ch.9 Getting Data

Getting Data

```
from collections import Counter
import math, random, csv, json, re

from bs4 import BeautifulSoup
import requests
```

만약 어떤 모듈이 설치되지 않았다면, 예를 들어, beautifulsoup 같은,

- googling: 아나콘다 beautifulsoup 설치 방법
- 구글 답변에서 아나콘다 클라우드를 찾으세요. 모듈들이 테스트되고 안전한 곳입니다.
- 데이터 과학자로서 여러분은 부끄러운 정도로 많은 시간을
  - acquiring(획득),
  - cleaning(정리),
  - and transforming data(데이터 변환)에 할애하게 될 것입니다.

stdin and stdout

Number of lines containing numbers

- 그런 다음 이를 사용하여 숫자가 포함된 파일의 행 수를 셀 수 있습니다.
- sys.stdin(Keyboard) 및 sys.stdout(Monitor)을 사용하여 데이터를 파이프할 수 있습니다.

```
import sys, re

# sys.argv는 명령줄 인수의 리스트입니다.
# sys.argv[0]은 프로그램 자체의 이름입니다.
# sys.argv[1]은 명령줄에서 지정한 정규 표현식이 될 것입니다.
regex = sys.argv[1]

# 스크립트에 전달된 각 줄에 대해
for line in sys.stdin:
    # 만약 정규 표현식과 일치하는 경우, 그것을 stdout에 씁니다.
    if re.search(regex, line): # Regular Expression과 맞는 라인
        sys.stdout.write(line)
```

💡 명령줄에서 사용자가 지정한 정규 표현식을 이용하여 입력으로 받은 각 줄에 대해 검색하고, 해당 정규 표현식과 일치하는 경우 해당 줄을 표준 출력에 씁니다.

```
import sys

count = 0
# sys.stdin에서 각 줄을 읽어들여서
```

```
for line in sys.stdin:
    # count 변수를 증가시킵니다.
    count += 1

# print 함수는 sys.stdout에 출력됩니다.
print(count)
```

```
0
```

```
# Windows

!type the_bible.txt | python egrep.py "[0-9]" | python line_count.py
```

```
/bin/bash: line 1: type: the_bible.txt: not found
python3: can't open file '/content/egrep.py': [Errno 2] No such file or directory
python3: can't open file '/content/line_count.py': [Errno 2] No such file or directory
```

Most Common Words

- 입력된 단어를 세어 가장 일반적인 단어를 작성하는 스크립트

```
import sys
from collections import Counter

# 첫 번째 인자로 단어의 수를 전달합니다.
try:
    num_words = int(sys.argv[1]) # 명령어 argument -> Command
except:
    print("usage: most_common_words.py num_words")
    sys.exit(1)  # 비정상적 종료 코드는 오류를 나타냅니다.

# 표준 입력에서 각 줄을 읽어들여서
# 소문자로 변환한 단어를 카운트합니다.
counter = Counter(word.lower()                          # 소문자로 변환된 단어
                  for line in sys.stdin                 # 표준 입력으로부터의 각 줄
                  for word in line.strip().split()      # 공백으로 분리된 단어들
                  if word)                              # 빈 '단어'는 건너뜁니다.

# 가장 빈도가 높은 단어를 찾아 출력합니다.
for word, count in counter.most_common(num_words):
    sys.stdout.write(str(count))   # 빈도를 출력합니다.
    sys.stdout.write("\t")          # 탭 문자로 구분합니다.
    sys.stdout.write(word)          # 단어를 출력합니다.
    sys.stdout.write("\n")          # 줄 바꿈 문자로 줄을 바꿉니다.
```

```
# Windows

!type the_bible.txt | python most_common_words.py 10
```

```
64193   the
51380   and
34753   of
```

```
13643   to
12799   that
12560   in
10263   he
9840    shall
8987    unto
8836    for
```

Reading Files

The Basics of Text Files

```
# 'r'은 읽기 전용을 의미합니다.
file_for_reading = open('reading_file.txt', 'r')

# 'w'는 쓰기를 의미합니다 — 이미 파일이 존재하면 파일을 파괴합니다!
file_for_writing = open('writing_file.txt', 'w')

# 'a'는 추가를 의미합니다 — 파일 끝에 추가합니다.
file_for_appending = open('appending_file.txt', 'a')

# 파일을 사용한 후에는 파일을 꼭 닫아야 합니다.
file_for_writing.close() # Why? Open file 개수가 정해져 있기 때문
```

- 파일을 닫는 것을 잊기 쉽기 때문에 항상 블록과 함께 사용해야 하며, 블록이 끝나면 자동으로 닫힙니다.

```
with open(filename, 'r') as f:
    # 'f'를 통해 데이터를 가져오는 함수를 호출하여 데이터를 가져옵니다.
    data = function_that_gets_data_from(f)

# 이 시점에서 'f'는 이미 닫혔으므로 사용하지 마십시오.
# 가져온 데이터를 처리합니다.
process(data)
```

```
starts_with_hash = 0

# 'input.txt' 파일을 읽기 모드로 엽니다.
with open('input.txt', 'r') as f:
    # 파일의 각 줄을 확인합니다.
    for line in f:                      # 파일의 각 줄을 확인합니다.
        if re.match("^#", line):        # 정규식을 사용하여 줄이 '#'로 시작하는지 확인합니다.
            starts_with_hash += 1       # 만약 그렇다면, 카운트에 1을 더합니다.
```

```
def get_domain(email_address):
    """'@'를 기준으로 분할하고 마지막 부분을 반환합니다."""
    return email_address.lower().split("@")[-1]

# 'email_addresses.txt' 파일을 읽기 모드로 엽니다.
with open('email_addresses.txt', 'r') as f:
    # 파일의 각 줄에 대해 도메인을 가져와서 카운트합니다.
    # 주소에 '@'가 있는 줄만 처리합니다.
    domain_counts = Counter(get_domain(line.strip())       # 각 줄의 도메인을 가져와서
```

```
                                for line in f                      # 파일의 각 줄에 대해
                                if "@" in line)                  # '@'가 있는 줄만 처리합니다.
```

Delimited Files

- csv 파일 : 이러한 파일은 쉼표로 구분되거나 탭으로 구분되는 경우가 많습니다.

```
!type tab_delimited_stock_prices.txt
```

```
6/20/2014   AAPL    90.91
6/20/2014   MSFT    41.68
6/20/2014   FB  64.5
6/19/2014   AAPL    91.86
6/19/2014   MSFT    41.51
6/19/2014   FB  64.34
```

- csv.reader는 줄 단위 튜플 생성기입니다

```
import csv

# 'tab_delimited_stock_prices.txt' 파일을 읽기 모드로 엽니다.
with open('tab_delimited_stock_prices.txt', 'r') as f:
    # 파일을 탭으로 구분하여 읽는 CSV 리더를 생성합니다.
    reader = csv.reader(f, delimiter='\t')

    # CSV 파일의 각 행을 반복하면서
    for row in reader:
        # 각 행의 열을 추출합니다.
        date = row[0]                       # 첫 번째 열: 날짜
        symbol = row[1]                     # 두 번째 열: 기호
        closing_price = float(row[2])    # 세 번째 열: 종가 (부동 소수점으로 변환)

        # 날짜, 기호 및 종가를 출력합니다.
        print(date, symbol, closing_price)
```

💡 탭으로 구분된 텍스트 파일을 읽어들여서 각 행의 날짜, 기호 및 종가를 추출하고 출력합니다. CSV 모듈의 `csv.reader()` 함수를 사용하여 파일을 탭으로 구분하여 읽습니다.

```
6/20/2014 AAPL 90.91
6/20/2014 MSFT 41.68
6/20/2014 FB 64.5
6/19/2014 AAPL 91.86
6/19/2014 MSFT 41.51
6/19/2014 FB 64.34
```

```
%%bash
cat colon_delimited_stock_prices.txt
```

```
date:symbol:closing_price
6/20/2014:AAPL:90.91
```

```
6/20/2014:MSFT:41.68
6/20/2014:FB:64.5
```

```
import csv

# 'colon_delimited_stock_prices.txt' 파일을 읽기 모드로 엽니다.
with open('colon_delimited_stock_prices.txt', 'r') as f:
    # 파일을 콜론으로 구분하여 읽는 CSV 딕셔너리 리더를 생성합니다.
    reader = csv.DictReader(f, delimiter=':')

    # CSV 파일의 각 행을 반복하면서
    for row in reader:
        # 각 행의 필드 값을 딕셔너리에서 추출합니다.
        date = row["date"]                      # 'date' 필드
        symbol = row["symbol"]                  # 'symbol' 필드
        closing_price = float(row["closing_price"])  # 'closing_price' 필드 (부동 소수점으로 변환)

        # 날짜, 기호 및 종가를 출력합니다.
        print(date, symbol, closing_price)
```

💡 콜론으로 구분된 텍스트 파일을 읽어들여서 각 행의 날짜, 기호 및 종가를 추출하고 출력합니다. CSV 모듈의 `csv.DictReader()` 함수를 사용하여 파일을 읽고, 각 행을 딕셔너리로 나타냅니다.

```
6/20/2014 AAPL 90.91
6/20/2014 MSFT 41.68
6/20/2014 FB 64.5
```

```
%%bash
cat comma_delimited_stock_prices.txt
```

```
FB,64.5
MSFT,41.68
AAPL,90.91
```

```
import csv

# 오늘의 주가를 나타내는 딕셔너리입니다.
today_prices = {'AAPL': 90.91, 'MSFT': 41.68, 'FB': 64.5}

# 'comma_delimited_stock_prices_1.txt' 파일을 쓰기 모드로 엽니다.
with open('comma_delimited_stock_prices_1.txt', 'w') as f:
    # 파일을 콤마로 구분하여 쓰는 CSV 라이터를 생성합니다.
    writer = csv.writer(f, delimiter=',')

    # 오늘의 주가 딕셔너리에서 각 주식과 가격을 가져와서 파일에 쓰기합니다.
    for stock, price in today_prices.items():
        # 각 주식과 가격을 리스트로 만들어서 쓰기합니다.
        writer.writerow([stock, price])
```

💡 오늘의 주가를 나타내는 딕셔너리를 콤마로 구분된 텍스트 파일에 쓰는 예제입니다. CSV 모듈의 `csv.writer()` 함수를 사용하여 파일을 쓰고, 각 주식과 가격을 리스트로 만들어서 파일에 씁니다.

```
%%bash
cat comma_delimited_stock_prices_1.txt
```

```
AAPL,90.91
MSFT,41.68
FB,64.5
```

```
results = [["test1", "success", "Monday"],
           ["test2", "success, kind of", "Tuesday"],
           ["test3", "failure, kind of", "Wednesday"],
           ["test4", "failure, utter", "Thursday"]]

# don't do this!
with open('bad_csv.txt', 'w') as f:
    for row in results:
        f.write(",".join(map(str, row))) # might have too many commas in it!
        f.write("\n")                    # row might have newlines as well!
```

```
%%bash
cat bad_csv.txt
```

```
test1,success,Monday
test2,success, kind of,Tuesday
test3,failure, kind of,Wednesday
test4,failure, utter,Thursday
```

Scraping the Web

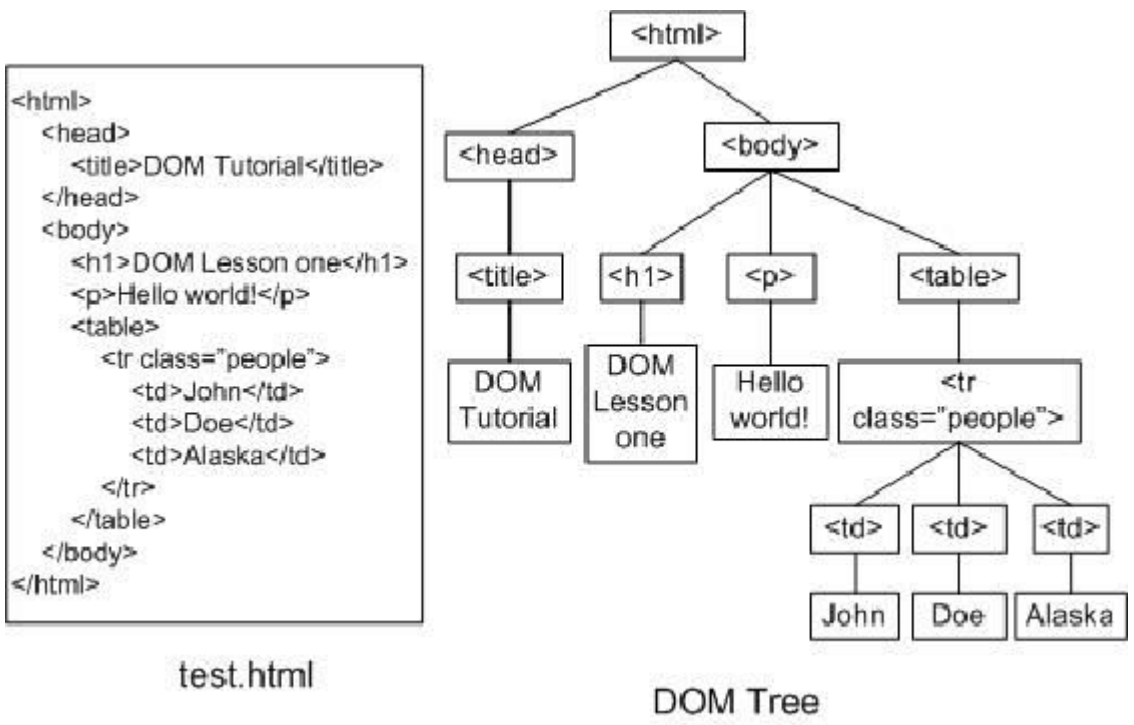
- 데이터를 얻는 또 다른 방법은 웹 페이지에서 스크랩하는 것입니다

HTML & Parsing

```
<html>
  <head>
    <title>A web page</title>
  </head>
  <body>
    <p id="author">Joel Grus</p>
    <p id="subject">Data Science</p>
  </body>
</html>
```

- for python3:
  - pip install html5lib
  - pip install beautifulsoup4

- for anaconda
  - conda install -c anaconda html5lib
    - https://anaconda.org/anaconda/html5lib
  - conda install -c anaconda beautifulsoup4
    - https://anaconda.org/anaconda/beautifulsoup4



DOM Lesson One

- Hello World!

John | Doe | Alaska

```
from bs4 import BeautifulSoup

html = """
<html>
  <head>
    <title>A web page</title>
  </head>
  <body>
    <p id="author">Joel Grus</p>
    <p id="subject">Data Science</p>
    <p class="price">30</p>
  </body>
</html>"""
soup = BeautifulSoup(html, 'html5lib')
```

💡 중요: Tag, Attribute, Text

`soup = BeautifulSoup(html, 'html5lib')` → Parsing 해서 자료구조화(Dict 화)

Query 1: Find title (제목 찾기)

```
soup.title.text
```

```
# 'A web page'
```

Query 2: Find title's text (제목 텍스트 찾기)

```
soup.title.text

# 'A web page'
```

Query 3: Find p of body (p의 본문 찾기)

```
soup.body.p

# <p id="author">Joel Grus</p>
```

Query 4: Find all p under body (p아래 본문 찾기)

```
soup.body('p')
```

```
[<p id="author">Joel Grus</p>,
 <p id="subject">Data Science</p>,
 <p class="price">30</p>]
```

Query 5: Find second p's text of body (두 번째 p의 본문 찾기)

```
soup.body('p')[1].text

# 'Data Science'
```

Query 6: Find last p of body (p의 마지막 부분 찾기)

```
soup.body('p')[-1]

# <p class="price">30</p>
```

Query 7: Loop over all p of body (p의 모든 부분에 루프를 씌우기)

```
for i, p in enumerate(soup.body('p')):
    print('paragraph {}: {}'.format(i, p.text))
```

```
paragraph 0: Joel Grus
paragraph 1: Data Science
paragraph 2: 30
```

Query 8: Find first p's id attribute's value (첫 번째 p의 ID 속성 값 찾기)

```
soup.p['id']

# 'author'
```

Query 9: Find all p whose attribute id is 'author' (속성 ID가 '저자'인 모든 p 찾기)

```
soup('p', {'id':'author'})

# [<p id="author">Joel Grus</p>]
```

Query 10: Find all p whose attribute class is 'price' (속성 클래스가 '가격'인 모든 p 찾기)

```
soup('p', 'price')
#soup('p', {'class':'price'})
```

```
[<p class="price">30</p>]
```

Query 11: Find all texts (모든 텍스트 찾기)

```
soup.text # List

# '\n    A web page\n  \n  \n    Joel Grus\n    Data Science\n    30\n  \n'
```

```
first_paragraph = soup.find('p')     # 또는 soup.p
print(first_paragraph)               # 첫 번째 <p> 태그의 내용을 출력합니다.
print(type(first_paragraph))         # 결과의 타입을 출력합니다.
```

```
<p id="author">Joel Grus</p>
<class 'bs4.element.Tag'>
```

💡 첫 번째 `<p>` 태그의 내용이 들어 있을 것이며, 그것의 타입은 BeautifulSoup의 특수한 타입인 `Tag`

```
first_paragraph_text = soup.p.text
first_paragraph_text

# 'Joel Grus'
```

```
first_paragraph_words = soup.p.text.split()
first_paragraph_words

# ['Joel', 'Grus']
```

💡 첫 번째 `<p>` 태그의 텍스트 내용을 추출한 후, `.split()` 메서드를 사용하여 공백을 기준으로 단어 단위로 분할

```
first_paragraph_id = soup.p['id']        # 'id' 속성이 없으면 KeyError를 발생시킵니다.
first_paragraph_id
#type(soup.p)

# Result: 'author'
```

💡 `soup`에서 첫 번째 `<p>` 태그의 `id` 속성 값을 가져와서 `first_paragraph_id` 변수에 할당합니다. 만약 해당 태그에 `id` 속성이 없다면 KeyError가 발생

```
first_paragraph_id2 = soup.p.get('id')   # 'id' 속성이 없으면 None을 반환합니다.
print(first_paragraph_id2)

# Result: 'author'
```

💡 `soup`에서 첫 번째 `<p>` 태그의 `id` 속성 값을 가져와서 `first_paragraph_id2` 변수에 할당합니다. 만약 해당 태그에 `id` 속성이 없다면 `None`을 반환

```
all_paragraphs = soup.find_all('p')      # 또는 soup('p')로도 가능합니다.
print(all_paragraphs)
```

```
[<p id="author">Joel Grus</p>, <p id="subject">Data Science</p>]
```

💡 `soup`에서 모든 `<p>` 태그를 찾아서 `all_paragraphs` 변수에 할당합니다. 그 결과는 리스트 형태로 반환

```
soup('p')

# [<p id="author">Joel Grus</p>, <p id="subject">Data Science</p>]
```

```
soup('p', {'id':'subject'})

# [<p id="subject">Data Science</p>]
```

💡 `id` 속성이 'subject'인 모든 `<p>` 태그를 찾는 예시입니다. 결과는 해당 조건에 맞는 `<p>` 태그들을 포함하는 리스트로 반환

`soup`에서 `id` 속성이 'subject' 인 모든 `<p>` 태그를 찾아 반환합니다. 두 번째 인자로는 딕셔너리를 사용하여 원하는 속성과 그 값의 조건을 지정

```
paragraphs_with_ids = [p for p in soup('p') if p.get('id')]
paragraphs_with_ids
```

```
[<p id="author">Joel Grus</p>, <p id="subject">Data Science</p>]
```

💡 BeautifulSoup 객체 `soup`에서 모든 `<p>` 태그를 찾은 후, 리스트 컴프리헨션을 사용하여 `id` 속성을 가진 태그들만 모아서 `paragraphs_with_ids` 리스트에 저장합니다.

결과는 `id` 속성을 가진 `<p>` 태그들의 리스트로 반환

```
important_paragraphs = soup('p', {'class': 'important'})

# [<p id="author">Joel Grus</p>, <p id="subject">Data Science</p>]
```

💡 `soup`에서 클래스가 'important'인 모든 `<p>` 태그를 찾아 반환합니다. 두 번째 인자로는 딕셔너리를 사용하여 원하는 클래스와 그 값의 조건을 지정

```
# 네이버 홈페이지의 HTML을 가져옵니다.
html = requests.get("http://www.naver.com").text

# HTML을 BeautifulSoup으로 파싱합니다.
soup = BeautifulSoup(html, 'html5lib')
```

💡 requests 모듈을 사용하여 네이버 홈페이지에 GET 요청을 보내고, 해당 페이지의 HTML을 가져옵니다. 그 후, BeautifulSoup을 사용하여 HTML을 파싱하여 객체로 저장합니다. 이렇게 하면 웹 페이지의 구조를 탐색하고 원하는 정보를 추출

```
# 경고: 만약 <span>이 여러 개의 <div> 안에 있는 경우, 같은 <span>을 여러 번 반환할 수 있습니다.
# 만약 그렇다면 좀 더 똑똑하게 처리해야 합니다.

spans_inside_divs = [span                       # <span>을 각각 리스트에 추가합니다.
                     for div in soup('div')   # 페이지의 각 <div>에 대해
                     for span in div('span')] # 그 안에 있는 각 <span>을 찾습니다.
```

💡 `soup`에서 모든 `<div>` 태그를 찾은 후, 각 `<div>` 태그 안에 있는 모든 `<span>` 태그를 찾아 리스트에 추가합니다. 그러나 만약 `<span>` 태그가 여러 개의 `<div>` 태그 안에 존재하는 경우, 동일한 `<span>`을 여러 번 반환

```
spans_inside_divs
```

💡 `spans_inside_divs` 변수에는 모든 `<div>` 태그 안에 있는 모든 `<span>` 태그가 포함된 리스트가 저장되어 있을 것입니다. 이 리스트는 각 `<div>` 태그에 대해 그 안에 있는 모든 `<span>` 태그를 포함하고 있습니다. 이 리스트를 출력하면 해당 정보를 확인

Example: O’Reilly Books About Data

- 한국외국어대학교 도서관에서 big data로 검색한 단행본의 연도별 단행본의 수를 bar chart로 나타내시오.
- 정말 지저분한 데이터
- 요즘 웹 페이지들에서 데이터를 추출하려면, javascript를 수행한 결과를 봐야하기 때문에 어렵다.

```
import requests
import re
from bs4 import BeautifulSoup

def extract_years(pnum):
    # 도서 목록이 있는 웹 페이지의 URL을 구성합니다.
    url = "https://library.hufs.ac.kr/yongin/search/Search.Result.ax?sid=1&q=ALL%3Abig+data&eq=&
         str(pnum) + "&pageSize=10&s=S_PYB&st=DESC&h=&cr=&py=&subj=&facet=Y&nd=&vid=0&tabID="

    # 해당 URL에서 HTML을 가져옵니다.
    hufs_lib_text = requests.get(url).text

    # BeautifulSoup을 사용하여 HTML을 파싱합니다.
    soup = BeautifulSoup(hufs_lib_text, 'html5lib')

    # 도서 목록을 찾습니다.
    booklist = soup('dl', 'bookList')

    # 발행 연도를 추출하기 위한 정규 표현식을 설정합니다.
    regex = re.compile('\d{4}\.')

    years = []
    # 각 도서 목록에서 발행 연도를 추출합니다.
    for book in booklist:
        years.append(int(regex.findall(book.find('div', 'body').text)[0][:-1]))

    return years
```

```
PAGENUM = 12
years = []
# 1부터 PAGENUM까지의 페이지에서 도서 목록을 가져와서 각 도서의 발행 연도를 추출합니다.
for pnum in range(1, PAGENUM + 1):
    years += extract_years(pnum)

print(len(years))  # 모든 페이지에서 추출된 도서의 발행 연도의 총 개수를 출력합니다.
```

💡 1부터 `PAGENUM` 까지의 페이지에서 `extract_years` 함수를 사용하여 도서 목록을 가져오고, 각 페이지에서 추출된 도서의 발행 연도를 `years` 리스트에 누적합니다. 마지막으로 `years` 리스트의 길이를 출력하여 모든 페이지에서 추출된 도서의 발행 연도의 총 개수를 확인

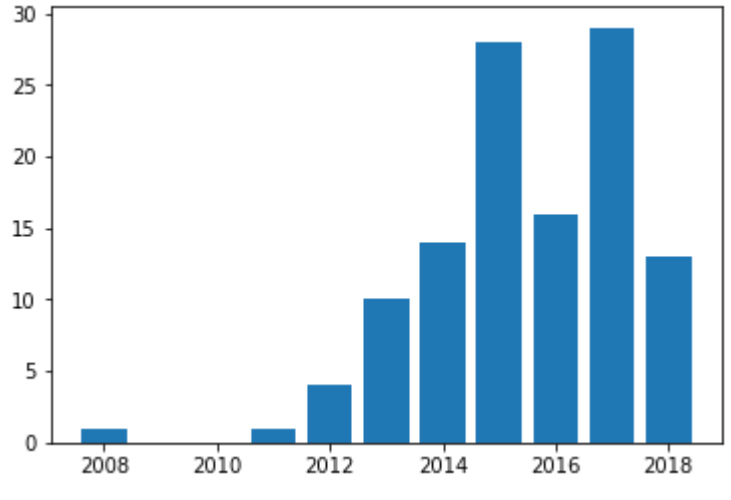
```
10
10
10
10
10
10
10
10
```

```
10
10
10
6
116
```

```
%matplotlib inline # 주피터 노트북에서 그래프를 인라인으로 표시하기 위한 매직 명령어

from collections import Counter
import matplotlib.pyplot as plt

# 발행 연도별 도서 수를 카운트합니다.
fd = Counter(years)
print(fd)

# 막대 그래프로 발행 연도별 도서 수를 표시합니다.
plt.bar(fd.keys(), fd.values())
plt.show()
```

💡 `Counter`를 사용하여 발행 연도별 도서 수를 카운트하고, 그 결과를 막대 그래프로 시각화

```
Counter({2017: 29, 2015: 28, 2016: 16, 2014: 14, 2018: 13, 2013: 10, 2012: 4, 2011: 1, 2008: 1})
```



Using APIs

트리 데이터 모델로서의 JSON (JSON as Tree Data Model)

- MongoDB는 JSON 스타일 트리 데이터베이스입니다
- 사전의 파이썬 사전으로 JSON...

```
{ "title" : "Data Science Book",
  "author" : "Joel Grus",
  "publicationYear" : 2014,
  "topics" : [ "data", "science", "data science"] }
```

```
import json

# JSON 형식의 문자열
serialized = """{ "title" : "Data Science Book",
                  "author" : "Joel Grus",
                  "publicationYear" : 2014,
                  "topics" : [ "data", "science", "data science"] }"""

# JSON을 파싱하여 Python 사전으로 변환합니다.
deserialized = json.loads(serialized)
```

💡 `json.loads()` 함수를 사용하여 JSON 형식의 문자열을 파싱하여 Python 사전으로 변환

`seralized` → String 이동, `deseralized` → Dictionary (자료구조화, seralization)

`loads` → dumps: 자료구조를 Text화

```
deserialized['title'] # JSON 문자열을 파싱하여 생성된 파이썬 딕셔너리, title 키를 사용하여 가져옴

# 'Data Science Book'
```

```
deserialized['author']  # JSON 문자열을 파싱하여 생성된 파이썬 딕셔너리, author 키를 사용하여 가져옴

# 'Joel Grus'
```

```
deserialized['publicationYear']

# 2014
```

```
deserialized['topics']

# ['data', 'science', 'data science']
```

```
if "data science" in deserialized["topics"]:
    print(deserialized)
```

💡 `deserialized 딕셔너리`의 "`topics`" 키에 "`data science`"가 있는지 확인하고, 있다면 해당 딕셔너리를 출력하는 예시

"`data science`"라는 항목이 `deserialized 딕셔너리의 "topics" 키에 포함`되어 있다면, 해당 딕셔너리를 출력

```
{'title': 'Data Science Book', 'author': 'Joel Grus', 'publicationYear': 2014, 'topics': ['data
```

XML as Tree Data Model

- tag, attribute, text에 대해 설명할 것
- `xml_text`: data 기술 용도로 사용

```
xml_text = """
<Book>
  <Title>Data Science Book</Title>
  <Author>Joel Grus</Author>
  <PublicationYear>2014</PublicationYear>
  <Topics>
    <Topic>data</Topic>
    <Topic>science</Topic>
    <Topic>data science</Topic>
  </Topics>
</Book>
"""
```

```
soup = BeautifulSoup(xml_text, 'lxml')

soup.book
```

💡 `xml_text`는 XML 형식의 문자열이고, 이를 BeautifulSoup으로 파싱하여 `soup` 객체에 저장합니다.

그리고 `soup.book`은 파싱된 XML에서 첫 번째 `<book>` 태그를 반환합니다.

만약 `<book>` 태그가 여러 개 있거나 하나도 없으면 `None`을 반환

```
<book>
<title>Data Science Book</title>
<author>Joel Grus</author>
<publicationyear>2014</publicationyear>
<topics>
<topic>data</topic>
<topic>science</topic>
<topic>data science</topic>
</topics>
</book>
```

Query 1: Find title of book

```
soup.book.title

# <title>Data Science Book</title>
```

Query 2: Find title's text of book

```
soup.book.title.text

# 'Data Science Book'
```

Query 3: Find author of book

```
soup.book.author

# <author>Joel Grus</author>
```

Query 4: Find all topic under topics

```
soup.topics('topic')

# [<topic>data</topic>, <topic>science</topic>, <topic>data science</topic>]
```

Query 5: Find second topic's text of topics of book

```
soup.book.topics('topic')[1].text

# 'science'
```

Query 6: Find last topic of book

```
soup.book('topic')[-1]

# <topic>data science</topic>
```

Query 7: Loop over all topic of book

```
for i, topic in enumerate(soup.book('topic')):
    print('topic {}: {}'.format(i, topic.text))

# topic 0: data
# topic 1: science
# topic 2: data science
```

💡 XML에서 `<book>` 태그 안에 있는 모든 `<topic>` 태그를 찾아서 순회하면서 각 `<topic>` 태그의 텍스트 내용을 출력하는 예시

`enumerate()` 함수를 사용하여 `<book>` 태그 안에 있는 모든 `<topic>` 태그를 순회하면서 각각의 인덱스와 텍스트 내용을 출력합니다. 예를 들어, `topic 0: data`와 같은 형식으로 출력

Query 8: Find the title's text of all bookss whose author is 'Joel Grus' and publicationyear >= 2000

- 8번 질문: 'Joel Grus'의 저자가 'Joel Grus'인 모든 책의 제목 본문 찾기 >= 2000

```
for book in soup('book'):
    if book.author.text == 'Joel Grus' and int(book.publicationyear.text) >= 2000:
        print(book.title.text)

# Data Science Book
```

💡 XML에서 `<book>` 태그를 찾고, 각 책의 저자가 'Joel Grus'이고 출판 연도가 2000년 이상인 책의 제목을 출력하는 예시

`for` 루프를 사용하여 XML에서 `<book>` 태그를 찾고, 각각의 책을 순회합니다. 그리고 조건문을 사용하여 각 책의 저자와 출판 연도를 확인하여 조건에 맞는 책의 제목을 출력

Belows are ommitted in class; Students will catch up.

```
soup.topic

# <topic>data</topic>
```

```
soup.topic.text

# 'data'
```

```
soup.find('topic')

# <topic>data</topic>
```

```
soup.find_all('topic')

# [<topic>data</topic>, <topic>science</topic>, <topic>data science</topic>]
```

```
soup.text

# '\nData Science Book\nJoel Grus\n2014\n\ndata\nscience\ndata science\n\n\n'
```

Using an Unauthenticated API (인증되지 않은 API 사용)

```
import requests
import json

# GitHub API의 엔드포인트 URL
endpoint = "https://api.github.com/users/joelgrus/repos"

# requests 모듈을 사용하여 API에 GET 요청을 보내고,
# 응답을 JSON 형식으로 파싱하여 저장소 정보를 가져옵니다.
repos = json.loads(requests.get(endpoint).text)
```

```
from dateutil.parser import parse
from collections import Counter
```

```
# 저장소 생성일 정보를 파싱하여 dates 리스트에 저장합니다.
dates = [parse(repo["created_at"]) for repo in repos]

# 각 월별로 저장소 생성 횟수를 카운트합니다.
month_counts = Counter(date.month for date in dates)

# 각 요일별로 저장소 생성 횟수를 카운트합니다.
weekday_counts = Counter(date.weekday() for date in dates)

print(month_counts)   # 각 월별로 저장소 생성 횟수를 출력합니다.
print(weekday_counts) # 각 요일별로 저장소 생성 횟수를 출력합니다.
```

```
Counter({11: 5, 7: 5, 9: 4, 12: 3, 2: 3, 1: 3, 5: 3, 8: 2, 6: 1, 4: 1})
Counter({4: 7, 2: 7, 1: 5, 5: 4, 6: 3, 3: 3, 0: 1})
```

```
# 저장소 리스트에서 가장 최근에 생성된 5개의 저장소를 선택합니다.
last_5_repositories = sorted(repos,
                             key=lambda r: r["created_at"],
                             reverse=True)[:5]

# 각 저장소에서 사용된 언어를 추출하여 리스트로 저장합니다.
last_5_languages = [repo["language"]
                    for repo in last_5_repositories]

print(last_5_languages)  # 최근에 생성된 5개의 저장소에서 사용된 언어를 출력합니다.
```

```
[None, 'Python', 'Python', 'HTML', 'C++']
```

Example: Using the Twitter APIs

```
CONSUMER_KEY="L9c3arjSFPGlaCAqpH07x5PwR"
CONSUMER_SECRET="RgxfMXXTW8Avdg6DvJ1m3ka6zyOQLXUONWl4CiJdgxOh73KEfF"
```

```
from twython import Twython

# Twython 객체를 생성합니다.
twitter = Twython(CONSUMER_KEY, CONSUMER_SECRET)

# "trump"이라는 구문을 포함하는 트윗을 검색합니다.
for status in twitter.search(q='"trump"')["statuses"]:
    # 각 트윗의 작성자와 내용을 가져와서 출력합니다.
    user = status["user"]["screen_name"].encode('utf-8')
    text = status["text"].encode('utf-8')
    print(user, ":", text)
    print()
```

💡 Twython을 사용하여 트위터 API를 호출하여 "trump"이라는 구문을 포함하는 트윗을 검색하고, 각 트윗의 작성자와 내용을 가져와서 출력

```
from twython import Twython

# Twython을 임포트합니다.

def call_twitter_search_api():
    # Twython 객체를 생성합니다.
    twitter = Twython(CONSUMER_KEY, CONSUMER_SECRET)

    # "data science"이라는 구문을 포함하는 트윗을 검색합니다.
    for status in twitter.search(q='"data science"')["statuses"]:
        user = status["user"]["screen_name"].encode('utf-8')
        text = status["text"].encode('utf-8')
        print(user, ":", text)
        print()

from twython import TwythonStreamer

# 전역 변수에 데이터를 추가하는 것은 좋은 방법은 아니지만,
# 예시를 간단하게 만듭니다.
tweets = []

class MyStreamer(TwythonStreamer):
    """TwythonStreamer의 하위 클래스로서,
    스트림과의 상호작용 방법을 지정합니다."""

    def on_success(self, data):
        """트위터에서 데이터를 수신할 때 어떻게 할까요?
        여기서 데이터는 트윗을 나타내는 Python 객체입니다."""

        # 영어로 된 트윗만 수집합니다.
        if data['lang'] == 'en':
            tweets.append(data)

        # 충분한 양의 데이터를 수집하면 멈춥니다.
        if len(tweets) >= 10:
            self.disconnect()

    def on_error(self, status_code, data
```

💡 `Twython`을 사용하여 트위터의 검색 API와 스트리밍 API를 호출하는 예시입니다.

이 코드는 'data science'이라는 구문을 포함하는 트윗을 검색하고, 영어로 된 트윗을 수집합니다.

```
def call_twitter_search_api():
    twitter = Twython(CONSUMER_KEY, CONSUMER_SECRET)

    # "김정은"이라는 구문을 포함하는 트윗을 검색합니다.
    for status in twitter.search(q='"김정은"')["statuses"]:
        user = status["user"]["screen_name"]
#         user = status["user"]["screen_name"].encode('utf-8')
        text = status["text"]
#         text = status["text"].encode('utf-8')
```

```
        print(user, ":", text)
        print()
```

💡 Twython을 사용하여 트위터의 검색 API를 호출하여 "김정은"이라는 구문을 포함하는 트윗을 검색하고, 각 트윗의 작성자와 내용을 출력하는 예시입니다. 이 예시에서는 트윗의 작성자와 내용을 바이트로 인코딩하는 부분이 주석 처리

💡 트위터 API에서 반환되는 텍스트는 기본적으로 유니코드로 제공되기 때문에 대부분의 경우 인코딩이 필요하지 않습니다. 만약 특정 환경에서 유니코드를 처리할 수 없는 경우에만 바이트로 인코딩하여 사용

```
twitter = Twython(CONSUMER_KEY, CONSUMER_SECRET)

# "김정은"이라는 구문을 포함하는 트윗을 검색합니다.
for status in twitter.search(q='"김정은"')["statuses"]:
    user = status["user"]["screen_name"]
    text = status["text"]
    print(user, ":", text)
    print()
```

💡 Twython을 사용하여 트위터의 검색 API를 호출하여 "김정은"이라는 구문을 포함하는 트윗을 검색하고, 각 트윗의 작성자와 내용을 출력.