Fast campus

파이썬을 활용한 실전 웹크롤링

크롤링의 법적문제와 다양한 API 크롤링

2018.08.18

신준호

jjjjooonno@gmail.com

이론 - 목차

1. selenium.common.exceptions 살펴보기
2. robots.txt에 대한 이해 & 실제 크롤링 법적 판례 살펴보기
3. header와 cookie에 대한 이해
4. API 활용해보기

실습 - 목차

1. exception 해결하는 코드 작성해보기
2. robots.txt 확인해보기
3. header와 cookie로 로그인해보기
4. API 활용해보기

exceptions 학습의 목적.

이제까지 나왔던 오류들에 대해서 좀더 상세히 학습하고, 오류마다 어떻게 해결해야 하는지를 학습합니다.

1. WebDriverException 에러메세지

WebDriverException: Message: unknown error:

Element **우리가 찾은 엘리먼트 A** is not clickable at point (**엘리먼트의 위치**).

Other element would receive the click: 다른 엘리먼트

다른 엘리먼트가 우리가 클릭하려는 엘리먼트의 클릭을 받기 때문에 클릭할 수 없는 상황입니다.
스크롤을 통해 화면에서 두 엘리먼트의 겹침을 해소하고 나서 클릭을 해야합니다.
주로 광고가 많은 사이트에서 많이 일어나는 에러입니다.

Fast campus

1. WebDriverException 에러메세지

우리가 선택하려는 엘리먼트 A

다른 엘리먼트

1. WebDriverException 에러메세지

에러메시지 -
WebDriverException: Message: unknown error:

Element 우리가 넣은 엘리먼트 A is not clickable at point (엘리먼트의 위치).

Other element would receive the click: 다른 엘리먼트

다른 엘리먼트가 우리가 클릭하려는 엘리먼트의 클릭을 받기 때문에 클릭할 수 없는 상황입니다.
스크롤을 통해 화면에서 두 엘리먼트의 겹침을 해소하고 나서 클릭을 해야합니다.
주로 광고가 많은 사이트에서 많이 일어나는 에러입니다.

2. NoSuchElementException 에러메세지

NoSuchElementException

우리가 찾는 엘리먼트가 없을 때의 에러입니다.

이런 경우에는

1. iframe에 속해 있어 찾을 수 없는 경우
    1. 해결법 : switch_to.frame으로 우리가 원하는 엘리먼트가 속한 iframe으로 옮겨간다.

2. 아직 로딩이 되지 않아 찾을 수 없는 경우
    1. 해결법 : time.sleep을 주어 로딩을 기다려 준다.
    2. 혹은 스크롤을 통해 나타나는 데이터라면 스크롤을 하는 코드를 삽입해 줍니다.

2. NoSuchElementException 에러메세지

NoSuchElementException

3. CSS_selector 혹은 xpath를 잘못 지정해 준 경우
    1. 해결법 : 대부분 속성값, 즉 attribute를 오타내거나, xpath의 인덱스(대괄호 안의 숫자)를 잘못 기입한 경우가 많습니다. 이럴 경우에는 속성값이나, xpath의 인덱스가 몇번째까지 들어갈 수 있는지 확인해야 합니다.

4. 키워드를 바꾼 검색결과거나, 혹은 해당 사이트가 변경되었을 경우
    1. 오래된 코드를 사용하거나, 혹은 파라미터(검색어 혹은 조건)을 변경한 경우 결과 사이트의 구조가 바뀔 수 있습니다. 특히 css_selector보다 xpath에서 이런 오류가 잦습니다. 왜냐하면 css_selector는 어떻게 보면 절대적인 경로이지만, xpath는 상대적인 위치를 통해 찾는 상대 경로이기 때문입니다. 이런경우에는 실제 브라우저상에서 다시 찾아야만 오류를 해결할 수 있습니다.

2. NoSuchElementException 에러메세지

1번과 2번의 예시는 저번시간의 예제로 충분히 해결이 가능합니다.

그러나 3,4번의 경우는 개인적인 노력이 필요합니다.

우선 3번의 경우는 에러가 난 코드의 줄을 확인하여 그 선택자를 계속 확인하여 줄 필요가 있습니다. 생각보다 모든 사이트가 정규적인 HTML구조를 띄고 있는 경우가 없습니다. 따라서 매번 예외를 확인하여 try, except문을 사용하여 에러를 잡아줄 필요가 있습니다.

2. NoSuchElementException 에러메세지

또한 4번의 경우에는 키워드 혹은 parameter(get방식에서 url에 삽입하는 변수들)가 바뀌면서
결과 페이지의 구조가 바뀌어 xpath의 결과가 달라지는 경우입니다.

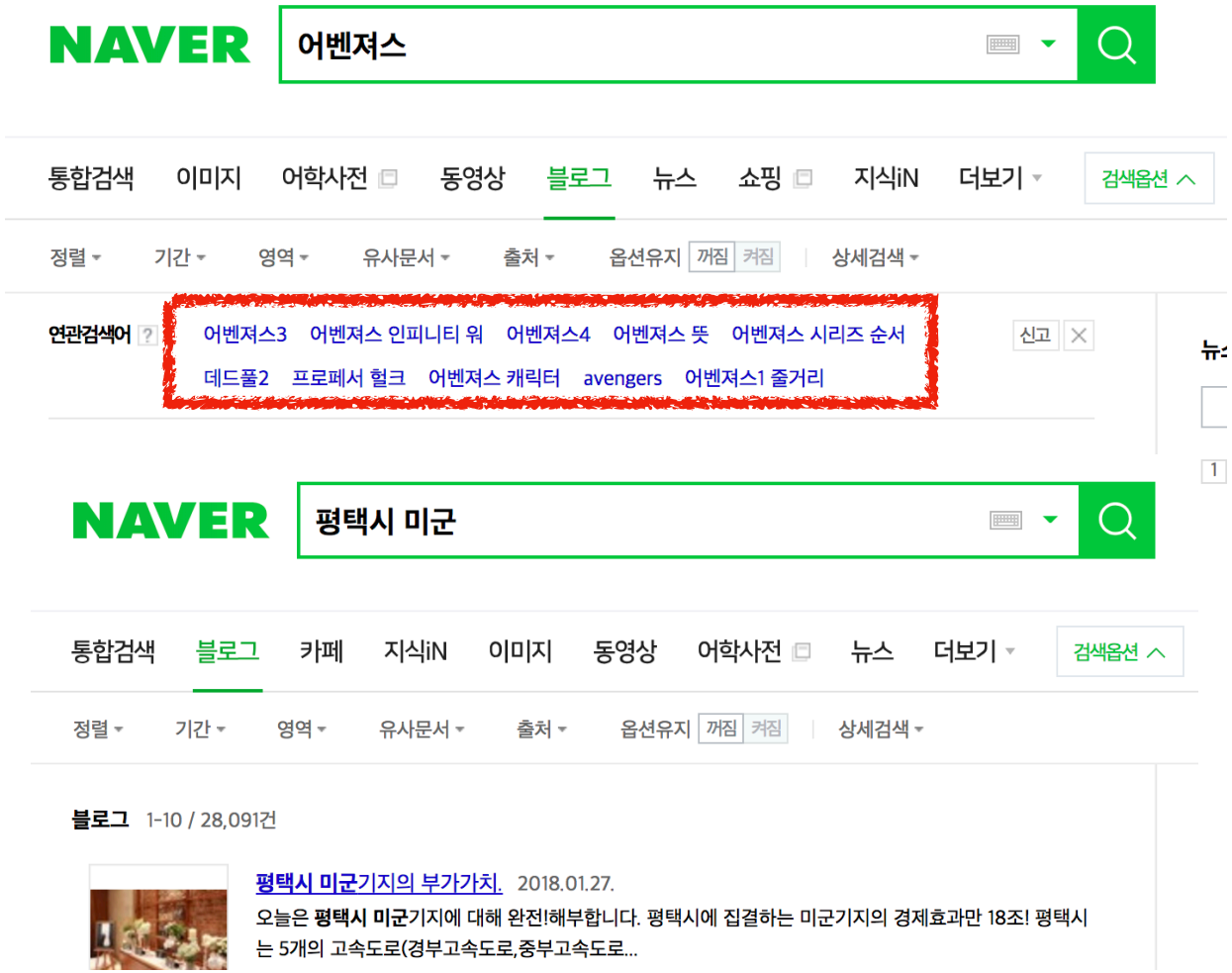
css_selector를 계속하여 사용하는 습관을 기르는 것은 이러한 경우 때문입니다.

css_selector는 왠만하면 이러한 예외에 면역이 되어 있습니다. 왜냐하면 절대 경로이기 때문에 속성값과
태그가 같다면 위치가 달라져도 찾을 수 있기 때문입니다.

하지만 xpath는 위치가 달라진다면 index가 달라져서 쉽게 오류가 나기 마련입니다.

예시 코드는 블로그 검색결과에서 연관검색어가 있는지의 여부에 따라서 블로그 검색 결과수를 찾는 xpath가
달라지는 것을 보여주고 있습니다. 이를 참고하여 xpath를 try문을 사용히여 사용하거나,
css_selector를 사용해야 합니다.

2. NoSuchElementException 에러메세지



두개 모두 같은 블로그 검색 결과이지만 연관검색어가 있고 없고의 차이가 있습니다.

따라서 계층이 차이가 나게 되고, 결과적으로 xpath의 인덱스가 달라지게 됩니다.

만약 css_selector가 아닌 xpath를 사용하여 블로그 건수를 찾는다면 아래와 위는 다른 xpath를 사용해야 합니다.

2. NoSuchElementException 에러메세지

```
from selenium import webdriver
from selenium.common.exceptions import NoSuchElementException
import time

dr = webdriver.Chrome('/Users/joono/chromedriver')
url = 'https://search.naver.com/search.naver?where=post&sm=tab_jum&query='
keyword = '평택시 미군'
# keyword = '어벤져스'
dr.get(url+keyword)

try:
    search_result_count = dr.find_element_by_xpath('//*[@id="main_pack"]/div[1]/div/span')
    print('1인경우')
    print(search_result_count)
except NoSuchElementException:
    search_result_count = dr.find_element_by_xpath('//*[@id="main_pack"]/div[2]/div/span')
    print('2인경우')
    print(search_result_count)
```

robots.txt와 법적문제 학습의 목적.

크롤링에 관련된 판례를 살펴보고 크롤링을 하기 전에 확인해야 할 최소한의 것들을 알아봅니다.

1. robots.txt와 법적문제 학습

robots.txt란 무엇인가?

robot.txt란 웹크롤러, 혹은 검색엔진의 로봇을 막기 위한 규약을 웹페이지마다 설정해 놓은 파일이다.

아직까지 법적 효력을 가지지는 못하는 권고안으로 알려져 있지만, 사실 뒤에 살펴볼 법원 판례에서 Robot.txt를 어느정도 인정하는 뉘앙스가 있으므로 무시할 수는 없다.

기본적인 구조는 다음과 같다.

```
# robots.txt for http://www.saramin.co.kr/
User-agent: Mediapartners-Google
Disallow:

User-agent: ia_archiver
Disallow:

User-agent: *
Disallow: /ad
Disallow: /zf_user/mandb
Allow:    /zf_user/mandb/view
Disallow: /zf_user/talent
```

1. robots.txt 확인하기

```
# robots.txt for http://www.saramin.co.kr/
User-agent: Mediapartners-Google
Disallow:

User-agent: ia_archiver
Disallow:

User-agent: *
Disallow: /ad
Disallow: /zf_user/mandb
Allow:    /zf_user/mandb/view
Disallow: /zf_user/talent
```

홈페이지 주소
특정할 로봇의 agent이름
Disallow가 비어있다는 것은 모두 허락한다는 것!
그러나 혹시라도 /가 들어가 있다면 모두를 금지!

*는 위에 특정한 agent를 제외한 모두를 말한다.
우리는 왠만하면 저 *에 속하니 저 밑에 있는 규약을 잘 살펴야 한다.
Disallow 뒤의 zf_user/mandb는 주소를 말한다.
즉 http://www.saramin.co.kr/zf_user/mandb가 url에 속해있다면 그 안의 어떤것이든 크롤링을 하지 말라는 명시이다.

2. 크롤링 법적 분쟁 사례 케이스

사람인 vs 잡코리아 크롤링 사건

사람인과 잡코리아라는 채용공고 게시 웹사이트가 있었습니다. 둘은 경쟁 업체였고, 둘중에 더 많은 구인공고 풀을 차지 하는 업체가 더 유리한것은 당연한 이치였습니다.
사람인은 잡코리아의 구인공고를 **크롤링**하여, 자사에 올라 오지 않은 공고가 있다면
구인하는 업체에 직접 연락하여 사람인에도 구인공고를 게시해도 되냐고 물어봐서 허락이 떨어지면, 잡코리아에 올라온 양식 그대로의 데이터를 자사에 게시하였고, 이 문제 때문에 잡코리아는 사람인을 민사로 고소하게 됩니다.

법원은 사람인이 잡코리아와의 공정경쟁을 침해 하였고, 지적재산권을 침해 하였다고 판결하게 되었고,
결국에는 사람인이 잡코리아에 4.5억원을 지불하고 관련 고소를 취하하는 결과를 불러왔습니다.

다음은 법원측의 판결에서의 의견입니다.

법원측 의견

"피고는 가상사설망을 쓰는 **VPN 업체를 통해 IP를 여러 개 로 분산한 뒤 검색로봇의 User-Agent에 피고의 정체를 명시하지 아니하고**, 크롤링해서는 안되는 페이지를 설명하는 **원고 웹사이트의 robots.txt를 확인하지도 아니한 채** 원 고 웹사이트의 HTML 소스를 크롤링하였는 바,
이는 정상적인 크롤링방식과는 차이가 있다”

이는 뒤에 배울 사람처럼 크롤링하기와 굉장한 관련이 있습니다.

3. 크롤링 할 때 가장 중요한 것!!

만약 여러분이 마케팅에 활용할 자료이든, 회사차원에서 필요한 디비를 적재할 목적이든, 크롤링을 시행하게 될 경우, 1차적으로 하실 업무는 크롤링의 목적이 되는 웹페이지의 운영자에 접촉하는 것입니다.

왜냐하면 방금의 판례를 살펴보면 robots.txt만으로도 웹페이지의 정보는 웹페이지의 운영자에게 재산권으로 인정되고, 함부로 갈취해서는 안되는 대상으로 인정이 되는것으로 보입니다.

따라서 우선은 필요한 데이터를 구입할 수 있는지 확인하고 나머지 업무를 진행해도 늦지 않는다는 것입니다. 사람인이 지불한 4.5억은 결코 작지 않은 금액입니다. 그리고 결국 대법원까지 진행한 법적 대치에는 법무팀, 대외적으로 선임한 변호사 수임료, 업무 조정 명령에 따른 손해 등등 수많은 금액이 더해집니다.

3. 크롤링 할 때 가장 중요한 것!!

크롤링은 물론 필요한 정보를 빠르고 자동적으로 수집해 주기에 업무에 큰 도움이 됩니다.
하지만 때로는 크롤링 자체가 회사의 발목을 잡을 큰 함정이 될 수 있기에 항상 조심하셔야 합니다.

데이터를 구입하는 비용이 1000억보다 싸다면 구입하시는게 더 나은 방법이 될 수 있다는 것입니다.

데이터를 내부적으로 사용하고, 의사결정에 도움이 되는 정도로 사용한다면 문제가 되지 않겠지만,
외부의 웹페이지의 정보를 사용하여 새로운 수익모델을 구상한다면 큰 문제가 될 수 있습니다.

크롤링 하기 전에 항상 내가 크롤링을 해서 이 데이터를 어디에 사용할 것인지를 생각하고 진행해 주시면 좋은 결과를 보기에 더욱 좋을 것입니다.

또한 디도스와 같은 공격은 어떻게 보면 크롤링과 비슷한 구조입니다. 웹서버에 단기간에 잦은 요청을 보내 웹서버를 다운시키는 것이기 때문입니다. 그래서 웹서버에 요청을 왠만하면 새벽에 보낸다던지, 혹은 낮에 보내야 한다면 시간간격을 여유있게 주시는 것이 좋습니다.

4. robots.txt 실습

다음의 코드는 내가 원하는 사이트의 robots.txt를 확인하고 내가 크롤링하려는 정확한 URL이 허용되는지를 확인해보는 코드입니다. 사람인은 한번 법적인 절차를 겪어서 그런지 굉장히 세부적으로 robots.txt를 짜놓았습니다.

앞으로 크롤링할 웹페이지의 robots.txt를 항상 확인하시고, 크롤링을 진행하시는게 회사와 본인을 위해서 항상 좋을 것입니다!

4. robots.txt 실습

```
from selenium import webdriver

dr = webdriver.Chrome('/Users/joono/chromedriver')

url = 'http://www.saramin.co.kr/robots.txt'

dr.get(url)
robotstxt = dr.find_element_by_tag_name('pre').text

want_part = robotstxt.split('User-agent: *')[1]
want_url = 'http://www.saramin.co.kr/zf_user/jobs/list/domestic?loc_mcd=101000'
if want_url.split('co.kr/')[1].split('/')[0] in want_part:
    print('크롤링 하지 마세요')
else:
    print('괜찮습니다')
```

header와 cookie 학습의 목적.

실제로 로그인이 자동으로 되는 크롬처럼 크롬드라이버에서도 로그인을 하기 위해 cookie를 사용합니다.

1. header와 cookie 학습

웹사이트의 운영자들은 자신들의 지적재산권을 지키기 위해 위에서 살펴본 robots.txt를
사용하기도 하지만 때로는 접근하는 우리 client의 header를 살펴보고 판단하기도 한다.

가끔 크롤링을 하려고 하는데 내가 브라우저에서 봤던 페이지가 아닌 페이지가 나올 때가 있다.
requests 패키지를 사용할 때 그런 경우가 많은데 대부분은 브라우저가 전달하는 정보보다 적기 때문이다.

그렇다면 이제까지 말했던 전달하는 정보가 적다는 것은 대체 무엇일까?

그 정보는 우리의 요청에 딸려가는 정보입니다.

우리가 사용하는 브라우저에서 get이나 post를 하게 되면 우리는 웹서버로 요청을 보낸다고 학습했다.

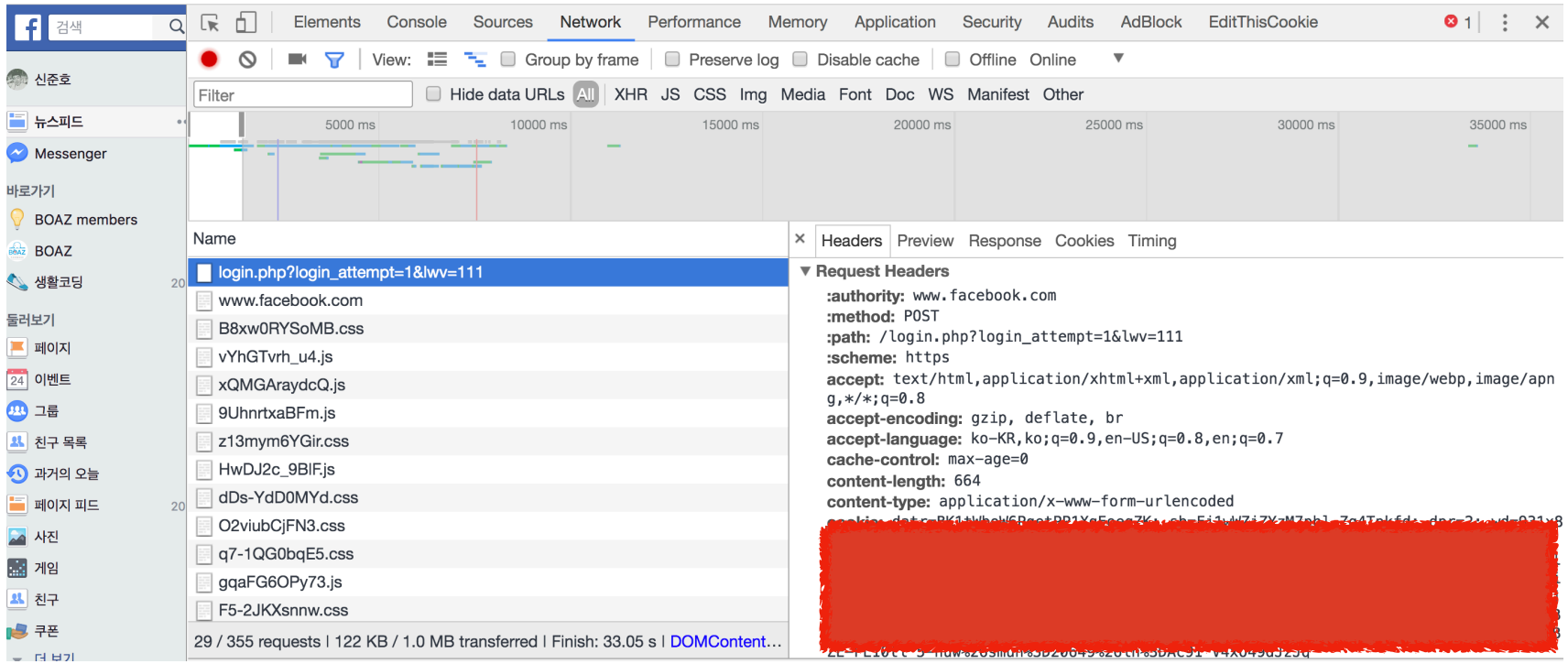
이 요청에는 url도 들어가지만, post에서 data라는 것도 같이 보내기도 한다.
실제로 get요청에는 request header라는 것도 같이 들어가게 된다. 이러한 정보들이 웹서버가 우리가
크롤링 봇인지 사람인지 확인하는 정보입니다!

그렇다면 이러한 정보는 대체 어디서 확인이 가능할까요?

이 요청은 개발자도구의 network 탭에서 가능합니다!

Fast campus

1. header와 cookie 학습

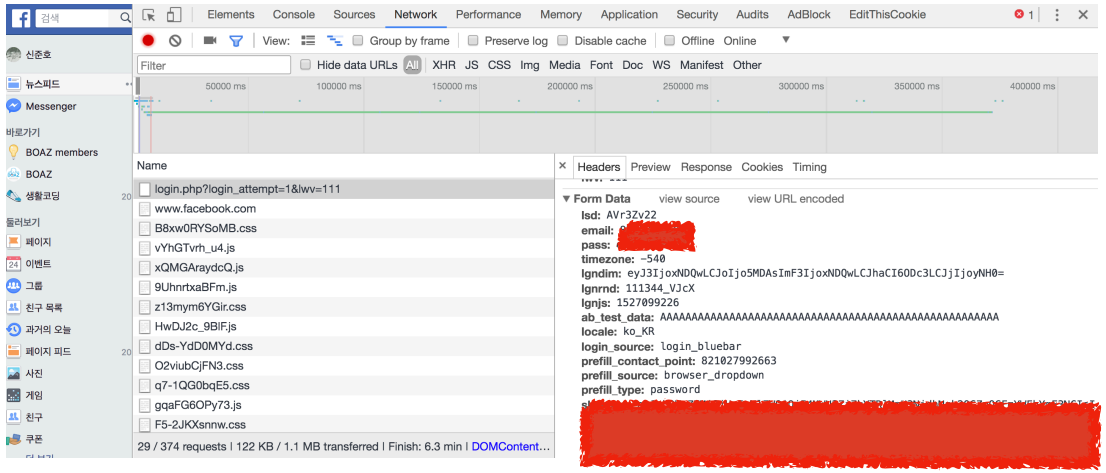


실제 강사의 아이디를 로그인한 상태이기 때문에 cookie부분은 가려놓았습니다!

이때 request headers 부분을 보면 수많은 인자들이 있는 것을 볼 수 있습니다.
이 header부분에 user-agent도 들어가게 되는데 이 부분이
페이스북 웹서버에서 우리가 누구인지 판단 할 수 있는 기준이 됩니다.

Fast campus

1. header와 cookie 학습



또한 우리의 요청의 방식이 POST이기 때문에 밑 부분에는 data에 무엇을 채워서 보냈는지 까지 나옵니다. 우리는 로그인 하기 위해 id와 password만 채웠지만 사실은 그것 말고도 수많은 데이터들이 페이스북 웹서버로 가게 됩니다. 만약 우리가 requests모듈을 사용해 로그인을 할때 봇으로 판명난다면 대부분이 form의 정보가 부족하기 때문인 경우가 많습니다.

1. header와 cookie 학습

아래의 코드는 로그인을 한번 해놓으면 생성되는 쿠키를 저장해놓고, 그 쿠키를 사용하여 자동로그인을 하게 되는
코드입니다. 페이스북을 예시로 해놓았지만, 네이버 다음 등 다양한 웹 페이지에서 사용이 가능합니다.

쿠키를 사용하면 자동로그인이 되기 때문에 로그인 회수 제한에 좀더 자유로울 수 있습니다.

또한 쿠키가 있다는 것은 웹브라우저를 사용해 실제 사람이 접근하는 것으로 속이기 쉽기 때문에 웹서버에서
크롤링봇으로 판단하는데 어려움이 있을 수 있습니다.

2. cookie로 로그인하기 실습

```
from selenium import webdriver
import time
import pickle
dr = webdriver.Chrome('/Users/joono/chromedriver')

url = 'https://www.facebook.com/'



dr.get(url)
id_input = dr.find_element_by_id('email')
id_input.send_keys('아이디넣기')
pw_input = dr.find_element_by_id('pass')
pw_input.send_keys('비밀번호넣기')
submit_button = dr.find_element_by_id('u_0_2')
submit_button.submit()
time.sleep(2)
pickle.dump(dr.get_cookies() , open("facebook_login_cookie.ini","wb"))
```

2. cookie로 로그인하기 실습

```
from selenium import webdriver
import time
import pickle
dr = webdriver.Chrome('/Users/joono/chromedriver')

url = 'https://www.facebook.com/'



dr.get(url)
cookies = pickle.load(open("cookies.ini", "rb"))
for cookie in cookies:
    print("cookies type ", type(cookie))
    print("dict ", cookie)
    dr.add_cookie(cookie)
dr.get(url)
```

가짜 에이전트를 사용하여 크롬드라이버 실행

아래의 코드는 가짜 에이전트를 사용하여 크롬드라이버를 실행하는 것입니다.

가짜 에이전트를 랜덤으로 생성하기 때문에 내가 맥으로 접근해도 윈도우인 것 처럼 속일 수도 있고,
또한 반대의 경우도 가능합니다. 반복적인 get을 해야 한다면 이러한 접근을 통해 여러 기기에서 접근하는 것으로 속일 수도 있습니다. 하지만 IP를 사용하여 접근을 제한하는 사이트에는 무용지물입니다.

만약 IP밴을 당하신다면 주변의 카페에 들어가서 그 카페의 와이파이로 접근해야 가능할 것입니다.

만약에 회사에서 여러가지의 와이파이를 제공한다면 와이파이를 바꿔가면서 진행하시는 것도 봇 판별을 피하는 좋은 방법이 될 수 있습니다.

가짜 에이전트를 사용하여 크롬드라이버 실행

```
from fake_useragent import UserAgent
from selenium import webdriver



fake_agent = UserAgent().random
print(fake_agent)

op = webdriver.ChromeOptions()
op.add_argument("user-agent={fake}".format(fake = fake_agent))

url = 'https://www.facebook.com/'

dr = webdriver.Chrome('/Users/joono/chromedriver',chrome_options=op)

dr.get(url)
```

API 활용 학습의 목적.

실제로 로그인이 자동으로 되는 크롬처럼 크롬드라이버에서도 로그인을 하기 위해 cookie를 사용합니다.

API 활용 학습 - 공공데이터 포털 API

가. 공공데이터포털 OpenAPI 사용해보기

공공데이터포털이란 정부 각 부처에서 수집하고 관리하는 데이터를 국민들에게 제공해주는 웹사이트입니다.

공공데이터 포털에서는 엑셀파일로도 데이터를 제공하지만, API를 통해 원하는 정보를 얻을 수 있게도 해놓았습니다.

requests를 학습할 때 사용해봤던 API를 사용하기 위해 공공데이터 포털에서 로그인한 후,
https://www.data.go.kr/dataset/15012005/openapi.do
위 URL에서 활용신청을 하시면 KEY를 받을 수 있습니다.

API 활용 학습 - 공공데이터 포털 API

가. 공공데이터포털 OpenAPI 사용해보기

API를 사용할 때 가장 중요한 것은 우선 GET인지 POST인지를 살펴보는 것입니다.
GET방식이라면 우리가 요청하는 정보를 필터링할 때 파라미터, 즉 URL을 바꿔서 설정하는 것이고, POST 방식이라면 data body에 넣는 정보를 바꿔서 필터링을 합니다.

예제로 사용해볼 상권정보 조회는 GET방식을 사용하고 있습니다.

또한 첨부해드린 공공데이터 가이드 파일.pdf에 45페이지를 보면 어떤 파라미터가 들어가는지가 나와있습니다.

항목명(영문)	항목명(국문)	항목크기	항목구분	샘플데이터	항목설명
divId	구분ID		1	ctprvnCd	시도는 ctprvnCd, 시군구는 signguCd, 행정동은 adongCd를 사용
key	행정구역코드		1	11	시도는 시도코드값, 시군구는 시군구코드값, 행정동은 행정동코드값을 사용
indsLclsCd	상권업종 대분류코드	1	0	Q	입력된 대분류 업종에 해당하는 것만 조회
indsMclsCd	상권업종 중분류코드	3	0	Q12	입력된 중분류 업종에 해당하는 것만 조회
indsSclsCd	상권업종 소분류코드	6	0	Q12A01	입력된 소분류 업종에 해당하는 것만 조회
numOfRows	페이지당 건수		0	100	최대 1000
pageNo	페이지 번호		0	1	현재 요청 페이지번호
type	타입		0	xml	xml / json

출처 : 공공데이터포털

API 활용 학습 - 공공데이터 포털 API

우리는 행정동을 사용할 것이기 때문에
divid는 adongCd로, key는 행정구역코드.xlsx에서 찾아서 넣을 것입니다.

또한 모든 업종을 모을 것이기 때문에 우선은 업종은 넣지 않고, 페이지당 1000건씩을 모으고, 결과는 json으로 받을 것입니다. 정리하자면 아래와 같은 변수를 만들 수 있습니다.

```
divid = 'adongCd'
key = '1168064000'
dtype = 'json'
servicekey =
'XGjk3lw3HWVPQUnt%2FbK0ps7Oun48a%2FVgEb0n1PnbqqERTRXj2wwdybeabsJBVinRb2braJKLp9y
0jGc8RpQAdw%3D%3D'
num = '1000'
page = '1'
```

Fast campus

API 활용 학습 - 공공데이터 포털 API

가. 공공데이터포털 OpenAPI 사용해보기

행정구역코드.xlsx 파일을 보시면 아래 그림과 같이 동별로 행정기관코드를 얻을 수 있습니다.

행정기관코드를 동 key값에 넣으시면 원하는 동의 상권정보를 얻을 수 있습니다.

C	D	E	F	G
행정구역명	행정동	법정동	행정구역분류	행정기관코드
서울특별시	서울특별시	서울특별시	11	1100000000
종로구	종로구	종로구	11010	1111000000
청운효자동	청운효자동	청운동	1101072	1111051500
청운효자동	청운효자동	신교동	1101072	1111051500
청운효자동	청운효자동	궁정동	1101072	1111051500
청운효자동	청운효자동	효자동	1101072	1111051500
청운효자동	청운효자동	창성동	1101072	1111051500

출처 : 공공데이터포털

Fast campus

API 활용 학습 - 공공데이터 포털 API

가. 공공데이터포털 OpenAPI 사용해보기

위 요청에 대한 response는 아래와 같습니다.


```
{
 "header" : {
  "description" : "소상공인시장진흥공단 행정동별 상가업소정보"
  ,"columns" : ["상가업소번호","상호명","지점명","상권업종대분류코드","상권업종대분류명","상권업종중분류코드","상권업종중분류명","상권업종소분류코드","상권업종소분류명","표준산업분류코드","표준산
 ","행정동코드","행정동명","법정동코드","법정동명","PNU코드","대지구분코드","대지구분명","지번본번지","지번부번지","지번주소","도로명코드","도로명","건물본번지","건물부번지","건물관리번호","건
 ","층정보","호정보","경도","위도"]
  ,"resultCode" : "00"
  ,"resultMsg" : "NORMAL SERVICE"
 },
 "body" : {
   "items" : [
    {
    "bizesId" : "4267333"
    ,"bizesNm" : "더클럽객석2층"
    ,"brchNm" : ""
    ,"indsLclsCd" : "Q"
    ,"indsLclsNm" : "음식"
    ,"indsMclsCd" : "Q07"
```


살펴보면 header는 response에 대한 속성정보를 주고 있고, 우리가 원하는 정보는 body 안에 들어있습니다.

json은 딕셔너리와 같이 ['원하는 부분']으로 접근 가능합니다.우리는 body에 있는 items를 가져와야 하니 response를 우선 json변수로 만들고 나서 접근합니다. response.json()을 사용하면 json 변수가 만들어 집니다.

우리가 원하는 정보들의 리스트는 response.json()['body']['items']를 통해 구할 수 있습니다.

API 활용 학습 - 공공데이터 포털 API

가. 공공데이터포털 OpenAPI 사용해보기


```
"items" : [
 {
 "bizesId" : "4267333"
 ,"bizesNm" : "더클럽객석2층"
 ,"brchNm" : ""
 ,"indsLclsCd" : "Q"
 ,"indsLclsNm" : "음식"
 ,"indsMclsCd" : "Q07"
 ,"indsMclsNm" : "패스트푸드"
 ,"indsSclsCd" : "Q07A04"
 ,"indsSclsNm" : "패스트푸드"
 ,"ksicCd" : "I56199"
 ,"ksicNm" : "그외 기타 음식점업"
 ,"ctprvnCd" : "11"
 ,"ctprvnNm" : "서울특별시"
 ,"signguCd" : "11680"
 ,"signguNm" : "강남구"
 ,"adongCd" : "1168064000"
 ,"adongNm" : "역삼1동"
 ,"ldongCd" : "1168010100"
 ,"ldongNm" : "역삼동"
 ,"lnoCd" : "1168010100206790000"
 ,"plotSctCd" : "1"
 ,"plotSctNm" : "대지"
 ,"lnoMnno" : 679
 ,"lnoSlno" : ""
 ,"lnoAdr" : "서울특별시 강남구 역삼동 679"
 ,"rdnmCd" : "116803121022"
 ,"rdnm" : "서울특별시 강남구 논현로"
 "bldMnno" : 508
```


items 안을 살펴보면 [] 안에 여러개의 딕셔너리가 존재하는 것을 볼 수 있습니다. 여기서 우리가 원하는 정보인 상호명은 bizesNm이라는 key로 가져올 수 있습니다.

그래서

```
for item in items:
    print(item['bizesNm'])
```

을 통해 각각의 상호명을 뽑아낼 수 있습니다.

API 활용 학습 - Youtube 댓글 API

나. Youtube API 사용해보기

https://developers.google.com/youtube/v3/docs/commentThreads?hl=ko

Youtube 영상의 댓글을 수집하는 API입니다.

공공데이터포털의 API와는 다르게, developers라는 곳에서 API를 제공합니다.

대부분의 회사는 자사의 서비스에서 정보를 요청할 수 있는 API를 developers를 통해 제공합니다.

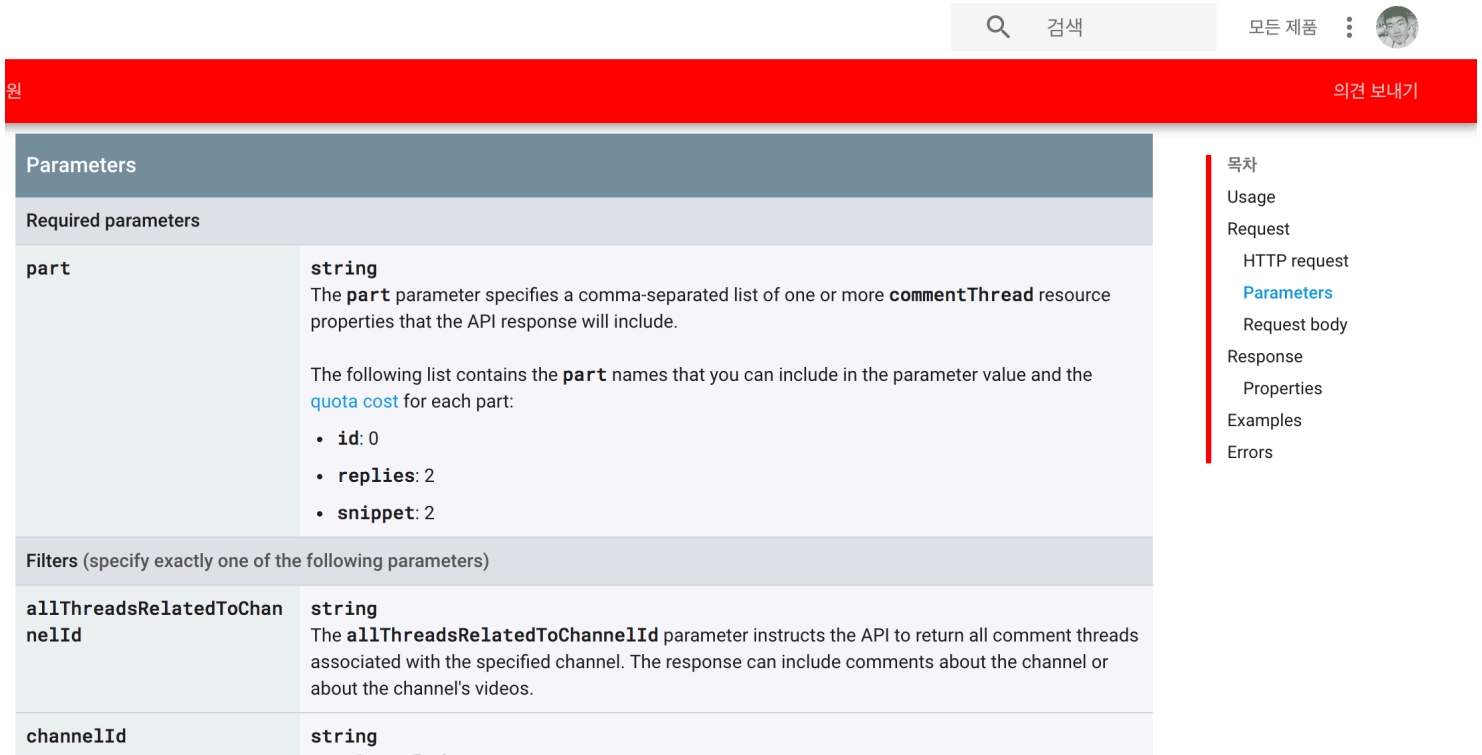
한글로는 'OOO 개발자센터'를 검색하시면 대부분 접근 할 수 있습니다.

이러한 개발자 센터에서는 로그인을 하면 요청을 할 수 있는 KEY 혹은 Token을 제공하며, 모든 요청에는
이러한 KEY가 포함되어야 합니다.

Youtube 또한 Key를 발급받아 요청을 진행합니다.

Fast campus

API 활용 학습 - Youtube 댓글 API



대부분의 API 사이트에는 그림의 오른쪽 목차에 있는 내용들을 표시해 놓습니다.

요청에 들어가는 Parameter들과, 요청에 대한 응답이 어떤 형식으로 오는지, 그리고 각 언어로 구성된 예제 코드도 제공하는 곳이 많습니다.

감사합니다.