Web Scraping – 기본편

이상엽 (연세대학교)

Web Scraping 은 Web(인터넷)에 존재하는 다양한 형태의 정보와 데이터를 수집하는 것을 말한다. 한국에서는 Web Crawling 이라고 더 많이 알려져 있다. 하지만, 명확하게 구분하자면 Web Scraping 은 Web Crawling 과 같은 의미는 아니다. Web Crawling 은 여러 개의 웹페이지를 이동한다 (Crawl)는 뜻이 더 큰 반면, Scraping 은 특정 페이지의 정보/데이터를 수집한다는 뜻이 더 크다. 대부분의 경우에 두 표현은 같은 의미로 사용된다. 본 수업에서는 Web Scraping 이라는 표현을 쓰기로 한다.

Web communication 이해

우리가 만들고자 하는 Web scraping 프로그램을 만들기 위해서는 일단 컴퓨터 간의 communication 이 어떻게 이뤄지는지를 이해할 필요가 있다. 예를 들어서 우리가 컴퓨터를 이용해서 인터넷에 연결을 하고 특정한 웹페이지 (예, www.daum.net)에 접속한다고 가정을 해보도록 하자. 컴퓨터가 켜져 있고 인터넷에 연결되어 있는 상황에서, 사용자가 특정한 웹 페이지 (즉, www.daum.net)에 접속하기 위해서 가장 먼저하는 것은 웹 브라우저를 실행시키는 것이다. 주로 사용되는 웹 브라우저로는 Chrome, Firefox, Internet Explorer 등이 있다. 브라우저를 실행시킨 후에 우리는 해당 페이지에 접속하기 위해 브라우저 주소창 (address bar)에 접속하고자 하는 페이지의 주소 (URL, uniform resource locator 주소라고 함)를 입력하게 된다. 다음(Daum) 페이지에 접속하고자 하는 경우에는 www.daum.net 의 주소를 브라우저의 주소창에 입력하게 된다. 그러면 해당 페이지의 내용이 브라우저 상에 나타나게 된다. 주소를 입력하고 화면에 우리가 원하는 페이지가 표시되기까지 걸리는 시간은 굉장히 짧다. 이 짧은 시간에 브라우저에서는 어떠한 일이 벌어져서 우리가 원하는 웹 페이지가 브라우저 상에 표시가 되는 것일까? 브라우저는 해당 주소의 웹페이지를 화면에 표시하기 위해 웹페이지를 화면에 표현하는데 필요한 데이터를 데이터가 저장되어 있는 컴퓨터 (서버라고 불림)에서 다운로드 받고, 이를 정해진 규칙에 의해서 브라우저 화면에 표시하게 된다. 브라우저의 역할을 좀 더 구체적으로 살펴보도록 하자.

브라우저의 역할

1) URL 주소를 IP 주소로 변환하기

사용자가 접속하고자 하는 웹 페이지의 주소 (즉, url 주소)를 입력 받은 브라우저는 DNS (domain name service) 서버라고 하는 특정한 컴퓨터에 접속해서해당 URL 주소를 컴퓨터가 이해할 수 있는 주소인 IP (Internet protocol) 주소로 변환을 한다. 각각의 URL 주소는 고유한 IP 주소를 갖는다. IP 주소는 숫자로 구성되어 있는데 보통 165.132.12.11 등으로 표현이 된다. 이러한 IP 주소는 하나의 컴퓨터가 인터넷 상에서 지니는 고유한 번호 혹은 주소라고 생각할 수 있다. 사람이 사용하는 웹페이지의 주소 즉, url 주소는 사람들이 이해할 수 있는 주소이고 이에 해당하는 컴퓨터가 이해할 수 있는 주소가 IP 주소인 것이다. DNS 서버는 거의 모든 웹페이지 고유의 IP 주소 정보

저장하고 있다. 우리가 사용하는 브라우저는 DNS 서버에 접속하여 사용자로부터 입력 받은 특정한 웹페이지에 해당하는 url 주소를 그 웹페이지가 저장되어 있는 컴퓨터의 IP 주소로 변환하게 되는 것이다.

2) IP 주소를 사용해서 해당 컴퓨터 (서버)에 접속하기

사용자가 입력한 URL 주소에 해당하는 웹페이지를 화면에 표시하기 위해서 브라우저는 해당 페이지를 표시하는데 필요한 원본 데이터가 있어야 한다. 이를 보통 source codes 라고 부른다. 이러한 원본 데이터는 해당 URL 주소에 해당하는 IP 주소를 갖는 컴퓨터 (흔히 서버라고 불리는 컴퓨터)에 저장되어 있다. DNS 서버를 통해서 URL 주소에 해당하는 IP 주소를 획득한 브라우저는 이 주소 정보를 이용해서 해당 IP 주소를 갖는 컴퓨터, 즉 서버, 에 접속하게 된다.

3) 웹페이지를 구성하는 원본 데이터 (source codes) 받기

접속한 다음, 사용자가 입력한 주소에 해당하는 웹 페이지를 브라우저에 표시하기 위해서 우리의 부라우저는 해당 페이지를 표시하는데 필요한 데이터를 서버 컴퓨터에게 요청하게 된다. 이를 위해 브라우저는 서버 컴퓨터에 request message 를 보낸다. 서버 컴퓨터는 request message 를 받은 후, message 를 보낸 브라우저가 아무런 문제가 없다는 것을 확인한 후에 (즉, hacking 의 목적으로 접속하는 것은 아닌지 등에 대한 문제), 아무런 문제가 없다라고 판단이 되면, 해당 request message 에 응답하게 된다. 이때 서버 컴퓨터는 사용자의 브라우저에게 response message 를 보내게 되는데, 이 response message 는 기본적으로 사용자가 접속하고자하는 웹페이지를 표시하는데 필요한 데이터(source codes)가 들어 있고, 추가적으로 해당 서버와 브라우저가 커뮤니케이션 하는데 필요한 정보가 포함되어 있다.

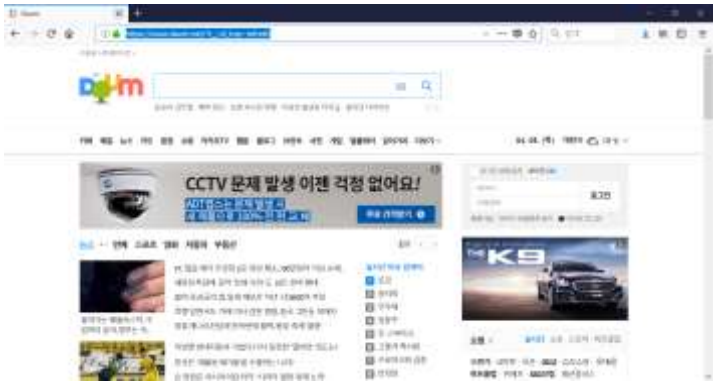
4) 받은 source codes 를 이용해서 브라우저 상에 표시하기

최종적으로 브라우저는 서버로부터 다운로드 받은 source codes 를 가지고 사전에 정해진 규칙에 따라서 사용자가 보기 편한 방식으로 화면에 해당 웹페이지를 표시하게 된다.

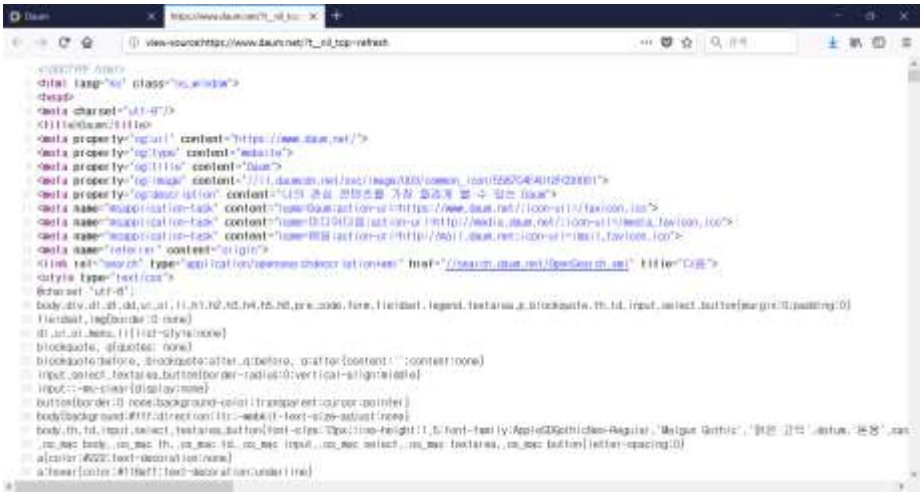
이러한 과정을 거쳐서 사용자는 접속하고자 하는 웹페이지의 URL 주소를 입력한 다음, 해당 페이지를 브라우저 상에서 볼 수가 있는 것이다.

웹 페이지를 구성하는 데이터(source codes)의 구조

예를 들어, 사용자가 접속하고자 하는 웹페이지가 www.daum.net 이라고 하자. 브라우저의 주소창에 www.daum.net 이라고 입력을 하게 되면, 사용자는 다음과 비슷한 화면을 보게 될 것이다.



이는 위에서 설명한 것 처럼 사용자의 브라우저가 사용자가 입력한 URL 주소를 1 차적으로 IP 주소로 변환하고 이를 이용해서 해당 페이지를 구성하는 source codes 를 저장하고 있는 서버에 접속해서 해당 source codes 를 받은 다음에, 그러한 source codes 를 정해진 규칙에 의해 화면에 user friendly 한 방식으로 표시한 것이다. 그렇다면, 하나의 웹페이지를 구성하는 source codes 는 어떠한 형태일까? source codes 는 아래와 같이 여러가지의 또 다른 컴퓨터 언어로 작성이 되어 있다.



주로 사용되는 언어로는 HTML, XML, Java Script 등이 있다. Web scraping 을 하는데 있어서는 source codes 가 어떠한 컴퓨터 언어를 사용해서 작성되었는지를 아는 것은 중요하지 않다. 즉, 어떠한 언어가 어떻게 작동하는지에 대해서 알필요가 없는 것이다. **여러분이 기억해야하는 것은 source codes 는 tag 라고 하는 것들로 이뤄져 있다는 것이다.** 예를 들어, 위의 www.daum.net 페이지를 구성하는 source codes 를 보면 아래와 같이 구성되어 있는 것을 확인할 수 있다.

```
<meta charset="utf-8"/>
<title>Daum</title>
<meta property="og:url" content="https://www.daum.net/">
<meta property="og:type" content="website">
<meta property="og:title" content="Daum">
<meta property="og:image"
content="//i1.daumcdn.net/svc/image/U03/common_icon/5587C4E4012FCD000
1">
<meta property="og:description" content="나의 관심 콘텐츠를 가장 즐겁게 볼
수 있는 Daum">
<meta name="msapplication-task" content="name=Daum;action-
uri=https://www.daum.net/;icon-uri=/favicon.ico">
<meta name="msapplication-task" content="name=미디어다음;action-
uri=http://media.daum.net/;icon-uri=/media_favicon.ico">
<meta name="msapplication-task" content="name=메일;action-
uri=http://mail.daum.net;icon-uri=/mail_favicon.ico">
<meta name="referrer" content="origin">
```

여기에서 <title>, <meta>와 같이 <> 괄호를 사용하여 표현된 것을 tag 라고 부른다. 보통 하나의 tag 는 하나의 시작 tag (start tag)와 하나의 끝 tag (end tag)의 pair 로 구성되어 있다. 위의 예에서 title tag 를 보면 <title> … </title>로 구성되어 있는 것을 볼 수 있는데, <title>이 시작 tag 이고, </title>이 끝 tag 이다. 그리고 보통의 경우 웹 브라우저 상에서 사용자에게 보이는 text 정보는 시작 tag 와 끝 tag 사이에 저장된다. title tag 를 보면 시작 tag 와 끝 tag 사이에 Daum 이라는 text 정보가 저장되어 있는 것을 알 수 있다. 이러한 text 정보가 우리가 웹에서 최종적으로 수집하고자 하는 데이터가 되는 것이다.

보통 하나의 tag 는 위에서 언급한 것 처럼 시작 tag 와 끝 tag 의 pair 로 구성이 되어 있는데, 항상 그런 것은 아니다. 시작 tag 만 존재하는 tag 들도 있다. 위의 예에서 meta 라는 tag 가 대표적인 예이다. 이러한 tag 는 필요한 정보를 <> 괄호안에 저장하게 된다.

Web scraping 프로그램의 역할

우리가 작성하는 Web scraping 프로그램은 크게 두가지 역할을 한다. 첫번째는 브라우저와 비슷한 역할인 우리가 추출하고자 하는 정보를 담고 있는 웹페이지의 source codes 를 저장하고 있는 서버와의 communication 이고, 두번째는 서버와의 communication 을 통해 다운로드 받은 source codes 에서 우리가 원하는 특정한 정보를 추출·저장하는 역할이다.

1) 서버와의 communication

우리가 추출하고자 하는 정보를 담고 있는 웹 페이지의 URL 주소를 이용해서 우리의 Web scraping 프로그램은 해당 서버에 접속하고 브라우저와 마찬가지로 request message 를 보내서 해당 페이지를 구성하는 source codes 를 받게 된다. Python 에서 서버와의 communication 을 위해서 사용되는 모듈은 여러가지 (urllib, urlib2, requests 등)가 있다. 그중에서 가장 흔하게 사용되는 것이 requests 이다. 본 tutorial 에서도 requests 모듈을 사용하도록 하겠다.

2) Source codes 로부터 원하는 정보 추출·저장

서버로부터 우리가 원하는 정보를 담고 있는 웹페이지의 source codes 를 다운로드 받은 다음에는 source codes 에서 우리가 원하는 정보만을 추출해야 한다. 특정 웹페이지를 구성하는 source codes 에는 해당 웹페이지를 브라우저에 표시하는데 필요한 수많은 정보를 담고 있다. 하지만, 우리는 이 중에서 극히 일부의 정보만을 필요로 한다. 우리가 원하는 정보는 보통 text 형태 (string type)로 저장되어 있는데, 많은 경우에 특정 tag 에 저장되어 있다. 즉, 우리가 web scraping 프로그램을 통해서 해야되는 작업은 source codes 에서 우리가 원하는 text 정보를 담고 있는 특정 tag 를 찾은 다음에 해당 tag 에서 우리가 원하는 정보를 추출하고, 이렇게 추출된 text 정보를 별도의 변수나 혹은 외부 파일에 저장하는 것이다.

Python 을 이용한 Web scraping 의 주요 절차

Web Scraping 의 주요 절차는 아래와 같다.

1. 얻고자하는 데이터의 설정

연구자 본인이 얻고자 하는 데이터가 무엇인지 명확하게 설정할 필요가 있다. Web 에는 굉장히 많은 형태의 데이터가 존재를 하는데, 그 중에서 본인이 필요로하는 데이터가 구체적으로 무엇인지, 그리고 그러한 데이터를 웹에서 획득할 수 있는지 여부를 미리 판단하여야 한다.

2. 얻고자하는 데이터를 담고 있는 웹페이지 확인하기

얻고자 하는 데이터가 무엇인지를 정했으면, 해당 데이터를 어느 웹페이지에서 제공하는지를 살펴보아야 하고, 어느 웹페이지에서 제공되는지가 확인된 후에는 어떠한 방법을 사용해서 가져올 수 있는지에 대해서 생각해야 한다.

1 번과 2 번의 과정은 그 순서가 바뀔 수도 있다. 즉, 먼저 관심있는 사이트를 정하고, 그 사이트에서 제공하는 다양한 정보 중에서 어떠한 정보를 수집/분석할 지에 대해서 결정할 수도 있는 것이다.

3. 웹페이지의 정보 가져오기

원하는 정보를 담고있는 혹은 제공하는 웹페이지를 파악한 후에, 우리가 해야할 것은 Python programing 을 통해서 실제로 해당 페이지에서 정보를 Scraping 하는 것이다. 이를 위해서는 우선 해당 페이지의 URL 주소를 알아야 한다. URL 주소는 브라우저의 주소창에서 복사해서 가져올 수도 있고, 우리가 원하는 페이지의 URL 주소를 담고 있는 또 다른 웹페이지에서 URL 주소를 추출할 수도 있다.

- URL 주소를 준비했다면, 다음과 같은 과정을 거쳐 우리가 원하는 정보를 수집한다.

1) URL 주소 정보를 사용해서 해당 페이지의 정보를 표현하는데 필요한 Source codes 다운 받기

- 필요 module: requests

예를 들어, URL 주소가 'http://www.daum.net/'라고 하면, 이 주소 정보를 이용해서 해당 웹페이지의 정보를 담고 있는 source codes 를 서버로부터 다운로드 받을 수 있다. 이를 위해서는 requests 모듈에서 제공하는 여러개의 함수들 중에서 get 이라는 함수를 사용한다. get 이라는 함수는 URL 주소를 파라미터로 입력 받는다. 다음과 같이 사용할 수 있다.

```
import requests
url = 'http://www.daum.net/'
r = requests.get(url)
```

이렇게 하면 r 이라는 변수에 URL 에 해당하는 페이지에 대한 다양한 정보 (Source codes, 서버 정보, 통신 성공 여부 등)가 저장이 된다. 이러한 여러가지 정보들 중에서 우리가 필요한 정보는 source codes 이다. Source codes 는 r.text 라는 곳에 저장되어 있다.

2) Source codes 에서 우리가 원하는 정보를 담고 있는 tag 확인하기

requests.get(url)을 통해서 다운로드 받은 source codes 에서 우리가 원하는 정보를 담고 있는 tag 가 무엇인지 확인해야 한다. 이를 위해서는 브라우저를 사용한다. 원하는 정보를 담고 있는 웹페이지에서 마우스 오른쪽 버튼을 클릭하여 나오는 메뉴 중에 '소스 보기' (혹은 이와 비슷한 것)를 선택한다. 이를 통해 보여지는 source codes 는 많은 경우에 requests.get(url)을 통해서 다운로드 받은 source codes 와 동일하다. 브라우저에서 보이는 source codes 에서 찾기 명령 (Ctrl+F 사용해서 실행)을 통해서 우리가 추출하고자 하는 정보가 어떠한 tag 에 저장되어 있는지 확인할 수 있다.

3) 원하는 정보를 담고 있는 tag 에 접근하기 혹은 찾기

원하는 정보를 담고 있는 tag 가 무엇인지 확인했으면 requests.get(url)을 통해서 다운로드 받은 source codes (즉 r.text 에 저장되어 있는 source codes)에서 해당 tag 를 찾아야 한다. 이를 위해서는 Python 에서 제공되는 BeautifulSoup 이라는 모듈을 사용한다. 아래와 같이 import 할 수 있다.

Comment [S11]: 보다 자세한 내용은 http://docs.python-requests.org/en/master/ 참조

Comment [S12]: requests 의 get() 함수는 request message 를 보낼 때 GET 이라는 방식을 사용한다. request message 를 서버에 전송할 때는 HTTP (Hypertext Transfer Protocol) 라는 protocol 을 사용하는데, 구체적은 전송 방식에는 GET, POST, PUT, DELETE, HEAD 등의 5 가지 방식이 있다.이중에서 가장 일반적으로 사용되는 방식이 GET 방식이다. 그 다음으로 자주 사용되는 방식이 POST 방식이다. POST 방식으로 request message 를 서버에 보낼 때는 requests 모듈에서 제공되는 post()함수를 사용하면 된다. post() 함수 사용 방법은 http://docs.python-requests.org/en/master/user/quickstart/#more-complicated-post-requests 을 참조할 수 있다.
GET vs. POST
둘의 차이는 request message 를 보내는데 필요한 세부 파라미터 정보를 URL 주소에 포함시켜 전송하느냐 (GET 방식) 아니면 URL 주소에 보이지 않게 숨겨서 전송하느냐 (POST 방식)의 차이이다. 구체적인 설명은 https://www.w3schools.com/tags/ref_httpmethods.asp 나 https://stackoverflow.com/questions/504947/when-should-i-use-get-or-post-method-whats-the-difference-between-them 을 참조할 수 있다.

Comment [S13]: HTTP 프로토콜을 사용해서 서버와 통신한다는 뜻이다. 프로토콜이란 컴퓨터 간의 통신을 위한 정해진 규칙이라고 생각하면 된다.

Comment [S14]: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ 참조

```
from bs4 import BeautifulSoup
```

BeautifulSoup 모듈을 이용해서 원하는 tag 를 찾고자 할 때는 다음과 같은 코드를 사용한다.

```
soup = BeautifulSoup(r.text, 'lxml')
```

이렇게 하면, soup 이라는 변수를 통해서 r.text 에 담겨 있는 source codes 의 개별 tag 를 찾고 접근할 수 있다. 해당 source codes 에 담겨 있는 특정한 tag 를 찾을 때는 BeautifulSoup 에서 제공하는 find()나 find_all() 함수를 사용한다. find() 함수는 특정한 조건을 만족시키는 단 하나의 tag 를 찾는 것이고, find_all() 함수는 해당 조건을 만족시키는 모든 tag 를 다 찾는 것이다. find_all()을 해당 조건을 만족 시키는 모든 tag 를 list 의 형태로 반환한다. 찾고자 하는 조건은 find()함수나 find_all()함수의 파라미터로 제공하게 된다.

특정 조건을 만족시키는 tag 가 source codes 안에 하나 밖에 없는 경우는 find()함수를, 2 개 이상 존재하는 경우는 find_all()함수를 사용하여 반환되는 리스트에 indexing 기능을 사용하여 원하는 tag 를 찾을 수 있다 (혹은 접근할 수 있다).

간단한 예로 원하는 정보가 아래와 같은 tag 에 담겨 있다고 하자.

```
<span class = 'price'> 36,000 </span>
```

이 경우에는 우리가 원하는 정보가 36,000 이고 해당 정보는 span 이라는 tag 에 담겨져 있다. 36,000 이라는 정보를 추출하기 위해서는 먼저 find()함수를 사용해서 해당 정보를 담고 있는 span tag 를 찾아야 한다. 이때 span 이라는 tag 의 이름말고, tag 의 속성 정보 (attributes)를 추가적으로 사용할 수 있다. 위의 예에서는 span 이라는 tag 가 class 라는 속성을 갖고 있고 해당 속성의 값이 price 이다. 이 정보를 사용해서 36,000 이라는 정보를 담고 있는 span 이라는 tag 는 아래와 같이 찾을 수 있다.

```
soup.find('span', attrs={'class': 'price'})
```

find()함수에는 첫번째 파라미터로 찾고자 하는 tag 의 이름을 제공하고 ('span'), 두번째 파라미터로 추가적인 속성 정보를 제공한다. 이를 위해서 attrs 라는 파라미터 이름을 사용한다. 만약 해당 조건을 만족하는 span 이라는 tag 가 source codes 에 하나 밖에 없다면 위의 find()함수는 우리가 원하는 정보를 담고 있는 span tag 를 찾을 것이다. 하지만, 해당 조건을 만족하는 span tag 가 두개 이상 존재하고 우리가 원하는 tag 가 첫번째 tag 가 아니라면 find()함수를 사용해서는 우리가 원하는 tag 에 접근할 수 가 없다. 이러한 경우에는 find_all() 함수를 사용해야 한다. 예를 들어서 source codes 에 다음과 같이 price 라는 class 속성값을 갖는 span tag 가 두개 있다고 가정하자. 그리고 우리가 원하는 정보를 담고 있는 span tag 가 두번째 tag 인 경우에는 find()함수를 사용할 수 없다.

```
<span class = 'price'> 25,000 </span>
```

…

Comment [S15]: 우리가 사용하고자 하는 모듈인 BeautifulSoup 은 bs4 라는 상위 모듈에 속해 있는 하위 모듈입니다. bs4 만을 import 해도 되지만, 우리가 필요한 것은 그 일부인 BeautifulSoup 이기 때문에 from 이라는 키워드를 사용해서 bs4 에서 BeautifulSoup 만을 import 한다.

Comment [S16]: 좀 더 정확하게 표현하자면 BeautifulSoup object 임.

Comment [S17]: 해당 조건을 만족하는 tag 가 여러개인 경우에는 가장 먼저 나오는 tag 만을 찾게 된다.

Comment [S18]: attrs 는 attributes 의 약자이다.

Comment [S19]: 우리가 접근하고자 하는 tag 의 속성 정보를 dictionary 형태로 제공하게 된다. class = 'price'인 경우에는 key 값으로 'class'를 value 에 'price'를 입력하면 된다. dictionary 와 마찬가지로 2 개 이상의 속성 정보를 key:value pair 형태로 추가적으로 제공할 수 있다.

 36,000

이러한 경우에는 find_all() 함수를 이용한다. 사용 방법은 find()함수와 같다. 하지만, 반환되는 결과가 파라미터로 전달된 조건을 만족시키는 모든 tag 들을 담고 있는 리스트라는 점이 다르다.

```
soup.find_all('span', attrs ={'class': 'price'})
```

위 find_all()이 반환하는 데이터는 아래와 같다.

[‘ 25,000 ’, ‘ 36,000 ’]

우리가 원하는 정보를 담고 있는 span tag 는 반환된 리스트의 두번째 element 로 저장되어 있다. 이에 접근하기 위해서는 아래와 같이 indexing 을 사용할 수 있다.

```
soup.find_all('span', attrs ={'class': 'price'})[1]
```

위의 코드가 반환하는 값은 아래와 같다.

‘ 36,000 ’

4) 해당 tag 로부터 원하는 정보 추출하기

우리가 원하는 정보를 담고 있는 tag 를 찾은 다음에 우리가 해야하는 것은 해당 tag 로 부터 우리가 원하는 정보를 추출하는 것이다. 특정 tag 로부터 그 tag 가 저장하고 있는 text 정보를 추출하기 위해서는 해당 tag 정보 끝에 .text 라는 키워드를 추가하면 된다. 이는 간단하게 아래와 같은 방법으로 할 수 있다.

```
data = soup.find(‘span’, attrs={‘class’:’price’}).text
```

혹은 아래와 같은 방법도 사용할 수 있다.

```
tag_result = soup.find(‘span’, attrs={‘class’:’price’})
data = tag_result.text
```

이렇게 하면 우리가 원하는 정보인 36,000 이 data 라는 변수에 저장된다.

find_all()함수에는 끝에 .text 를 사용할 수 없다. 왜냐하면 find_all()이 반환하는 값은 tag 의 내용이 아니고 list 이기 때문이다. 예를 들어서 find_all(‘span’)이라는 함수를 사용한다면, 해당 source codes 에 존재하는 모든 span tag 를 반환할 것이다. 만약 해당 sources codes 에 span 이라는 tag 가 아래와 같이 두 개 있다면

 36,000

….

 Menu A

soup.find_all(‘span’)이라는 코드가 반환하는 결과는 아래와 같다.

[‘ 36,000 ’, ‘ Menu A ’]

따라서 soup.find_all(‘span’).text 라고 하는 것은

[‘ 36,000 ’, ‘ Menu A ’].text

라고 하는 것과 마찬가지이다. 이는 불가능하다.

아래와 같이 개별 요소 (element)에 접근하여 그 뒤에 .text 를 사용하여야 한다.

soup.find_all(‘span’)[0].text → 36,000 반환

soup.find_all(‘span’)[1].text → Menu A 반환

5) 해당 정보를 저장하기 혹은 외부 파일에 기록하기

다운로드 받은 데이터를 이용해서 어떠한 추가 작업을 할 때, 외부의 별도 프로그램 (예, 엑셀)을 이용할 수도 있고, Python 에서 제공하는 기능을 사용할 수도 있다. 외부의 별도 프로그램에서 추가적인 작업을 하고자 하는 경우에는 웹으로부터 수집한 데이터를 외부파일에 저장해야 한다. 이 때는 1 차적으로 원하는 정보를 구분자 (예, tab(\t), comma(,), semicolon (;) 등)를 사용해서 텍스트 파일의 형태로 저장하는 것이다. 그리고 이를 엑셀에서 불러와서 엑셀 형태로 저장을 하고 이를 다른 프로그램에서 다시 불러올 수 있다.

Comment [S110]: File input & output tutorial 참조

원하는 정보를 text 파일로 저장하기 위해서는 다음과 같은 코드를 사용할 수 있다.

f = open(‘파일이름.txt’, ‘w’, encoding = ‘utf-8’)

원하는 정보가 한글인 경우에 encoding parameter 를 위와 같이 제공해야 한다.

f.write(data)

f.close()

위의 코드가 하는 것은 Python file 이 저장되어 있는 폴더에 ‘파일이름.txt’파일을 새롭게 생성하고 write()라는 함수를 사용해서 웹으로부터 수집한 데이터 (즉 data 라는 변수에 저장되어있는 값)을 해당 파일에 저장하는 것이다. 저장하고자 하는 내용이 두개 이상인 경에는 구분자를 사용해야 한다. 원하는 정보를 text 파일로 저장한 이후에는 close()함수를 사용해서 해당 파일을 안전하게 닫아야 한다.

BeautifulSoup 의 Navigation 기능 사용하기

Source codes 에서 원하는 tag 를 찾고자 하는 경우에는 보통 위에서 설명한 find() 또는 find_all() 함수를 사용하면 된다. 하지만, 이 두개의 함수를 사용해서는 원하는 tag 에 접근하지 못하는 경우도 있다. 이러한 경우에는 BeautifulSoup 에서 제공되는 Navigation 기능을 사용할 수 있다. BeautifulSoup 이 어떻게 source codes 를 navigate 하는지를 알기위해서는 source codes, 즉, tag 들이 어떻게 구성되어 있는지를 먼저 이해할 필요가 있다. 우리가 웹으로부터 다운로드 받은 source codes 가 아래와 같이 구성되어 있다라고 가정하자.

```
<html>
        <body>
                <p class="title">HTML code for the lecture</p>
                <p class="description"> This is an example. </p>
        </body>
</html>
```

위의 예에서 볼 수 있듯이 source codes 를 구성하고 있는 tag 들은 hierarchical (위계적)하게 구성되어 있다. tag 들 간의 관계는 그 구조에 따라서 부모 (parent), 자식(contents), 형제자매 (siblings)로 이뤄져 있다. 위의 source codes 를 예로 설명하도록 하겠다. 첫번째로 html tag 와 body tag 간의 관계를 보도록 하자. body tag 가 html tag 안에 포함된 것을 볼 수 있는데, 이러한 경우 html 은 body tag 의 부모 tag 가 되고 반대로 body tag 는 html tag 의 자식 tag 가 되는 것이다.

Comment [S111]: BeautifulSoup 에서 children 은 다른 의미를 갖는다. https://www.crummy.com/software/BeautifulSoup/bs4/doc/ 참조

예를 들어서, 우리가 find()나 find_all() 함수를 통해서 body tag 에 직접적으로 접근할 수 없다라고 가정하자. 하지만, html tag (즉, body 의 부모 tag)에는 직접적으로 접근할 수 있다라고 한다면, 이러한 상황에서는 find()함수를 통해서 html tag 에 먼저 접근하고 navigation 기능을 통해서 html tag 의 자식 tag 인 body tag 에 접근하는 것이 가능하다. BeautifulSoup 에서는 특정 tag 의 자식 tag 에 접근하기 위해서 .contents 라는 키워드를 사용할 수 있다. html tag 를 찾은 다음에 그 뒤에 .contents 를 붙여서 html tag 의 자식 tag 인 body tag 에 접근할 수 있다.

위의 코드에 대해서 아래와 같은 코드를 실행하면,

```
soup = BeautifulSoup(html, 'lxml')
soup.html.contents
```

soup.html.contents 는 html tag 가 갖고 있는 모든 자식 elements 를 리스트의 형태로 리턴한다. 즉, 그 결과는 아래와 같다.

```
['\n', <body>
 <p class="title">HTML code for the lecture</p>
 <p class="description"> This is an example. </p>
 </body>, '\n']
```

위의 결과를 보면 html 의 자식 elements 에는 tag 들만 아니라는 것을 알 수 있다. 즉, \n 문자가 들어 있는 것이다. 이는 html tag 가 포함하고 있는 tag 뿐만 아니라 다른 문자들도 자식 elements 에 해당이 된다는 것이다. 위의 source codes 를 보면 <html> 다음에 new line (즉, Enter sign)이 포함되어 있는 것을 알 수 있다. 즉, <html> 다음에 한줄 밑으로 내려 간다음에 <body> tag 가 시작한다. 이러한 공백문자 (\n)도 한 tag 의 자식 elements 가 된다. 그래서 soup.html.contents 가 리턴하는 결과의 첫번째 element 와 세번째 element 가 '\n'이 되는 것이다. 리턴된 결과에서 우리가 관심있는 것은 body tag 즉, 두번째 element 가 된다. 따라서 우리는 해당 tag 에 다음과 같은 indexing 으로 접근할 수가 있다.

```
soup.html.contents[1]
```

위의 코드는 다음과 같은 string value 를 리턴한다.

```
<body>
<p class="title">HTML code for the lecture</p>
<p class="description"> This is an example. </p>
</body>
```

반대로 우리는 .parent 라는 키워드를 사용해서 특정 tag 의 부모 tag 에 접근할 수가 있다. 예를 들어서 body tag 에 먼저 접근한다음에 .parent 키워드를 사용해서 body 의 부모 tag 인 html 에 접근할 수 있는 것이다. 아래와 같이 할 수 있다.

```
soup.body.parent
```

위 코드의 결과는 아래와 같다. 즉, html tag (body tag 의 부모 tag)의 모든 내용이 반환된다.

```
<html>
<body>
<p class="title">HTML code for the lecture</p>
<p class="description"> This is an example. </p>
</body>
</html>
```

형제자매 tag 에 접근하기

이번에는 특정한 tag 의 형제자매 (sibling) tag 에 접근하는 방법에 대해서 살펴보도록 하겠다. 위의 예제 source codes 에서 첫번째 p tag 와 두번째 p tag 를 살펴보자. 두 tag 는 들여쓰여진 정도가 같은데, 이는 두 tag 가 서로 형제자매(sibling)이라는 것이다. 즉, 같은 부모 (body tag) tag 를 가지고 있는 것이다. 만약에 우리가 첫번째 p tag 에 find() 함수를 사용해서 접근할 수 있고, 두번째 tag 는 find()를 사용해서 접근할 수 없다라고 한다면, 그러한 경우에 find() 함수를 사용해서 첫번째 p tag 에 먼저 접근한 다음에 navigation 기능을 사용해서 두번째 p tag 에 접근할 수 있다. 특정 tag 의 다음 sibling tag 에 접근하기 위해서는 BeautifulSoup 에서 제공되는 .next_sibiling 키워드를

사용하고 이전 sibling tag 에 접근하기 위해서는 .previous_sibling 키워드를 사용할 수 있다. 첫번째 p tag 를 찾은 다음에 그 다음 p tag 에 접근하기 위해서는 다음과 같이 코드를 작성할 수 있다.

```
soup.find('p',attrs={'class':'title'}).next_sibiling
```

Comment [S112]: HTML code for the lecture 가 리턴 된다.

하지만 이렇게만 코드를 작성하면 그 결과로 리턴되는 값은 '\n'이다. 왜냐하면 위에서 설명한 것 처럼 tag 만이 아니라 문자도 (공백문자 포함) 하나의 element 로 간주되기 때문이다. 두번째 p tag 에 접근하기 위해서는 '\n'을 찾은 다음 다시 한번 .next_sibling 을 붙여 줘야 한다. 즉, 두번째 p tag 는 '\n'의 next sibling element 가 되는 것이다.

```
soup.find('p',attrs={'class':'title'}).next_sibling.next_sibling
```

위 코드의 결과는

```
<p class="description"> This is an example. </p>
```

가 된다.