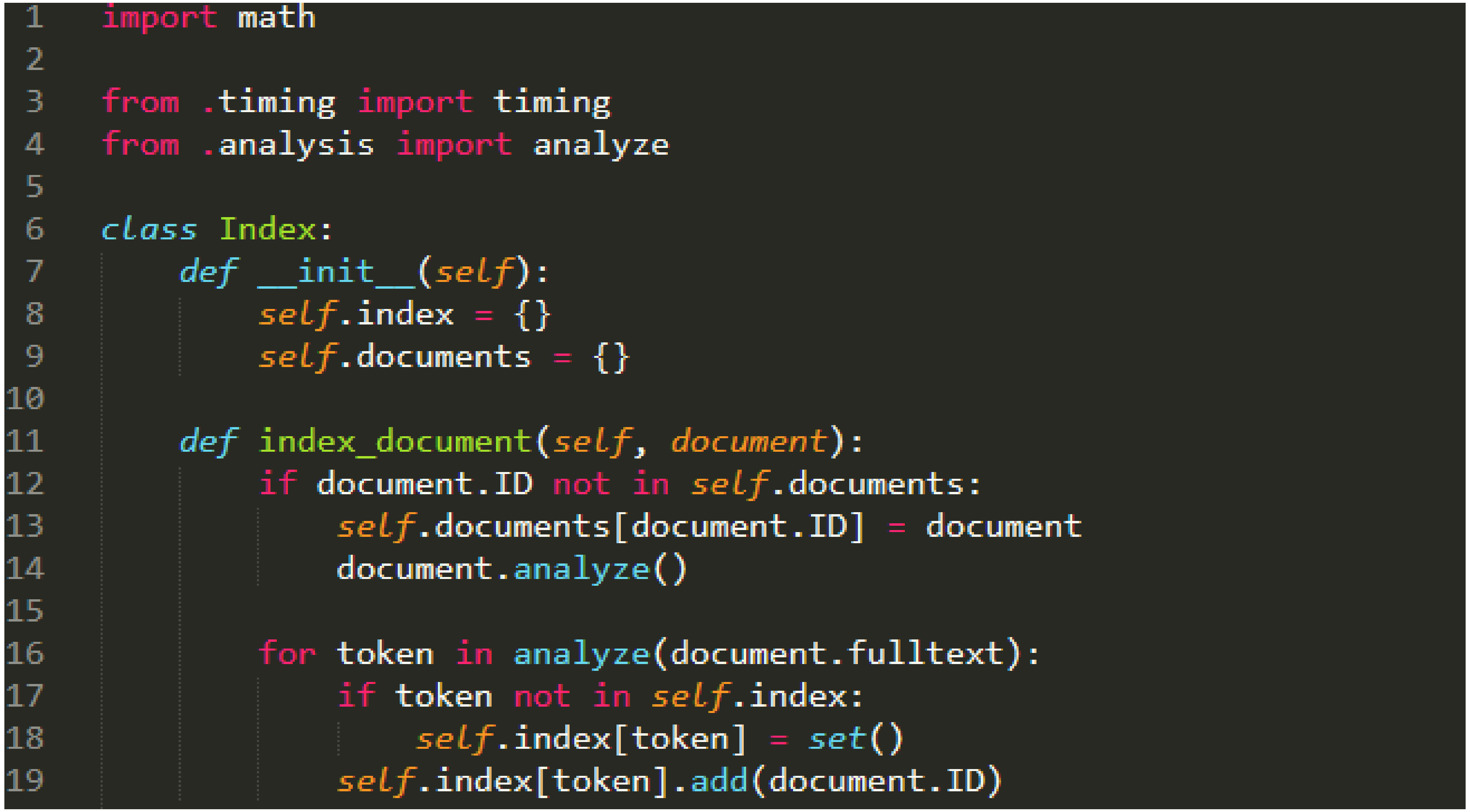
analysis.py



```python
def punctuation_filter(tokens):
    return [PUNCTUATION.sub('', token) for token in tokens]

def stopword_filter(tokens):
    return [token for token in tokens if token not in STOPWORDS]

def stem_filter(tokens):
    return STEMMER.stemWords(tokens)

def analyze(text):
    tokens = tokenize(text)
    tokens = lowercase_filter(tokens)
    tokens = punctuation_filter(tokens)
    tokens = stopword_filter(tokens)
    tokens = stem_filter(tokens)

    return [token for token in tokens if token]
```

Indexing the corpus

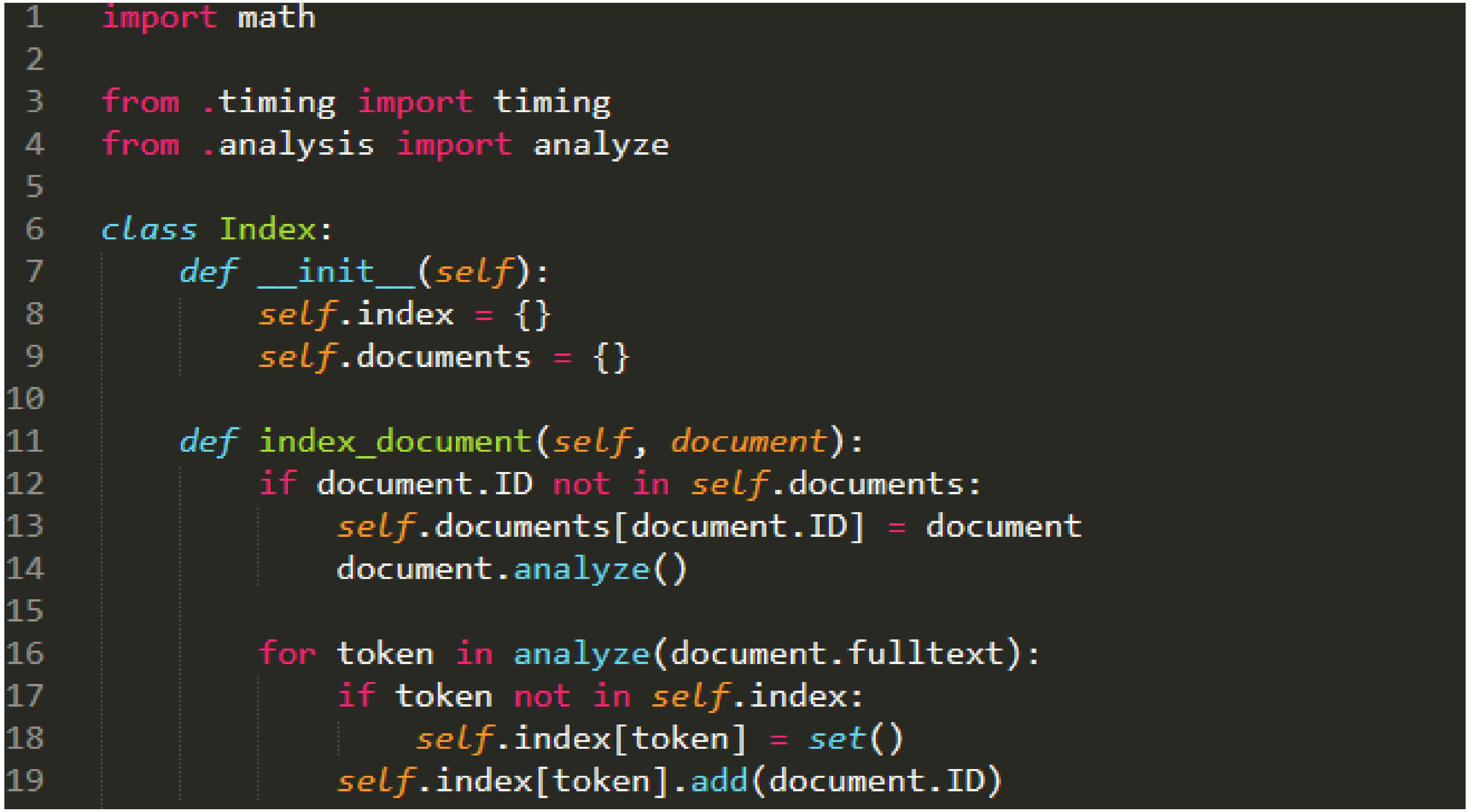
index.py



```python
import math

from .timing import timing
from .analysis import analyze

class Index:
    def __init__(self):
        self.index = {}
        self.documents = {}

    def index_document(self, document):
        if document.ID not in self.documents:
            self.documents[document.ID] = document
            document.analyze()

        for token in analyze(document.fulltext):
            if token not in self.index:
                self.index[token] = set()
            self.index[token].add(document.ID)
```

Indexing the corpus

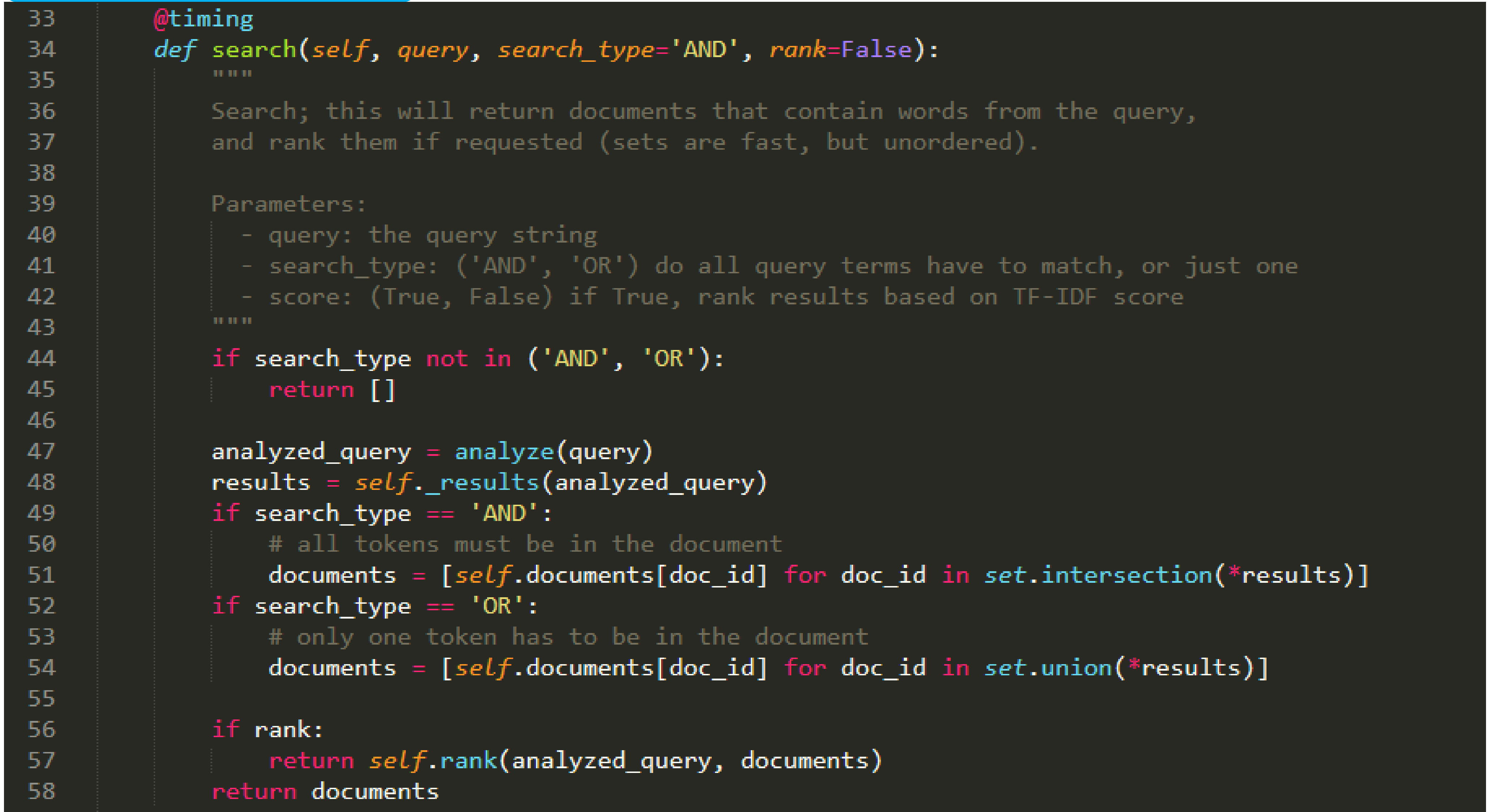
index.py



```python
import math

from .timing import timing
from .analysis import analyze

class Index:
    def __init__(self):
        self.index = {}
        self.documents = {}

    def index_document(self, document):
        if document.ID not in self.documents:
            self.documents[document.ID] = document
            document.analyze()

        for token in analyze(document.fulltext):
            if token not in self.index:
                self.index[token] = set()
            self.index[token].add(document.ID)
```

Searching

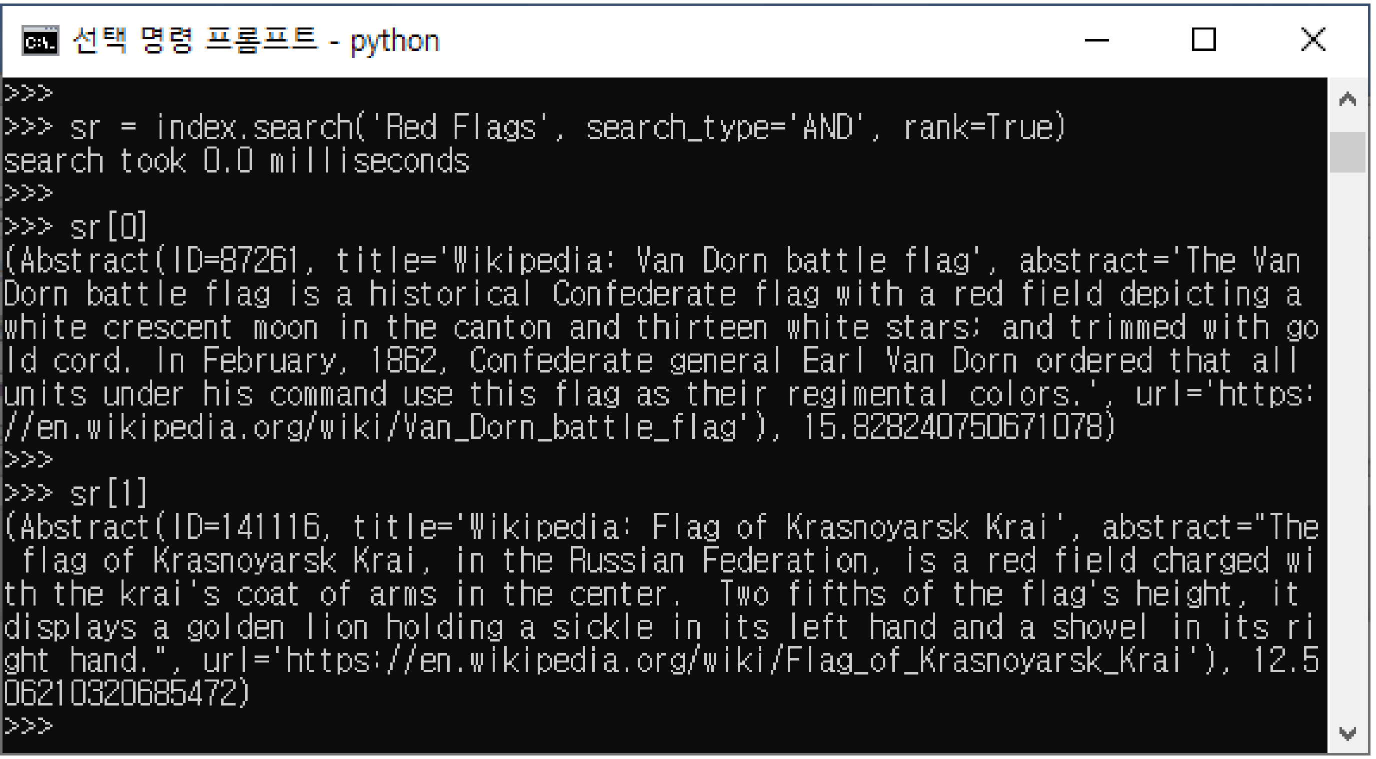
index.py



```
    @timing
    def search(self, query, search_type='AND', rank=False):
        """
        Search; this will return documents that contain words from the query,
        and rank them if requested (sets are fast, but unordered).

        Parameters:
          - query: the query string
          - search_type: ('AND', 'OR') do all query terms have to match, or just one
          - score: (True, False) if True, rank results based on TF-IDF score
        """
        if search_type not in ('AND', 'OR'):
            return []

        analyzed_query = analyze(query)
        results = self._results(analyzed_query)
        if search_type == 'AND':
            # all tokens must be in the document
            documents = [self.documents[doc_id] for doc_id in set.intersection(*results)]
        if search_type == 'OR':
            # only one token has to be in the document
            documents = [self.documents[doc_id] for doc_id in set.union(*results)]

        if rank:
            return self.rank(analyzed_query, documents)
        return documents
```

Run

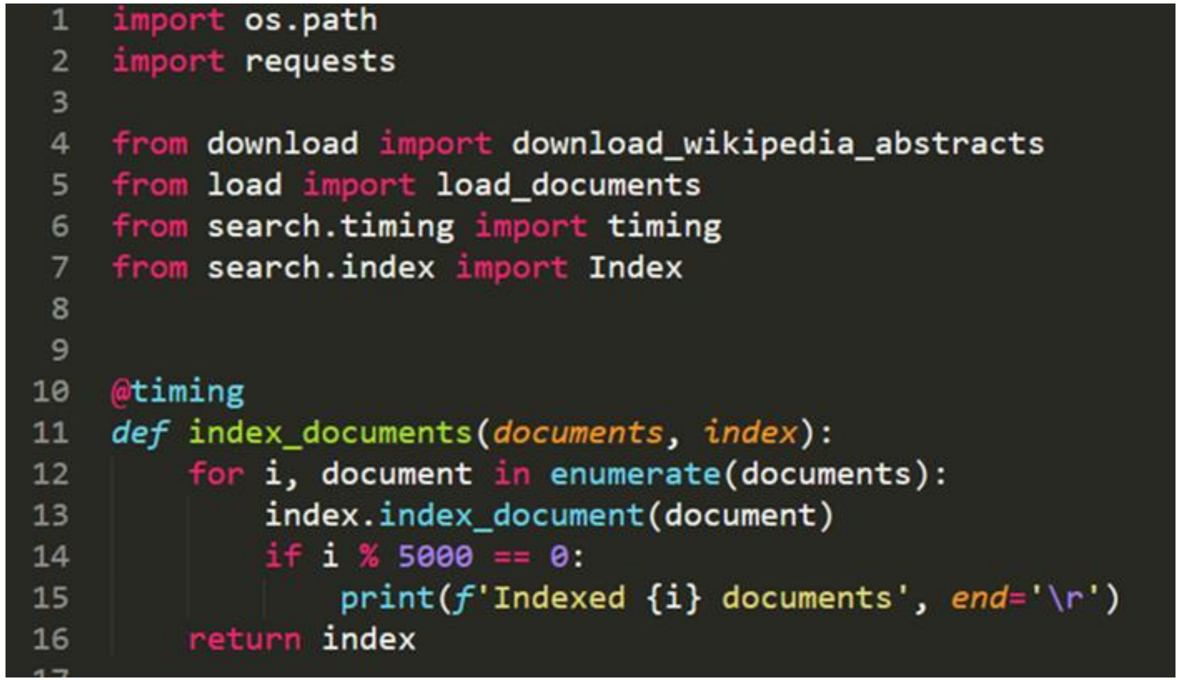
python 실행



선택 명령 프롬프트 - python

```
>>>
>>> sr = index.search('Red Flags', search_type='AND', rank=True)
search took 0.0 milliseconds
>>>
>>> sr[0]
(Abstract(ID=87261, title='Wikipedia: Van Dorn battle flag', abstract='The Van
Dorn battle flag is a historical Confederate flag with a red field depicting a
white crescent moon in the canton and thirteen white stars; and trimmed with go
ld cord. In February, 1862, Confederate general Earl Van Dorn ordered that all
units under his command use this flag as their regimental colors.', url='https:
//en.wikipedia.org/wiki/Van_Dorn_battle_flag'), 15.828240750671078)
>>>
>>> sr[1]
(Abstract(ID=141116, title='Wikipedia: Flag of Krasnoyarsk Krai', abstract="The
 flag of Krasnoyarsk Krai, in the Russian Federation, is a red field charged wi
th the krai's coat of arms in the center.  Two fifths of the flag's height, it
displays a golden lion holding a sickle in its left hand and a shovel in its ri
ght hand.", url='https://en.wikipedia.org/wiki/Flag_of_Krasnoyarsk_Krai'), 12.5
06210320685472)
>>>
```

run.py 복사 실행



```python
import os.path
import requests

from download import download_wikipedia_abstracts
from load import load_documents
from search.timing import timing
from search.index import Index


@timing
def index_documents(documents, index):
    for i, document in enumerate(documents):
        index.index_document(document)
        if i % 5000 == 0:
            print(f'Indexed {i} documents', end='\r')
    return index
```

Thank you

kgpark88@gmail.com