# 데이터 크롤링과 정제

3장. 크롤링 시작하기

# 목차

- 단일 도메인 내의 이동
- 전체 사이트 크롤링
  - 전체 사이트에서 데이터 수집
- 인터넷 크롤링

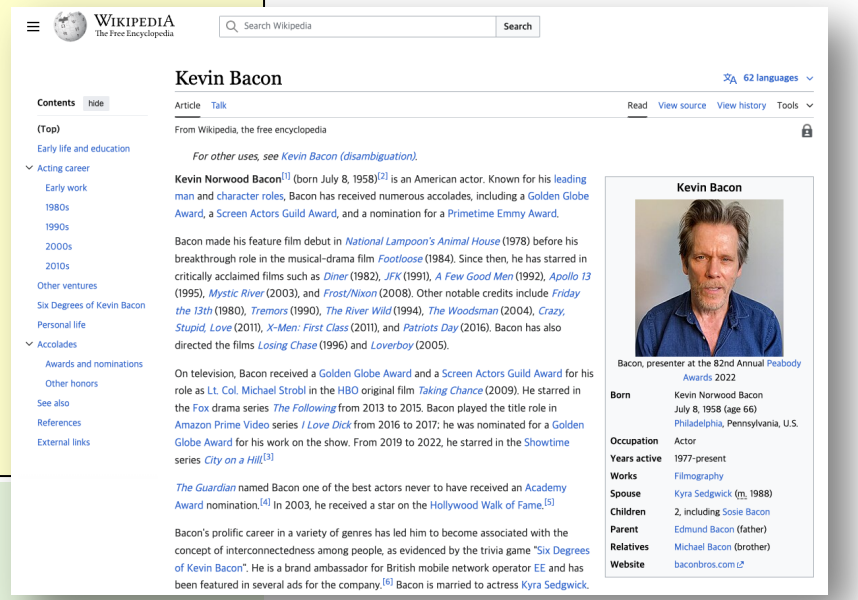
# Wikipedia 페이지 가져오기

- Kevin Bacon 위키피디아 URL
  - https://en.wikipedia.org/wiki/Kevin_Bacon

<chap03_01.py>

```
#임의의 위키 페이지에서 모든 링크 목록 가져오기
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('https://en.wikipedia.org/wiki/Kevin_Bacon')
bs = BeautifulSoup(html, 'html.parser')
for link in bs.find_all('a'):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```

<a> 태그 속성에 'href'가 있는 경우




```
#bodyContent
/wiki/Main_Page
/wiki/Wikipedia:Contents
/wiki/Portal:Current_events
/wiki/Special:Random
/wiki/Wikipedia:About
//en.wikipedia.org/wiki/Wikipedia:Contact_us
https://donate.wikimedia.org/wiki/Special:FundraiserRedirector?utm_source=donate&utm_me
dium=sidebar&utm_campaign=C13_en.wikipedia.org&uselang=en
/wiki/Help:Contents
```

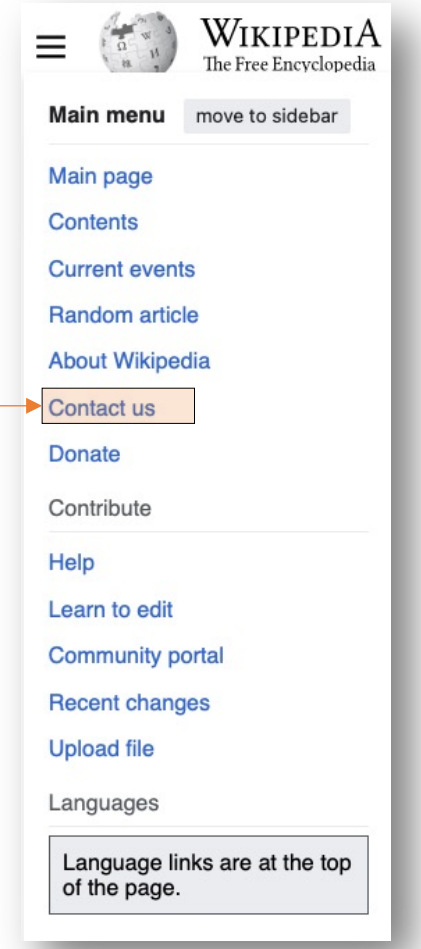
불필요한 내용이 많음

# 단일 도메인 내의 이동 #1

- 필요한 정보만 가져오기 위해, 위키백과 페이지를 분석할 필요

- 위키백과 페이지
  - 위키백과 내부 페이지 링크:'//en.wikipedia.org/wiki/…'

```
<li id="n-contactpage" class="mw-list-item">
    <a href="//en.wikipedia.org/wiki/Wikipedia:Contact_us" title="How to contact Wikipedia">
        <span>Contact us</span>
    </a>
</li>
```

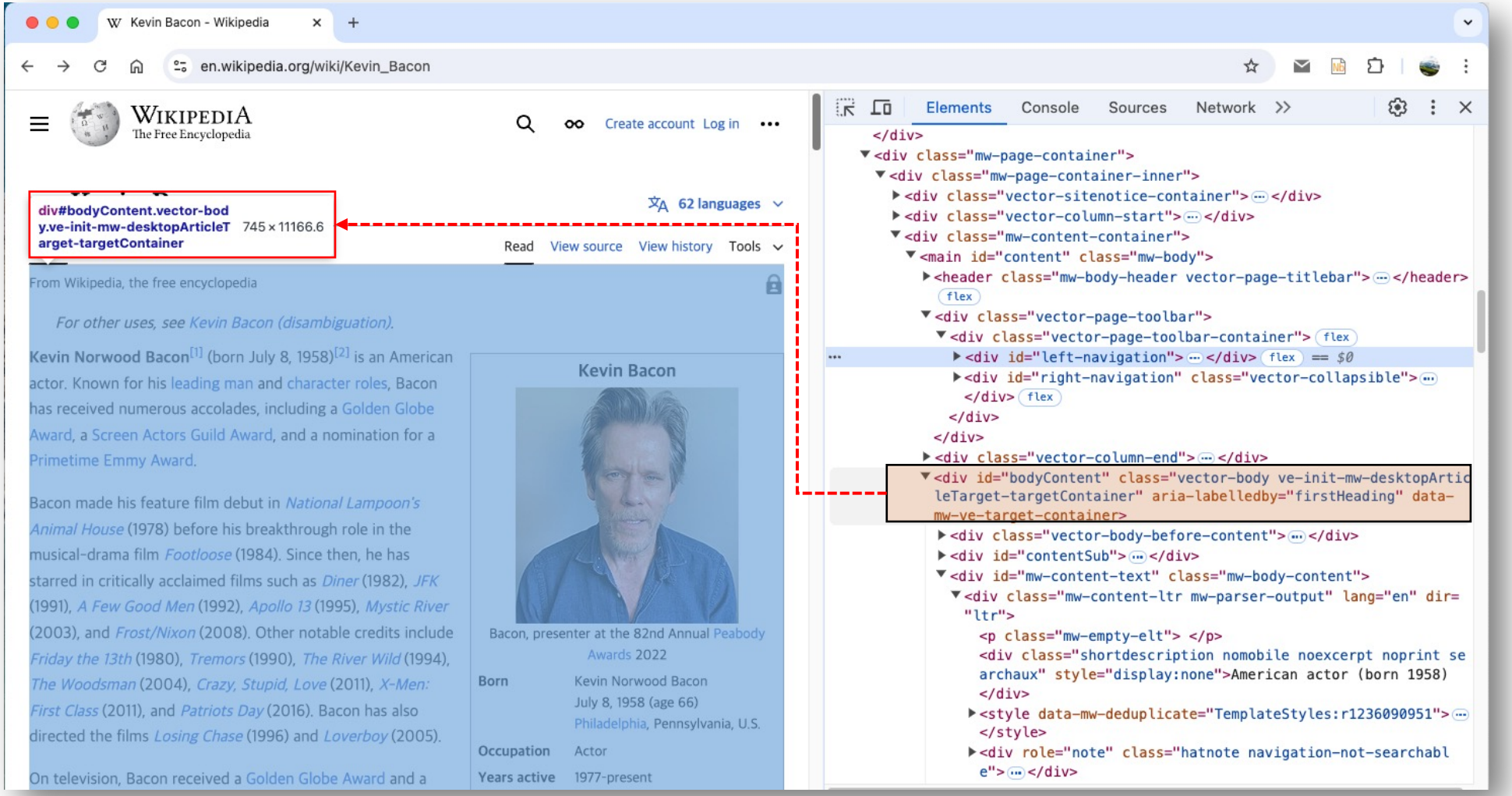



  - 위키백과의 연관 기사 내용 링크
    - 연관 기사 내용 링크
    - 기사 링크의 공통점
      -  태그의 id="bodyContent" 내부에 있음
      - URL에는 콜론이 포함되어 있지 않음
      - URL은 /wiki/로 시작

```
<i>
    <a href="/wiki/Footloose_(1984_film)" title="Footloose (1984 film)">Footloose</a>
</i>
 (1984), the controversial historical conspiracy legal thriller
<i>
    <a href="/wiki/JFK_(film)" title="JFK (film)">JFK</a>
</i>
```

# 위키백과의 연관 기사 내용 HTML 구성

- 위키백과의 기사 내용
  - 



# 연관 기사 링크 찾기

- 연관 기사 링크 가져오기
  - 연관 기사의 3가지 특성을 이용
  - 정규식: ^(/wiki/)((?!:).)*$
    - ^: 정규식 시작, $: 정규식 끝
    - (/wiki/): '/wiki/' 문자열 포함
    - ((?!:).)*: ':'이 없는 문자열 및 임의의 문자(.)가 0회 이상(*) 반복되는 문자열 검색
      - ➢ 전방 부정 탐색

<chap03_02.py>

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

html = urlopen('https://en.wikipedia.org/wiki/Kevin_Bacon')
bs = BeautifulSoup(html, 'html.parser')
body_content = bs.find('div', {'id': 'bodyContent'})

pattern = '^(/wiki/)((?!:).)*$'
for link in body_content.find_all('a', href=re.compile(pattern)):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```

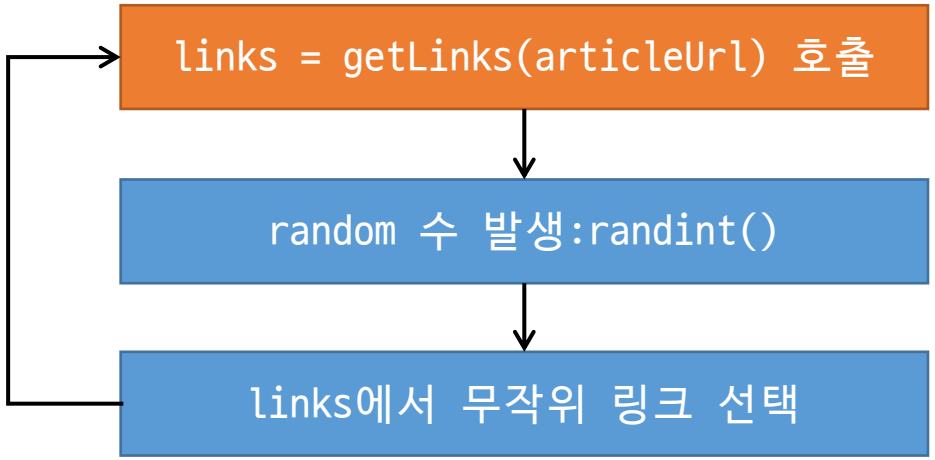
bodyContent 부분을 검색

```
/wiki/Kevin_Bacon_(disambiguation)
/wiki/Philadelphia
/wiki/Kevin_Bacon_filmography
...
/wiki/Michael_Keaton
/wiki/Sam_Elliott
```

# 링크간 무작위 이동하기: 동작 과정

- getLinks(articleUrl) 함수 작성
  - 파라미터: 임의의 '/wiki/<article_name>' 형태를 받음
  - 리턴값: 해당 링크의 모든 URL 목록을 리턴(리스트 형태)

- 동작 과정
  - 시작 URL: https://en.wikipedia.org/wiki/Kevin_Bacon
  - 시작 URL 내부의 연관 기사 URL을 가져옴
  - 연관 기사 URL에서 랜덤하게 하나의 URL 선택
  - 선택된 URL로 이동해서 다시 연관 기사 URL을 가져오는 과정 반복
    - 무한 반복




```
n = random.randint(a, b)
-  랜덤 숫자 n 리턴
-  a <= n <= b
```

# 링크간 무작위 이동하기

<chap03_03.py>

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import random
import re

#random.seed(datetime.datetime.now())
random.seed(None) # Python 3.9 이상

def getLinks(articleUrl):
    html = urlopen('https://en.wikipedia.org' + articleUrl)
    bs = BeautifulSoup(html, 'html.parser')
    bodyContent = bs.find('div', {'id': 'bodyContent'})
    wikiUrl = bodyContent.find_all('a', href=re.compile('^(/wiki/)((?!:).)*$'))
    return wikiUrl


links = getLinks('/wiki/Kevin_Bacon')
print('links 길이: ', len(links))
while len(links) > 0:
    newArticle = links[random.randint(0, len(links)-1)].attrs['href']
    print(newArticle)
    links = getLinks(newArticle)
```

random.seed(None)
- 난수 발생기 초기화
- None: 현재 시스템 시간을 사용

random.randint(start, end)
- 발견한 링크 중 무작위로 선택

실행 결과

```
links 길이:  447
/wiki/Rachel_Blanchard
/wiki/List_of_Clueless_characters#Cher_H
orowitz
/wiki/Virginity
/wiki/Anal_sex
/wiki/Wayback_Machine
/wiki/List_of_Web_archiving_initiatives
. . .

KeyboardInterrupt
```

# 전체 사이트에서 데이터 수집

- getLinks() 함수 수정
  - set 사용: 동일한 페이지를 두 번 크롤링하는 것을 방지

<chap03_04.py>

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

pages = set()
count = 0
def getLinks(pageUrl):
    global pages
    global count
    html = urlopen('https://en.wikipedia.org{}'.format(pageUrl))
    bs = BeautifulSoup(html, 'html.parser')
    for link in bs.find_all('a', href=re.compile('^(/wiki/)')):
        if 'href' in link.attrs:
            if link.attrs['href'] not in pages:
                # 새로운 페이지 발견
                newPage = link.attrs['href']
                count += 1
                print(f'[{count}]: {newPage}')
                pages.add(newPage) # 세트에 추가
                getLinks(newPage)

getLinks('')
```

재귀 호출 (자기 자신을 호출)
- getLinks(pageUrl) 함수 내부에서 자신을 호출
- Python에서는 재귀 호출을 1000회로 제한

재귀호출

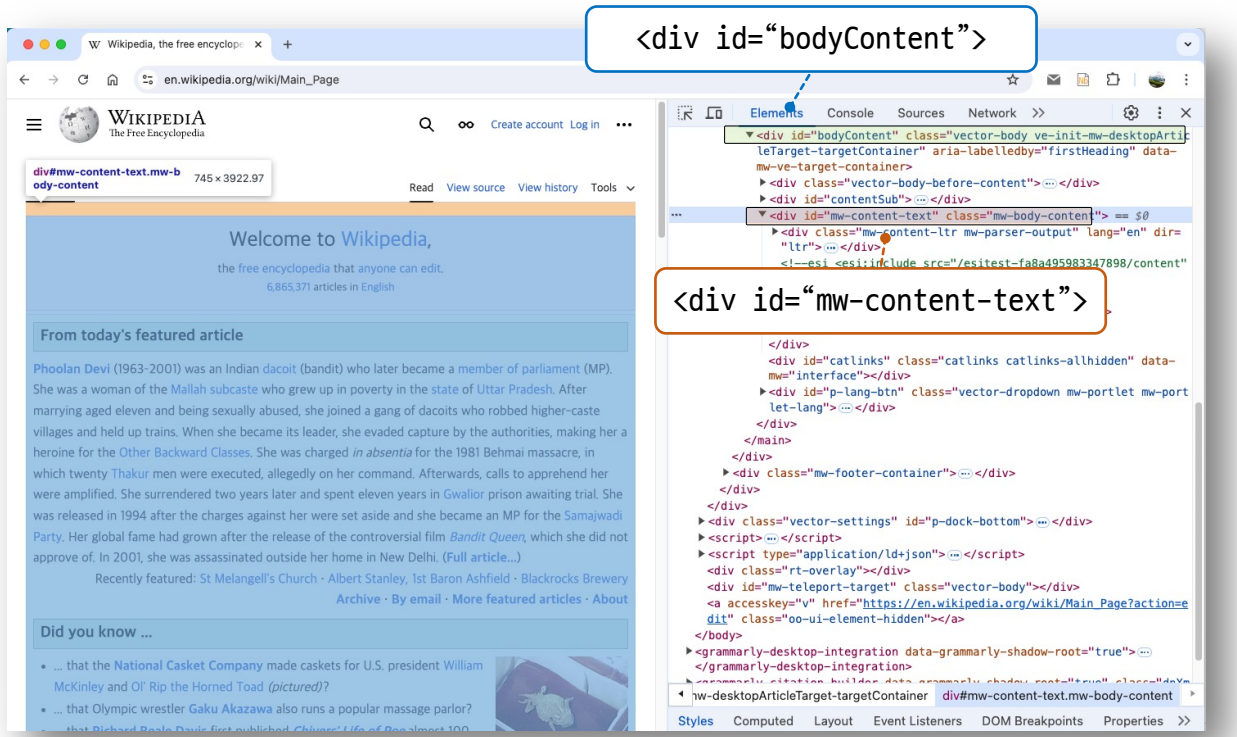
Set 내부에 해당 link가 없는지 확인 (not in)

Set에 새로운 link 추가
- add()함수 사용

```
[1]: /wiki/Main_Page
[2]: /wiki/Wikipedia:Contents
[3]: /wiki/Portal:Current_events
[4]: /wiki/Special:Random
[5]: /wiki/Wikipedia:About
[6]: /wiki/Help:Contents
[7]: /wiki/Help:Introduction
. . .
urllib.error.HTTPError: HTTP Error 404:
Not Found
```

# Wikipedia 전체 사이트에서 데이터 수집

- 페이지 방문 과정에서 필요한 정보를 추출
  - 수집 정보
    - 페이지 제목
    - 첫 번째 문단
    - 편집 페이지 링크 등




- Wikipedia 웹 페이지의 패턴 분석
  - 제목: <h1> 태그 사용 (하나만 사용)
  - body 텍스트: div#bodyContent 태그에 있음
  - 첫 번째 문단의 텍스트만 선택
    -  태그 사용
      - ➢  태그: 웹 페이지의 내용을 구분하는데 사용

# 전체 사이트 데이터 수집 소스

<chap03_05.py>

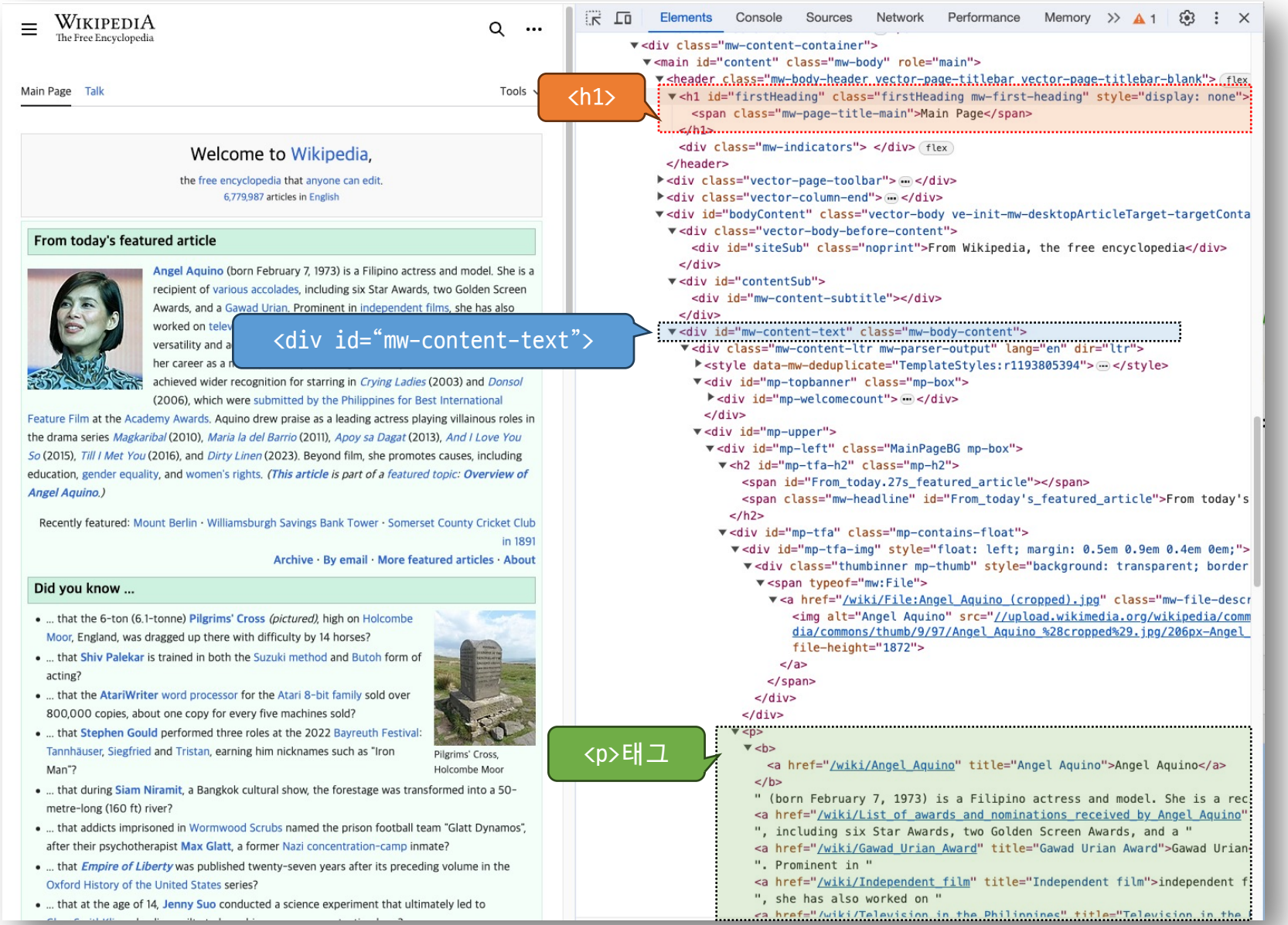
```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

pages = set()
count = 0
def getLinks(pageUrl):
    global pages
    global count
    html = urlopen('https://en.wikipedia.org{}'.format(pageUrl))
    bs = BeautifulSoup(html, 'html.parser')
    try:
        print(bs.h1.get_text()) # <h1>태그 검색
        #print(bs.find(id='mw-content-text').find('p').text)
        print(bs.find('div', attrs={'id': 'mw-content-text'}).find('p').text)
    except AttributeError as e:
        print('this page is missing something! Continuing: ', e)
```

```
    pattern = '^(/wiki/)((?!:).)*$'
    for link in bs.find_all('a', href=re.compile(pattern)):
        if 'href' in link.attrs:
            if link.attrs['href'] not in pages:
                newPage = link.attrs['href']
                print('-'*40)
                count += 1
                print(f'[{count}]: {newPage}')
                pages.add(newPage)
                getLinks(newPage)


getLinks('')
```

# 전체 사이트 데이터 수집 소스: 실행 결과

```
Main Page
T2 was a torpedo boat of the Royal Yugoslav Navy. Originally a 250t-class torpedo boat of the Austro-Hungarian Navy, commissioned on 11 August 1914 as 77T, she saw active service during World War I, performing convoy, patrol, escort, minesweeping and minelaying tasks, anti-submarine operations, and shore bombardment missions. Present in the Bocche di Cattaro during the short-lived mutiny by Austro-Hungarian sailors in early February 1918, members of her crew raised the red flag but took no other mutinous actions. The boat was part of the escort force for the Austro-Hungarian dreadnought Szent István when that ship was sunk by Italian torpedo boats in June 1918. Following Austria-Hungary's defeat in 1918, the boat was allocated to the Navy of the Kingdom of Serbs, Croats and Slovenes, which became the Royal Yugoslav Navy in 1921, and was renamed T2. During the interwar period, Yugoslav naval activity was limited by reduced budgets. Worn out after twenty-five years of service, T2 was scrapped in 1939. (This article is part of a featured topic: Ships of the Royal Yugoslav Navy.)

----------------------------------------
[1]: /wiki/Main_Page
Main Page
T2 was a torpedo boat of the Royal Yugoslav Navy. Originally a 250t-class torpedo boat of the Austro-Hungarian Navy, commissioned on 11 August 1914 as 77T, she saw active service during World War I, performing convoy, patrol, escort, minesweeping and minelaying tasks, anti-submarine operations, and shore bombardment missions. Present in the Bocche di Cattaro during the short-lived mutiny by Austro-Hungarian sailors in early February 1918, members of her crew raised the red flag but took no other mutinous actions. The boat was part of the escort force for the Austro-Hungarian dreadnought Szent István when that ship was sunk by Italian torpedo boats in June 1918. Following Austria-Hungary's defeat in 1918, the boat was allocated to the Navy of the Kingdom of Serbs, Croats and Slovenes, which became the Royal Yugoslav Navy in 1921, and was renamed T2. During the interwar period, Yugoslav naval activity was limited by reduced budgets. Worn out after twenty-five years of service, T2 was scrapped in 1939. (This article is part of a featured topic: Ships of the Royal Yugoslav Navy.)

----------------------------------------
[2]: /wiki/Wikipedia
Wikipedia

.  .   .
```

# wikipedia 초기 화면 구성



# 인터넷 크롤링

- 웹 크롤러를 만들기 전에 고려할 사항
  - 수집하려는 데이터는 무엇인가?
  - 특정 웹사이트에 도달하면, 새 웹사이트 링크를 따라가야 할까?
  - 특정 사이트를 제외할 것인가?
    - 다른 언어를 사용하는 웹사이트 정보 수집 여부
  - 저작권 침해 관련 문제는 없을까?

- 예제
  - 시작 URL: http://oreilly.com

# 인터넷 크롤링 예제 소스 분석 내용 #1

- 정규식
  - href=re.compile('^(/¦.*' + includeUrl + ')’)
    - ‘/’로 시작하는 링크를 찾음
    - ^: 문자열 시작
    - ( ): 그룹
    - /¦.*: ‘/’ 문자 또는(¦) 임의의 한 문자(.)가 없거나 여러 개 존재 (*: zero or more)

  - href=re.compile('^(http¦www)((?!' + excludeUrl + ').)*$’)
    - (http¦www): http 또는 www로 시작하는 문자열
    - (?!excludeUrl): excludeUrl 문자열이 존재하지 않는 링크
      - ➢ 전방 부정 탐색

# 인터넷 크롤링: URL 구조

- URL 구조

```
scheme://netloc/path;parameters?query#fragment
```

  - scheme: ‘http’ 또는 ‘https’
    - ftp, file, gopher, mms, news, nntp, sftp, telnet 등
  - netloc: 인터넷 주소

- urllib.urlparse
  - 파이썬 표준 라이브러리
  - HTTP요청, 파싱과 관련된 패키지

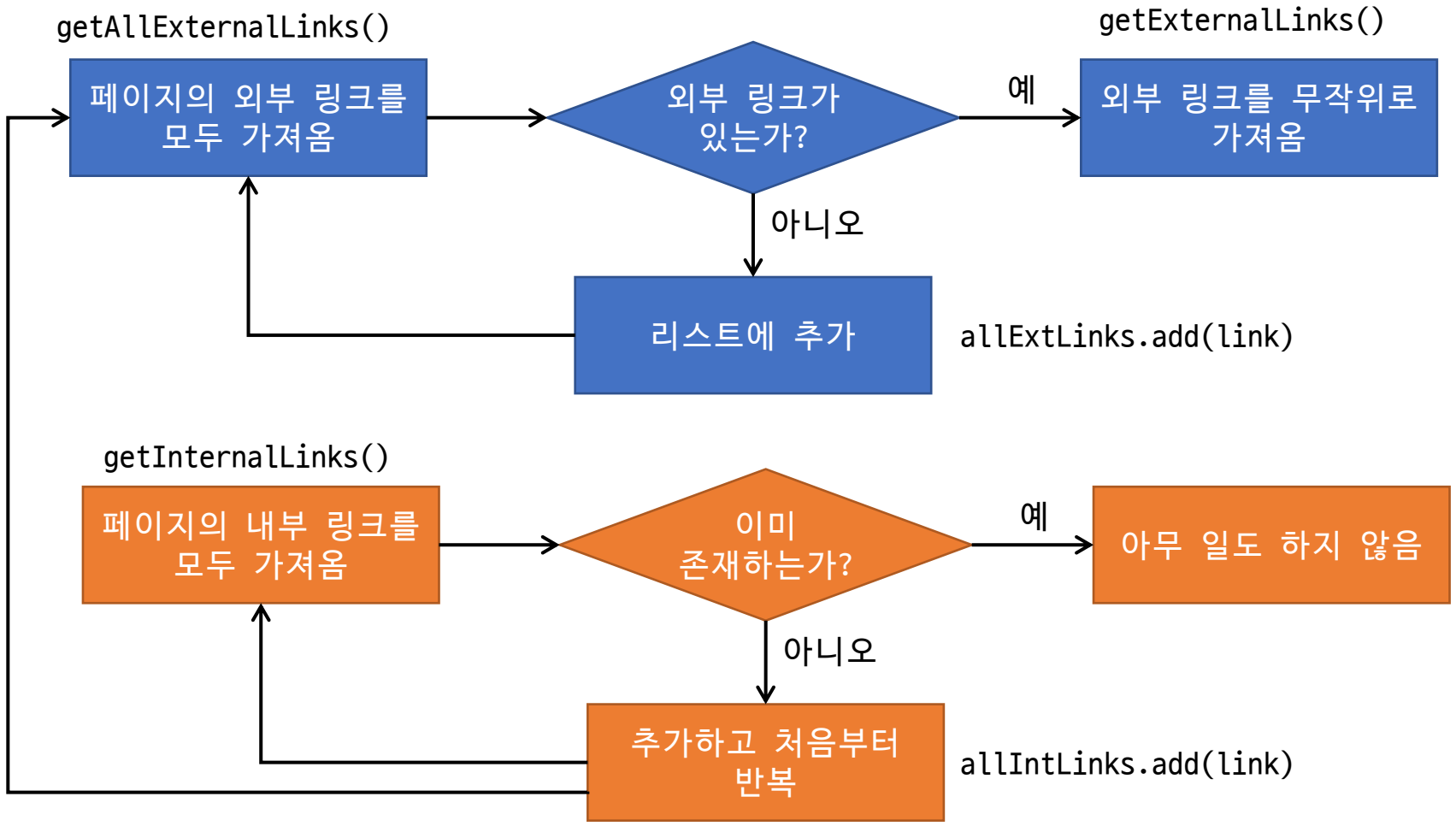
<chap03_05_urlparse.py>

```
from urllib.parse import urlparse

urlString1 = 'https://shopping.naver.com/home/p/index.naver'

url = urlparse(urlString1)
print(url.scheme)
print(url.netloc)
print(url.path)
```

```
https
shopping.naver.com
/home/p/index.naver
```

# 인터넷 크롤링: 인터넷 사이트 탐색 순서도

- 외부 링크 수집 과정



# 인터넷 크롤링: 외부, 내부 링크 모두 저장 #1

<chap03_07.py>

```
from urllib.request import urlopen
from urllib.parse import urlparse
from bs4 import BeautifulSoup
import re


# 웹 페이지에서 발견된 내부 링크를 모두 목록으로 만듦
def getInternalLinks(bs, includeUrl):
    includeUrl = f'{urlparse(includeUrl).scheme}://{urlparse(includeUrl).netloc}'

    internalLinks = []
    # "/"로 시작하는 링크를 모두 찾음"
    for link in bs.find_all('a', href=re.compile('^(/|.*' + includeUrl + ')')):
        if link.attrs['href'] is not None:
            if link.attrs['href'] not in internalLinks:
                if (link.attrs['href'].startswith('/')):
                    internalLinks.append(includeUrl + link.attrs['href'])
                else:
                    internalLinks.append(link.attrs['href'])
    return internalLinks
```

http://oreilly.com

'http://oreilly.com' + '/about/'

# 인터넷 크롤링: 외부, 내부 링크 모두 저장 #2

<chap03_07.py>

```
# 웹 페이지에서 발견된 외부 링크를 모두 목록으로 만듬
def getExternalLinks(bs, excludeUrl):
    externalLinks = []

    # 현재 URL을 포함하지 않으면서 http나 www로 시작하는 링크를 모두 찾음
    for link in bs.find_all('a', href=re.compile('^(http|www)((?!' + excludeUrl + ').)*$')):
        if link.attrs['href'] is not None:
            if link.attrs['href'] not in externalLinks:
                externalLinks.append(link.attrs['href'])

    return externalLinks
```

http://oreilly.com을
포함하지 않는 외부 링크

# 인터넷 크롤링: 외부, 내부 링크 모두 저장 #3

<chap03_07.py>

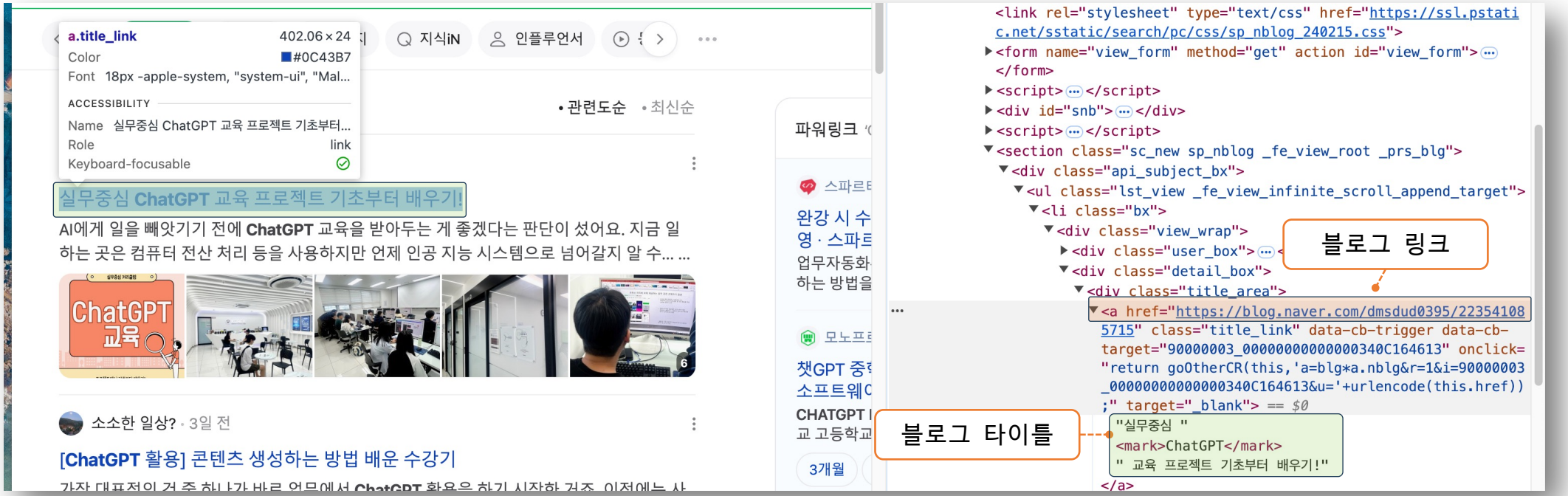
```
# 사이트에서 찾은 외부 URL을 모두 리스트로 수집
allExtLinks = set()
allIntLinks = set()

def getAllExternalLinks(siteUrl):
    html = urlopen(siteUrl)
    netloc_str = urlparse(siteUrl).netloc
    domain = f'{urlparse(siteUrl).scheme}://{urlparse(siteUrl).netloc}'
    bs = BeautifulSoup(html, 'html.parser')
    internalLinks = getInternalLinks(bs, domain)
    externalLinks = getExternalLinks(bs, netloc_str)

    for link in externalLinks:
        if link not in allExtLinks:
            allExtLinks.add(link)
            print(link)
    for link in internalLinks:
        if link not in allIntLinks:
            allIntLinks.add(link)
            getAllExternalLinks(link)

allIntLinks.add('http://oreilly.com')
getAllExternalLinks('http://oreilly.com')
```

```
https://www.linkedin.com/company/oreilly-media
https://www.youtube.com/user/OreillyMedia
https://oreilly.hk/
https://oreillylearning.in/
https://oreilly.id/
https://www.oreilly.co.jp/index.shtml
https://itunes.apple.com/us/app/safari-to-go/id881697395

. . .
```

# 네이버 블로그 검색

- 검색어: “ChatGPT”
  - https://search.naver.com/search.naver?ssc=tab.blog.all&sm=tab_jum&query=ChatGPT

- 첫 번째 블로그의 타이틀 부분
  - <a href=“https://blog.naver.com/dmsdud0395…” class=“title_link”>
    - 클래스 속성에 여러 이름이 존재하는 경우, dot(.)으로 접근 가능
    - select(a.title_link)



# 네이버 블로그 검색

<chap03_08_naver.py>

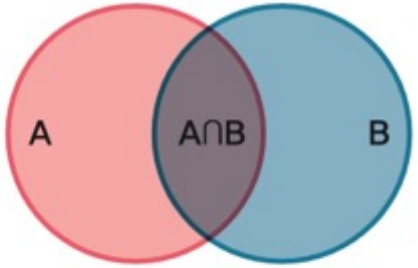
```
from urllib.request import urlopen
import requests
from bs4 import BeautifulSoup

query='ChatGPT'
url = f'https://search.naver.com/search.naver?where=view&sm=tab_jum&query={query}'
#response = requests.get(url)
#soup = BeautifulSoup(response.text, 'html.parser')

html = urlopen(url)
soup = BeautifulSoup(html.read(), 'html.parser')
blog_results = soup.select('a.title_link')
print('검색 결과수: ', len(blog_results))

for blog_title in blog_results:
    title = blog_title.text
    link = blog_title['href']
    print(f'{title}, [{link}]')
```

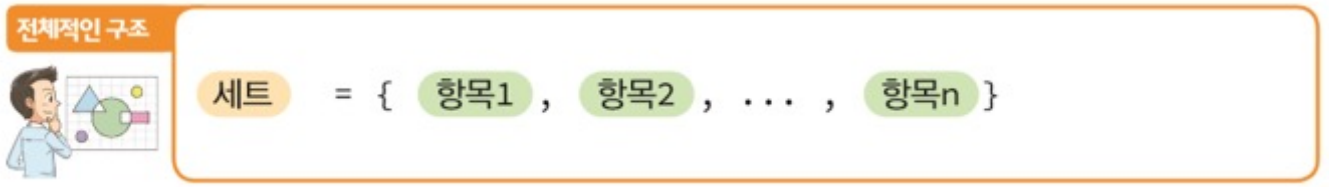
```
검색 결과수:  30
실무중심 ChatGPT 교육 프로젝트 기초부터 배우기!, [https://blog.naver.com/dmsdud0395/223541085715]
[ChatGPT 활용] 콘텐츠 생성하는 방법 배운 수강기, [https://blog.naver.com/songin06/223541110860]
chatgpt 4.0 사용법 무료 가능할까, [https://blog.naver.com/guitarjeus/223519289429]
chatgpt교육 메디치교육센터 무료로 배우고 취업하기!, [https://blog.naver.com/stacy0701/223517997493]
.  .  .
```

# 참고 자료

# 세트(Set)

- 세트(set)는 우리가 **수학에서 배웠던 집합**이다.
- 세트는 중복되지 않은 항목들이 모인 것
- 세트의 항목 간에는 순서가 없다.
- 파이썬에서 세트를 생성하려면 중괄호 기호{ }를 사용



# 세트: 중복 요소 자동 제거

- 세트는 집합이기 때문에
  - 요소가 중복되면 자동으로 중복된 요소를 제거함
- 중복된 원소를 포함하는 리스트의 경우, set으로 변경

```
cities = {“Paris”, “Seoul”, “London”, “Berlin”, “Paris”, “Seoul”}
print(cities)
mySet = {1, 2, 3, 2, 5, 4, 5, 3}
print(mySet)

myList = [1, 2, 3, 2, 5, 4, 5, 3]
print(myList)
mySet = set(myList) # 리스트를 세트로 변경
print(mySet)
myList = list(mySet) # 세트를 리스트로 변경
print(myList)
----------------
실행 결과
{'Paris', 'Seoul', 'Berlin', 'London’}
{1, 2, 3, 4, 5}
[1, 2, 3, 2, 5, 4, 5, 3]
{1, 2, 3, 4, 5}
[1, 2, 3, 4, 5]
```

# in 연산자

```
numbers = {2, 1, 3}
if 1 in numbers:
    print("집합 안에 1이 있습니다.")

---------------
실행 결과

집합 안에 1이 있습니다.
```

if 문장에서 사용된 in은 **존재 여부**를 확인

```
numbers = {2, 1, 3}
for x in numbers:
    print(x, end=" ")

---------------
실행 결과

1 2 3
```

for 문장에서 사용된 in은 **요소를 하나씩 가져옴**

출력 순서는 입력 순서와 다를 수 있음

# 세트에 요소 추가하기

```
numbers = { 2, 1, 3 }
numbers[0]
...
TypeError: 'set' object does not support indexing
```

set는 순서가 없기 때문에 index를 사용하여 세트 항목에 접근할 수 없음

- add(항목): 요소 추가

```
numbers = { 2, 1, 3 }
numbers.add(4)    # Set에 요소 추가
print(numbers)
numbers.discard(4) # discard(): Set에서 요소 삭제
print(numbers)     # discard()는 없는 요소를 삭제해도 예외 발생하지 않음
numbers.remove(3)  # remove(): Set에 없는 요소를 삭제하면 예외를 발생시킴
print(numbers)
numbers.clear()    # clear(): 모든 요소 삭제
print(numbers)
---------------
실행 결과
{1, 2, 3, 4}
{1, 2, 3}
{1, 2}
set()
```

# Questions?