

Python 크롤러 기본

키워드 : [robots.txt](#)

https://ko.wikipedia.org/wiki/%EB%A1%9C%EB%B4%87_%EB%B0%B0%EC%A0%9C_%ED%91%9C%EC%A
[웹 크롤러 \(https://ko.wikipedia.org/wiki/%EC%9B%B9_%ED%81%AC%EB%A1%A4%EB%9F%AC\)](https://ko.wikipedia.org/wiki/%EC%9B%B9_%ED%81%AC%EB%A1%A4%EB%9F%AC),
[BeautifulSoup \(https://www.crummy.com/software/BeautifulSoup/bs4/doc/\)](https://www.crummy.com/software/BeautifulSoup/bs4/doc/),
[requests \(http://docs.python-requests.org/en/master/\)](http://docs.python-requests.org/en/master/),
[HTTP status code](#)
https://ko.wikipedia.org/wiki/HTTP_%EC%83%81%ED%83%9C_%EC%BD%94%EB%93%9C

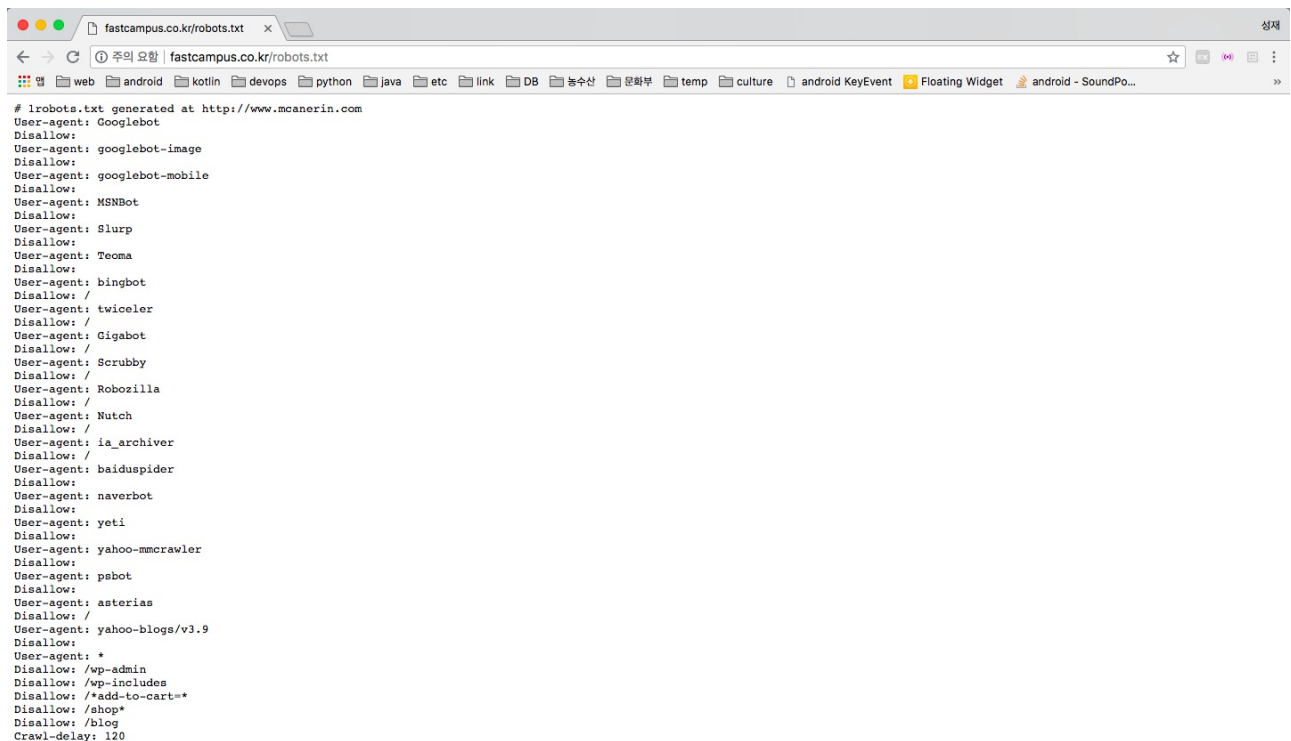
우리의 강의 목표는 웹 클라이언트 개발 기초에 대한 공부와 파이썬 크롤러 개발의 관한 내용을 다루는 것이었습니다. 이번 장에서부터 마지막 장까지는 우리의 최종 목표인 크롤링과 관련된 내용을 정리해보도록 하겠습니다.

robots.txt

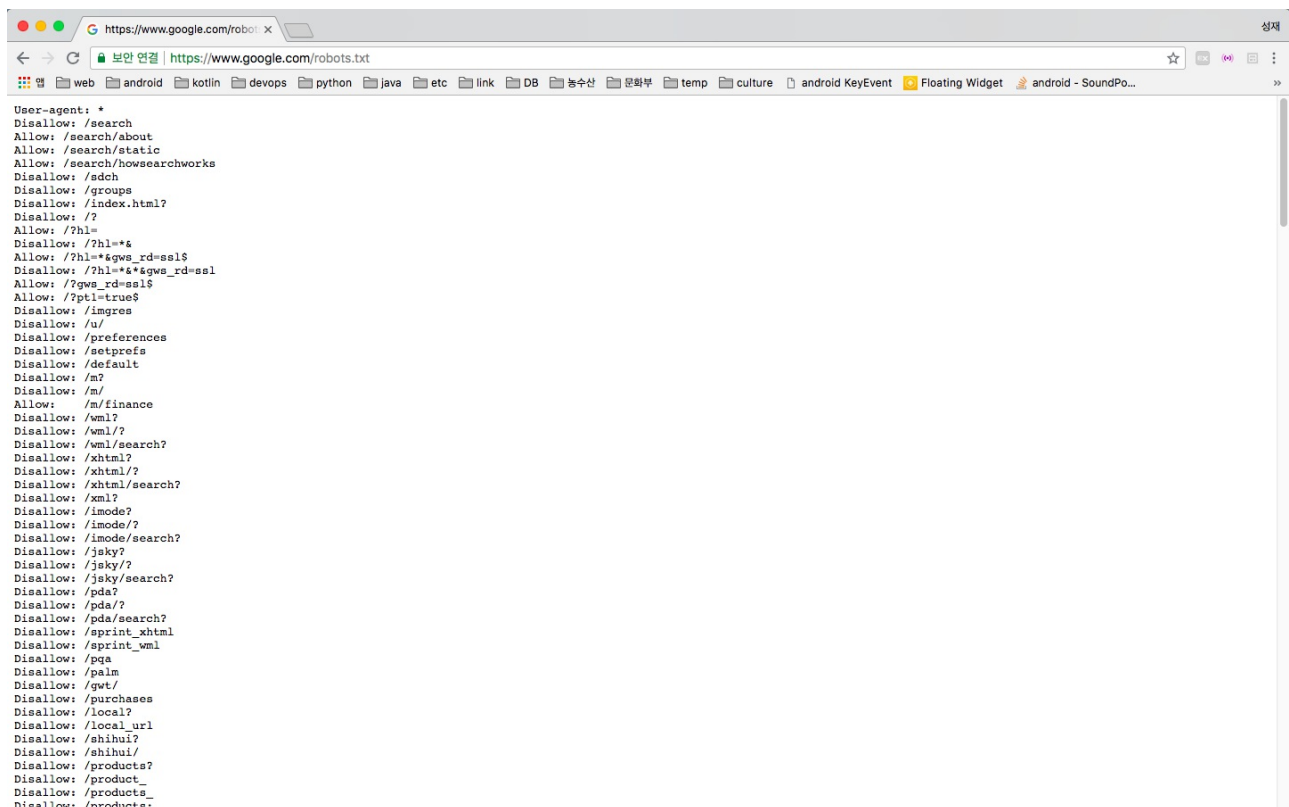
우선, 검색 엔진과 robots.txt에 대해서 이야기하고 시작하도록 하겠습니다. 강의에서도 잠시 언급했듯이, 우리가 보는 포털 사이트와 여러 검색 엔진은 우리가 특정 키워드를 검색하면, 해당 검색어에 해당하는 웹 페이지를 미리 정의된 알고리즘에 따라서 '검색 봇'(Search Bot)이 웹 페이지를 찾고 우리에게 보여주게 됩니다.

웹 사이트마다 검색 봇에게 잘 노출되길 희망하는 부분도 있겠지만, 특정 페이지나 디렉터리는 검색 노출이 되지 않길 희망할 수도 있습니다. 이번 절에서 살펴볼 robots.txt는 검색 봇에게 노출 허용, 노출 거부 등을 설정할 수 있는 텍스트 파일이라고 할 수 있습니다.

우리 말로는 '로봇 배제 표준'이라는 명칭으로 불리며, 검색 봇이 접근하는 것을 방지하기 위한 국제규약입니다.



```
# robots.txt generated at http://www.mcanerin.com
User-agent: Googlebot
Disallow:
User-agent: googlebot-image
Disallow:
User-agent: googlebot-mobile
Disallow:
User-agent: MSNBot
Disallow:
User-agent: Slurp
Disallow:
User-agent: Teoma
Disallow:
User-agent: bingbot
Disallow: /
User-agent: twiceler
Disallow: /
User-agent: Gigabot
Disallow: /
User-agent: Scrubby
Disallow: /
User-agent: Robozilla
Disallow: /
User-agent: Nutch
Disallow: /
User-agent: ia_archiver
Disallow: /
User-agent: baiduspider
Disallow:
User-agent: naverbot
Disallow:
User-agent: yeti
Disallow:
User-agent: yahoo-mmecrawler
Disallow:
User-agent: pabot
Disallow:
User-agent: asterias
Disallow: /
User-agent: yahoo-blogs/v3.9
Disallow:
User-agent: *
Disallow: /wp-admin
Disallow: /wp-includes
Disallow: /*add-to-cart=*
Disallow: /shop*
Disallow: /blog
Crawl-delay: 120
```



위 사진은 패스트캠퍼스와 구글의 robots.txt를 나타냅니다. 위 사진에 나와있는 것 처럼, 최상위 도메인 주소(root)에 /robots.txt를 입력하면, 로봇 배제 표준을 확인할 수 있습니다.(e.g., <https://www.google.com/robots.txt>)

국내 포털을 비롯한, 대부분의 웹 사이트가 이 로봇 배제 표준을 가지고 있습니다. robots.txt의 의미는 [여기](https://ko.wikipedia.org/wiki/%EB%A1%9C%EB%B4%87_%EB%B0%B0%EC%A0%9C_%ED%91%9C%EC%A) (https://ko.wikipedia.org/wiki/%EB%A1%9C%EB%B4%87_%EB%B0%B0%EC%A0%9C_%ED%91%9C%EC%A)를 참고해주세요.

이 규약은 권고안이며, 로봇이 robots.txt파일을 읽고 접근을 중지하는 것을 목적으로 합니다. 따라서 접근방지 내용을 작성하였다고 해도, 다른 사람들이 그 파일에 접근할 수 있습니다. 반대로 말하면, 위 권고안을 지키지 않을 경우엔 웹 사이트에서 접근 IP를 강제 차단할 수도 있습니다.

또한, robots.txt가 구성되어 있지 않거나 구체적으로 명시되어 있지 않다고 하더라도, 고유한 저작권이 없는 것은 아닙니다. 크롤링을 진행할 때, 가능하다면 최대한 robots.txt의 명시된 내용을 존중해주는 것이 좋습니다.

결론적으로, 공격적이고 악의적인 목적의 크롤링은 무조건적으로 **지양**되어야 한다는 점 말씀드리고 싶습니다.

HTTP 상태 코드

강의에서 살펴봤던 HTTP 상태(또는 응답) 코드에 대해서 잠시 이야기하고 진행하도록 하겠습니다. 우리가 프로그램을 통해 서버에게 웹 자원을 요청하면, 서버는 우리에게 웹 자원과 함께 여러 가지 정보를 함께 제공합니다.

예를 들어, 클라이언트의 요청에 서버가 잘 반응했는지, 반응하지 못했는지, 만약 반응하지 못했다면 왜 반응하지 못했는지 등에 대한 정보를 1xx ~ 5xx까지의 숫자로 나타내 우리에게 전송해줍니다. 이 정보가 바로 HTTP 상태 코드라고 할 수 있습니다.

한 번쯤 접해보셨을 상태 코드는 아마도, **404 Not Found**일 것입니다. 이 상태 코드는 웹과 관련해서 필수적인 지식입니다. 안타깝게도 이 상태 코드의 종류는 굉장히 여러 가지가 있습니다. 외우실 필요는 없습니다. 찾아보면 금방 나오기도 하며, 자연스럽게 응답 코드가 나타내는 의미에 익숙해지실 겁니다.

강의에서도 말씀드렸듯이, 지금 당장은 한 가지만 기억해주시면 됩니다. 서버에 웹 자원 요청 시 돌아온 상태 코드가 200 이라면, 정상적으로 서버가 우리의 요청을 받고, 응답한 것이라는 것을 말입니다! :)

더 자세한 HTTP 상태 코드와 관련된 내용은 [여기](#)

(https://ko.wikipedia.org/wiki/HTTP_%EC%83%81%ED%83%9C_%EC%BD%94%EB%93%9C)를 참고해주세요.

requests

웹 자원을 요청하기 위해서 사용하는 패키지로 requests를 소개했습니다. requests 패키지는 굉장히 다양한 기능들이 제공합니다. requests 패키지의 몇 가지 주요 내용에 대해서 살펴보도록 하겠습니다.

```
import requests

params = {'key1':1, 'key2':2}
r = requests.get('https://www.fastcampus.co.kr/dev_online_introdev/',
params=params) # HTTP GET 요청

print(r.encoding) # UTF-8
print(r.url) # https://www.fastcampus.co.kr/dev_online_introdev?key1=1&key2=2
print(r.text) # 응답 받은 콘텐츠를 해석할 수 있는 인코딩 형태로 만들어줍니다.
```

우선, 우리는 강의에서 requests 모듈의 get 메서드를 이용해서, 웹 사이트의 정보를 자원을 검색하고, 파이썬 코드로 핸들링 할 수 있다고 말씀드렸습니다. 웹 자원을 요청할 때, params라는 이름 있는 인자에 키:값 형태의 딕셔너리로 쿼리 문자열을 만들어 요청할 수 있습니다. 또한, requests 모듈의 경우 가져온 웹 자원의 인코딩 설정 등도 확인할 수 있습니다.

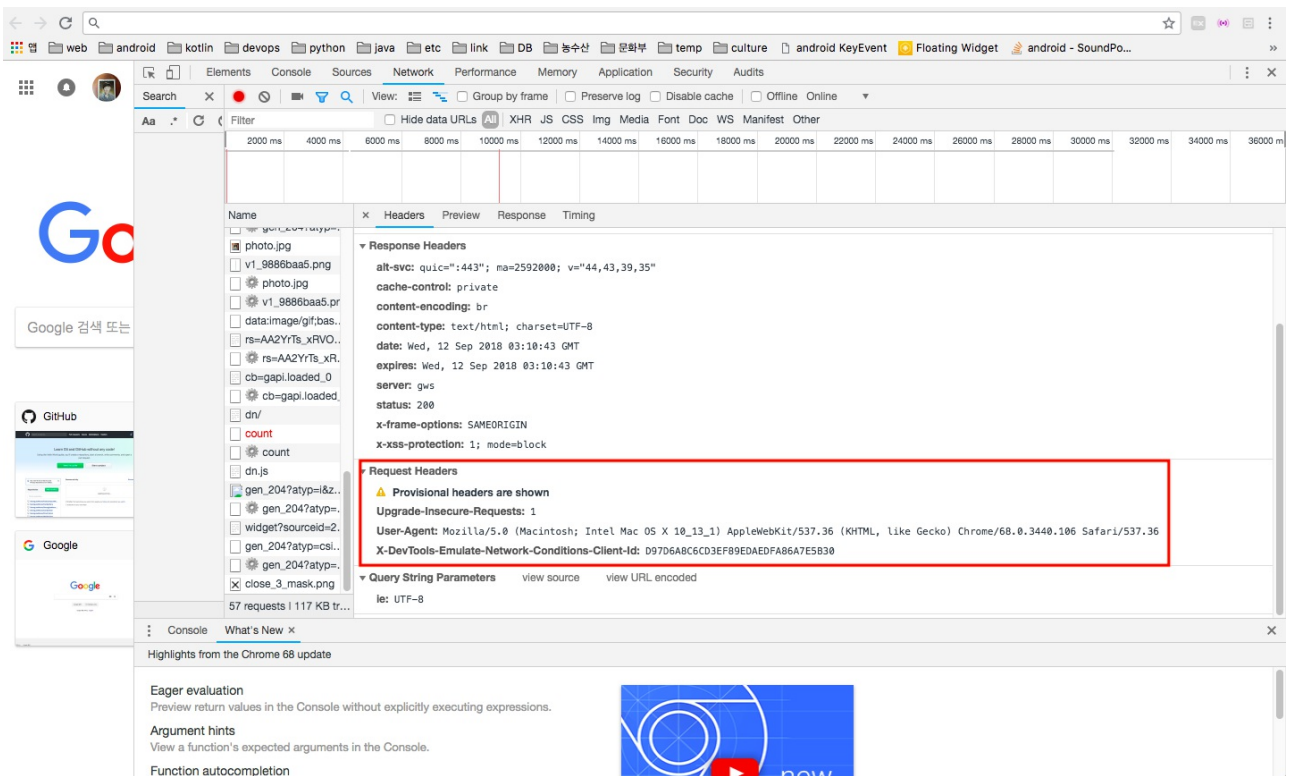
```
import requests

data = {'key1':1, 'key2':2}
r = requests.post('https://www.fastcampus.co.kr/dev_online_introdev/',
data=data)
```

우리가 웹 자원을 요청할 때는 get 방식 뿐만 아니라, post 방식으로도 요청할 수 있습니다. post 메서드는 data라는 이름있는 인자에 get 메서드와 마찬가지로 키:값 형태의 딕셔너리로 name, value를 만들어 웹 자원을 요청할 수 있습니다.

```
import requests
headers = {
'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36'
}
r = requests.get('https://www.fastcampus.co.kr/dev_online_introdev/',
headers=headers)
```

웹 자원을 요청할 때, 헤더 정보도 함께 담아서 요청할 수 있는데요, HTTP에 대해서 이야기할 때, 헤더에 대해서 잠시 언급한 적이 있습니다. 우선, 개발자 도구를 통해 구글 사이트의 요청 헤더 정보를 확인해보도록 하겠습니다.



Network 탭의 가장 상단부 URL을 클릭합니다. 우리는 요청하는 측이기 때문에, Request Headers 부분을 봐주시면 됩니다. 이 헤더 정보에는 User-Agent라는 기기 정보와 브라우저 정보를 담는 부분이 있습니다.

우리가 크롤링을 통해 웹 자원을 요청하면, 기존의 브라우저가 아닌 프로그램 코드를 통해서 요청하는 것이 됩니다. 몇몇 웹 사이트는 이 부분을 해석하여 정상적인 브라우저가 아니라 크롤링을 위한 독립적인 프로그램인 것을 확인하고, 걸러내는 경우가 있습니다. 이러한 경우에 headers라는 이름 있는 인자에 User-Agent 값을 마찬가지로 디크서너리 형태로 만들어 전송해줘야 합니다.

참고로, requests 모듈을 이용하여 https 프로토콜의 웹 자원을 요청할 시에는 SSLError라는 에러가 발생할 수 있습니다. 이런 경우 get 메서드의 인자로 verify=False라는 것을 추가로 입력해야 합니다. (e.g., requests.get(URL.format(str(page)), verify=False)

BeautifulSoup

요청한 웹 자원이 정상적으로 잘 돌아오면, 불러온 웹 자원을 우리가 분석하기 용이한 HTML 문서로 파싱(Parsing)하는 작업이 필요합니다. HTML 문서를 파싱하고 분석하기 위한 BeautifulSoup 패키지의 몇 가지 주요 내용에 대해서 살펴보도록 하겠습니다.

```
import requests
from bs4 import BeautifulSoup as bs

r = requests.get('https://www.fastcampus.co.kr/dev_online_introdev/')
soup = bs(r.text, 'html.parser')
```

웹 자원을 불러온 후, 불러온 웹 자원을 BeautifulSoup 객체의 첫 번째 인자로 r.text, 두 번째 인자로 html.parser를 입력 값으로 넘겨줍니다. 참고로, html뿐만 아니라, 우리가 공부하진 않았지만, xml, lxml 등 다양한 형식의 파일 구조를 파싱 할 수 있습니다.

```

import requests
from bs4 import BeautifulSoup as bs

r = requests.get('https://www.fastcampus.co.kr/dev_online_introdev/')
soup = bs(r.text, 'html.parser')

print(soup)

# 찾은 태그를 하나만 반환합니다.
soup.find('div')
soup.find(id = 'id')
soup.find('div', id='id')
soup.find('div', class_ = 'class')
# 찾은 태그를 리스트로 반환합니다.
soup.find_all('div', attrs={'id':'id'})
soup.find_all('div', attrs={'class':'class'})
soup.select('css selector')
# 속성에 해당하는 값을 반환합니다.
href = soup['href']
name = soup['name']
value = soup['value']
# 태그가 담고 있는 콘텐츠를 가져옵니다.
print(soup.get_text())

```

soup 객체를 print 함수로 터미널에 출력하면, 우리가 잘 아는 형태의 html 구조로 만들어진 것을 확인할 수 있습니다. 이제 이 html 형식의 soup 객체의 다양한 메서드를 통해서 특정 태그의 정보만 추출해서 가져올 수 있습니다. 주의해야 할 점은, find 메서드의 경우 하나의 BeautifulSoup 객체를 반환하고, find_all과 select 메서드의 경우엔 리스트로 BeautifulSoup 객체를 반환합니다.

위에서 모두 다루지 못한 requests 패키지에 대한 더 자세한 설명은 [여기 \(http://docs.python-requests.org/en/master/user/quickstart/#make-a-request\)](http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)를 참고해주시고, BeautifulSoup와 관련된 더 자세한 설명은 [여기 \(https://www.crummy.com/software/BeautifulSoup/bs4/doc/\)](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)를 참고 부탁드립니다.

이번 장에서는 웹 자원 요청에 대해서 알아보았습니다. 세상에는 수많은 웹 사이트가 있습니다. 각각의 웹 사이트마다 서로 다른 특징을 가지고 있으며, 한 가지 방법으로 모든 웹 사이트의 자원과 정보를 가지고 올 순 없습니다. 하지만, 코드를 구성하는 기본적인 로직 자체는 큰 변화가 없습니다. 전체적인 틀에 대한 이해가 있다면, 약간의 검색과 몇 번의 시도로 금세 해결할 수 있습니다. :)