# Ask Django

## EP04. BeautifulSoup4 라이브러리 살펴보기

# HTTP 응답

- 웹서버에서는 일반적으로 **HTML**, CSS, JavaScript, Image 형식의 응답을 합니다.
- HTML 문서는 중첩된 태그로 구성된 계층적인 구조입니다.

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>AskDjango</title>
    </head>
    <body>
        <h1>AskDjango VOD</h1>
        <ul id="vod_list">
            <li class="vod">파이썬 차근차근 시작하기</li>
            <li class="vod">장고 기본편</li>
        </ul>
        <hr/>
        &copy; 2017 AskDjango
    </body>
</html>
```

# DOM 문서 MDN #doc

## The Document Object Model

- 브라우저는 HTML문자열을 DOM Tree로 변환하여, 문서를 표현 [1]

```
<!-- 서버로부터 아래 응답을 받는다면 (우리는 이 부분에 집중!!!) -->

<table>
    <tr>
        <td>테이블 컬럼</td>
    </tr>
</table>

<!-- 브라우저를 이를 DOM Tree로 다음과 같이 변환 (이때 브라우저 나름의 해석이 들어갑니다.) -->

<table>
    <tbody>
        <tr>
            <td>테이블 컬럼</td>
        </tr>
    </tbody>
</table>
```

---

[1] requests를 통한 응답에서는 HTML은 "페이지 소스보기"를 참고하셔야 합니다. 개발자도구에서의 내역은 브라우저의 DOM Tree내역입니다.

# 복잡한 문자열에서 특정 문자열 정보를 가져올려면?

- 방법1: 정규 표현식을 활용
  - 가장 빠른 처리가 가능하나, 정규표현식 Rule을 만드는 것이 많이 번거롭고 복잡합니다.
  - 때에 따라 필요할 수도 있습니다.
- 방법2: HTML Parser 라이브러리를 활용
  - DOM Tree을 탐색하는 방식으로 적용이 쉽습니다.
  - ex) BeautifulSoup4, lxml

# BeautifulSoup4 #doc

- HTML/XML Parser : HTML/XML문자열에서 원하는 태그정보를 뽑아냅니다.
- 설치 : pip3 install beautifulsoup4
  - 주의: `pip3 install beautifulsoup` 명령은 버전3가 설치됩니다.

```
from bs4 import BeautifulSoup

html = '''
<ol>
    <li>NEVER - 국민의 아들</li>
    <li>SIGNAL - TWICE</li>
    <li>LONELY - 씨스타</li>
    <li>I LUV IT - PSY</li>
    <li>New Face - PSY</li>
</ol>
'''

soup = BeautifulSoup(html, 'html.parser')
for tag in soup.select('li'):
    print(tag.text)
```

# Parser / Python's html.parser

- BeautifulSoup4 내장 파서
- 샘플코드

```
soup = BeautifulSoup(파싱할문자열, 'html.parser')
```

# Parser / lxml's HTML parser

- lxml HTML 파서 사용 (외부 C 라이브러리)
  - html.parser보다 좀 더 유연하고, 빠른 처리
- 설치 : `pip3 install lxml`
- 샘플코드

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

# 태그를 찾는 2가지 방법

1. find를 통해 태그 하나씩 찾기
2. 태그 관계를 지정하여 찾기 (CSS Selector 사용)

# find를 통해 태그 하나씩 찾아가기

**예시) 멜론 TOP100 차트**

```
import requests
from bs4 import BeautifulSoup

html = requests.get('http://www.melon.com/chart/index.htm').text
soup = BeautifulSoup(html, 'html.parser')

tag_list = []
for tr_tag in soup.find(id='chartListObj').find_all('tr'):
    tag = tr_tag.find(class_='wrap_song_info')
    if tag:
        tag_sub_list = tag.find_all(href=lambda value: (value and 'playSong' in value))
        tag_list.extend(tag_sub_list)

for idx, tag in enumerate(tag_list, 1):
    print(idx, tag.text)
```

# 태그 관계를 지정하여 찾기 (CSS Selector 사용)

- CSS Selector를 통한 Tag 찾기 지원
  - tag name : "tag_name"
  - tag id : "#tag_id"
  - tag class names : ".tag_class"

# CSS Selector Syntax

- * : 모든 Tag
- tag : 해당 모든 Tag
- Tag1 **>** Tag2 : Tag1 의 직계인 모든 Tag2
- Tag1 Tag2 : Tag1 의 자손인 모든 Tag2 (직계임이 요구되지 않음)
- Tag1, Tag2 : Tag1**이거나** Tag2인 모든 Tag
- tag**[attr]** : attr속성이 정의된 모든 Tag
- tag[attr**=**"bar"] : attr속성이 "bar"문자열과 일치하는 모든 Tag
- tag[attr***=**"bar"] : attr속성이 "bar"문자열과 부분 매칭되는 모든 Tag
- tag[attr**^=**"bar"] : attr속성이 "bar"문자열로 시작하는 모든 Tag
- tag[attr**$=**"bar"] : attr속성이 "bar"문자열로 끝나는 모든 Tag

- tag**#tag_id** : id가 **tag_id**인 모든 Tag
- tag**.tag_class** : 클래스명 중에 **tag_class**가 포함된 모든 Tag
- tag**#tag_id.tag_cls1.tag_cls2**
  - id가 **tag_id** 이고, 클래스명 중에 **tag_cls1**와 **tag_cls2**가 모두 포함된 모든 Tag
- tag**.tag_cls1.tag_cls2**
  - 클래스명 중에 **tag_cls1**와 **tag_cls2**가 모든 포함된 모든 Tag
- tag**.tag_cls1 .tag_cls2**
  - 클래스명 중에 **tag_cls1**이 포함된 Tag의 자식 중에 (직계가 아니어도 OK), 클래스명에 **tag_cls2**가 포함된 모든 Tag

# 예시) 멜론 TOP100 차트

**http://www.melon.com/chart/index.htm**

```
import requests
from bs4 import BeautifulSoup

html = requests.get('http://www.melon.com/chart/index.htm').text
soup = BeautifulSoup(html, 'html.parser')

tag_list = soup.select('#chartListObj tr .wrap_song_info a[href*=playSong]')
for idx, tag in enumerate(tag_list, 1):
    print(idx, tag.text)
```

# (주의) CSS Selector를 지정하실 때에는

- 패턴을 너무 타이트하게 지정하시면, HTML 마크업이 조금만 변경되어도 태그를 찾을 수 없게 됩니다.
- 적절히 최소한의 패턴을 타이트하게 지정해주세요.

이는 다양한 연습을 통해, 감을 익힐 수 밖에 없습니다.

# 예시) Google Finance

**html.parser보다 lxml Parser가 보다 유연하게 파싱해줍니다.**

```
<div id=prices class="gf-table-wrapper sfe-break-bottom-16">
<table class="gf-table historical_price">
    <tr class=bb>
        <th class="bb lm lft">Date
        <th class="rgt bb">Open
        <th class="rgt bb">High
        <th class="rgt bb">Low
        <th class="rgt bb">Close
        <th class="rgt bb rm">Volume
    <tr>
        <td class="lm">Feb 28, 2014
        <td class="rgt">100.71
        <td class="rgt">100.71
        <td class="rgt">100.71
        <td class="rgt">100.71
        <td class="rgt rm">0
</table>
```

- tr/th/td 닫는 태그가 없습니다. :-(
  - html.parser로는 각 태그를 구분할 수 없습니다. 하나의 큰 태그로 인식
  - **lxml**이 보다 유연하게 처리해줍니다.

```
import requests
from bs4 import BeautifulSoup

params = {
    'q': 'EPA:BRNTB',
    'startdate': 'Jan 01, 2016',
    'enddate': 'Jun 02, 2016',
}
html = requests.get('https://www.google.com/finance/historical', params=params).text
soup = BeautifulSoup(html, 'lxml') #'html.parser')

for tr_tag in soup.select('#prices > table > tr'):
    row = [td_tag.text.strip() for td_tag in tr_tag.select('th, td')]
    print(row)
```

# 연습문제

**http://www.reddit.com 내 각 reddit의 링크명과 링크URL을 출력하세요.**

- 힌트
  - requests라이브러리로 위 주소에 GET요청을 하여 HTML응답을 받아냅니다.
  - 이때 User-Agent헤더를 변경하지않으면, 4XX응답을 받습니다.