# Web Scraping

Web Scraping은 Web(인터넷)에 존재하는 다양한 형태의 정보와 데이터를 수집하는 것 (Web data collection)을 의미합니다. 한국에서는 Web Crawling이라고 더 많이 알려져 있습니다. 하지만, 명확하게 구분하자면 Web Scraping과 Web Crawling의 의미는 다릅니다. Web Crawling은 여러 개의 웹페이지를 이동한다(crawl)는 뜻이 더 큰 반면, scraping은 특정 페이지의 정보/데이터를 수집한다 혹은 긁어온다(scrape)는 뜻이 더 많이 포함되어 있습니다. 하지만, 많은 경우 두 표현은 같은 의미로 사용됩니다. 본 글에서는 Web Scraping이라는 표현을 쓰기로 하겠습니다.

Web scraping은 기본적으로 컴퓨터 통신 (computer networking)을 통해 이뤄집니다. 즉, 여러분들이 여러분이 원하는 정보를 담고 있는 웹페이지에서 해당 정보를 여러분들의 컴퓨터로 다운로드 받기 위해서는 해당 정보를 담고 있는 웹페이지를 저장하고 있는 컴퓨터[[1]](#footnote-1)와 통신을 해서 해당 컴퓨터에서 웹페이지에 대한 내용을 다운로드 받게 되는 것입니다. 따라서 Web scraping을 제대로 이해하기 위해서는 컴퓨터 간 통신에 대해서 기본적으로 어느 정도 이해하고 있는 것이 필요합니다.

## Web communication 이해

Web scraping 프로그램을 만들기 위해서는 일단 컴퓨터 간의 통신이 어떻게 이뤄지는지를 이해할 필요가 있습니다. 이를 위해서, 하나의 예를 들어보겠습니다. 우리가 컴퓨터를 이용해서 인터넷에 연결을 하고 특정한 웹페이지 (예, www.daum.net)에 접속한다고 가정해 보겠습니다. 여러분들은 컴퓨터를 이용해서 어떠한 웹이지에 접속할 때 무엇을 제일 먼저 하나요? 컴퓨터가 켜져 있고 인터넷에 연결되어 있는 상황에서, 사용자가 하나의 웹페이지 (예, [www.daum.net](http://www.daum.net))에 접속하기 위해서 가장 먼저하는 것은 아마도 웹 브라우저 (browser)를 실행시키는 것일 겁니다. 주로 사용되는 웹 브라우저로는Chrome, Firefox, Internet Explorer등이 있습니다. 웹 브라우저를 실행시킨 후에 우리는 해당 페이지에 접속하기 위해 브라우저 주소창에 접속하고자 하는 페이지의 주소 (URL, uniform resource locator 주소라고 함)를 입력합니다. 다음(Daum) 페이지에 접속하고자 하는 경우에는 [www.daum.net](http://www.daum.net)의 주소를 브라우저의 주소창에 입력하는 것입니다. 그런 다음 Enter를 누르면, 해당 페이지의 내용이 여러분들의 웹 브라우저 상에 나타나게 됩니다. 주소를 입력하고 화면에 우리가 원하는 페이지가 표시되기까지 걸리는 시간은 굉장히 짧습니다. 이 짧은 시간에 브라우저에서는 어떠한 일이 벌어져서 (혹은 브라우저가 어떠한 일들을 수행해서) 우리가 원하는 웹 페이지가 브라우저 상에 표시가 되는 것일까요?

간단하게 말하면, 브라우저는 해당 주소의 웹페이지를 화면에 표시하기 위해 웹페이지를 화면에 표현하는데 필요한 데이터를 (데이터가 저장되어 있는) 서버 컴퓨터에서 다운로드 받고, 이를 정해진 규칙에 의해서 브라우저 화면에 표시하게 됩니다. 브라우저의 역할을 좀 더 구체적으로 살펴보도록 하겠습니다.

### 브라우저의 역할

여러분들이 접속하고자 하는 웹페이지를 브라우저상에 표현하기 위해 브라우저는 다음과 같은 일들을 합니다.

**① URL 주소를 IP주소로 변환하기**

사용자가 접속하고자 하는 웹 페이지의 주소 (즉, URL 주소)를 입력 받은 브라우저는 해당 URL 주소를 컴퓨터가 이해할 수 있는 주소인 IP (Internet protocol) 주소로 변환을 합니다 (URL 주소를 IP주소로 변환하기 위해서는 URL 주소와 IP주소가 저장되어 있는 또 다른 서버 컴퓨터를 사용합니다. 이러한 서버 컴퓨터를 DNS (domain name service) 서버합니다. 각각의 URL 주소는 고유한 IP 주소를 갖습니다). URL 주소는 사용자를 위한 주소이고, IP 주소는 컴퓨터를 위한 주소라고 생각하면 됩니다.. IP주소는 숫자로 구성되어 있는데 보통 161.131.12.11 등으로 표현이 됩니다. 이러한 IP 주소는 하나의 컴퓨터가 인터넷 상에서 지니는 고유한 번호 혹은 주소라고 생각할 수 있습니다. 그리고 브라우저는 이러한 IP주소를 사용해서 해당 IP주소를 갖고 있는 컴퓨터를 인터넷 상에서 찾을 수 있고, 통신할 수 있는 것입니다.

**② IP 주소를 사용해서 서버 컴퓨터 에 접속하기**

사용자가 입력한 URL 주소에 해당하는 웹페이지를 화면에 표시하기 위해서 브라우저는 해당 페이지를 표시하는데 필요한 원본 데이터가 있어야 합니다. 이러한 원본 데이터는 해당 URL 주소에 해당하는 IP주소를 갖는 서버 컴퓨터에 저장되어 있습니다. DNS 서버를 통해서 URL 주소에 해당하는 IP주소를 획득한 브라우저는 이 주소 정보를 이용해서 해당 IP주소를 갖는 서버 컴퓨터에 접속합니다.

**③ 웹페이지를 구성하는 원본 데이터 받기**

해당 서버 컴퓨터에 접속한 다음, 사용자가 입력한 주소에 해당하는 웹 페이지를 브라우저에 표시하기 위해서 우리의 부라우저는 해당 페이지를 표시하는데 필요한 데이터를 서버 컴퓨터에게 요청합니다. 이를 위해 브라우저는 서버 컴퓨터에 요청 메시지 (request message)를 보내게 됩니다. 즉, “내 사용자가 [www.daum.net](http://www.daum.net) 페이지에 접속하고자 하니까, 해당 페이지를 브라우저 상에 표현하는데 필요한 데이터를 나에게 보내줘.” 라는 내용이 들어있는 메시지를 서버에게 보내는 것입니다. 그러면 서버 컴퓨터는 요청 메시지를 받은 후, 메시지를 보낸 브라우저나 컴퓨터가 누구인지 확인합니다. 그리고, 보안상 아무런 문제가 없다는 것을 확인한 후에 아무런 문제가 없다라고 판단이 되면, 해당 요청 메시지에 응답하게 됩니다. 이때 서버 컴퓨터는 사용자의 브라우저에게 응답 메시지(response message)를 보내게 되는데, 이 응답 메시지에는 기본적으로 사용자가 접속하고자하는 웹페이지를 표시하는데 필요한 데이터가 들어 있고, 추가적으로 해당 서버와 브라우저가 커뮤니케이션 하는데 필요한 정보가 포함되어 있다. 응답 메시지에 저장되어 있는 이러한 여러가지 정보 혹은 데이터 중에서 우리가 필요한 것은 우리가 원하는 웹페이지를 브라우저상에 표현하는데 필요한 데이터 입니다.

**④ 서버로부터 받은 데이터를 이용해서 브라우저 상에 표시하기**

최종적으로 브라우저는 서버로부터 다운로드 받은 데이터를 가지고 사전에 정해진 규칙에 따라서 사용자가 보기 편한 방식으로 화면에 해당 웹페이지를 표시하게 됩니다.

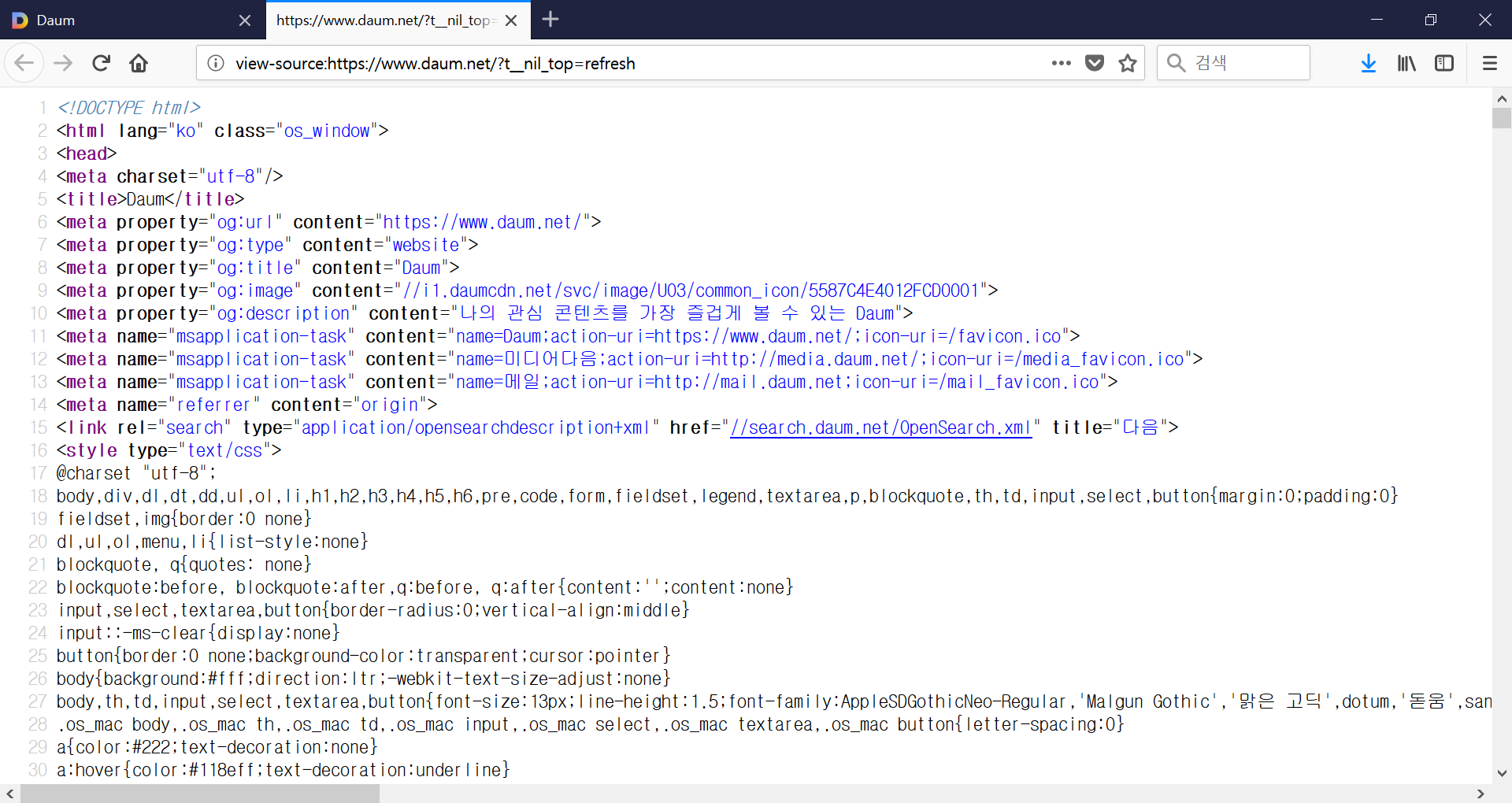
이러한 과정을 거쳐서 사용자는 접속하고자 하는 웹페이지의 URL주소를 입력한 다음, 해당 페이지를 브라우저 상에서 볼 수가 있는 것입니다.

## 웹 페이지를 구성하는 데이터의 구조

그렇다면, 웹페이지를 구성하는 데이터는 어떻게 생겼을까요? 이를 설명하기 위해, 여러분이 접속하고자 하는 웹페이지가 [www.daum.net](http://www.daum.net)이라고 가정하겠습니다. 브라우저의 주소창에 [www.daum.net](http://www.daum.net)이라고 입력을 하고 Enter 버튼을 누르면, 여러분은 다음과 비슷한 화면을 보게 될 것입니다.



이는 위에서 설명한 것 처럼 사용자의 브라우저가 사용자가 입력한 URL주소를 먼저 IP주소로 변환하고 이를 이용해서 해당 페이지를 구성하는 데이터를 저장하고 있는 서버에 접속해서 해당 데이터를 받은 다음에, 정해진 규칙에 의해 화면에 사용자가 보기 편한 방식으로 표시한 것입니다. 웹페이지를 구성하는 (혹은 웹페이지를 브라우저 상에 표현하는데 필요한) 데이터는 대부분 소스 코드(source code)라는 것으로 구성되어 있습니다. 이러한 소스 코드는 해당 웹페이지에서 여러분의 마우스 오른쪽 버튼을 클리하면 나오는 메뉴 중에서 ‘소스코드 보기’ 라는 하위 메뉴를 선택하면 볼 수 있습니다. Daum 페이지의 소스 코드는 아래 그림과 같습니다.



웹페이지를 구성하는 소스 코드는 여러가지 종류의 컴퓨터 프로그래밍 언어로 작성되어 있습니다. 주로 사용되는 언어로는 HTML, XML, Java Script등이 있습니다. 하지만, Web scraping을 하는데 있어서는 소스 코드가 어떠한 컴퓨터 프로그래밍 언어로 작성되어 있는지, 그리고 각 언어의 특성이 무엇인지에 대해서 잘 몰라도 됩니다.

웹 데이터 수집에 있어서 **여러분이 기억해야하는 것은 소스 코드는 여러 개의 태그 (tags) 들로 구성되어 있다라는 것입니다.** 예를 들어, 위의 [www.daum.net](http://www.daum.net) 페이지를 구성하는 소스 코드는 아래와 같이 구성되어 있습니다.

<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/5587C4E4012FCD0001">

<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">

이하 생략

여기에서 <title>, <meta>와 같이 각괄호(<> 괄호)를 사용하여 표현된 것을 태그(tag)라고 합니다. 보통 하나의 태그는 시작 태그(start tag)와 종료 태그 (end tag)의 짝으로 구성되어 있습니다. 위의 예에서 title 태그를 살펴보도록 하겠습니다. title 태그를 보면 <title> … </title>로 구성되어 있는 것을 알 수 있는데, <title>이 시작 태그이고, </title>이 종료 태그가 됩니다. 일반적으로 이러한 태그의 주된 역할은 웹 브라우저 상에서 사용자에게 보이는 텍스트 정보를 저장하는 것입니다. 그러한 텍스트 정보는 특정 태그의 시작 태그와 종료 태그 사이에 저장되게 됩니다. 위의 경우, title 태그를 보면 시작 태그 (<title>)와 종료 태그 (</title>) 사이에 “Daum”이라는 텍스트 정보가 저장되어 있는 것을 알 수 있습니다. 일반적으로 이러한 텍스트 정보가 우리가 웹에서 최종적으로 수집하고자 하는 데이터가 되는 것입니다.

보통 하나의 태그는 위에서 언급한 것 처럼 시작 태그와 종료 태그의 짝으로 구성이 되어 있는데, 항상 그런 것은 아닙니다. 시작 태그만 존재하는 태그들도 있습니다. 그러한 태그들 중에서 가장 대표적인 태그가 위의 예에서 사용된 meta라는 태그입니다. meta 태그는 사용자를 위한 태그가 아니라 컴퓨터나 브라우저를 위한 해당 페이지의 추가정보, 즉 메타 정보,를 담고있는 태그입니다. 그래서 meta 태그에 저장되어 있는 정보는 브라우저 상에 표시되지 않습니다. meta 태그와 같이 시작 태그만 있는 경우에는 어떠한 정보가 각괄호 (<>) 사이에 저장됩니다.

## Web scraping 프로그램의 역할

우리가 작성하는 Web scraping 프로그램은 크게 두가지 역할을 수행하여 웹에 있는 정보를 추출합니다. 첫번째는 브라우저와 비슷한 역할인 우리가 추출하고자 하는 정보를 담고 있는 웹페이지를 구성하는 데이터, 즉 소스 코드를 저장하고 있는 서버 컴퓨터와의 통신을 통해 해당 소스코드를 우리들의 컴퓨터로 다운로드하는 역할이고, 두번째는 그렇게 다운로드 받은 소스 코드에서 우리가 원하는 특정한 정보를 추출∙저장하는 역할입니다.

### 서버로부터 웹페이지의 소스코드 다운로드 받기

웹 브라우저가 서버 컴퓨터와 통신을 하기 위해서 URL 주소가 필요한 것 처럼, 우리가 Python을 통해 만드는 Web scraping 프로그램도 특정 서버 컴퓨터와 통신하기 위해서는 URL 주소가 필요합니다. 그러면 우리의 Web scraping 프로그램은 해당 URL 주소 (예, [www.daum.net](http://www.daum.net)) 를 이용해서 서버에 접속하고 브라우저와 마찬가지로 요청 메시지를 보냅니다. 그러면 역시나 마찬가지로 서버 컴퓨터는 해당 요청 메시지에 대한 응답 메시지를 보내는데, 여기에 우리가 원하는 데이터가 저장되어 있는 것입니다. 즉, 우리가 원하는 정보를 담고 있는 웹페이지의 소스 코드가 저장되어 있는 것입니다. 파이썬에서는 다른 컴퓨터와의 통신을 위해 특정 모듈을 사용하게 됩니다. 가장 일반적으로 많이 사용되는 모듈에 requests, urllib, urllib2 등이 있습니다.

### 소스 코드로부터 원하는 정보 추출∙저장

서버로부터 우리가 원하는 정보를 담고 있는 웹페이지의 소스 코드를 다운로드 받은 다음에는 소스 코드에서 우리가 원하는 정보만을 추출해야 합니다. 특정 웹페이지를 구성하는 소스 코드에는 해당 웹페이지를 브라우저에 표시하는데 필요한 수많은 정보를 담고 있습니다. 하지만, 우리는 이 중에서 일부의 정보만을 필요로 합니다. 우리가 원하는 정보는 보통 텍스트 형태 (즉, 문자열 데이터 타입)로 특정 태그의 시작 태그와 종료 태그 사이에 저장되어 있습니다. 웹페이지의 소스 코드를 서버와의 통신을 통해서 다운로드 받은 다음 해야하는 작업은 소스 코드를 구성하고 있는 여러 개의 태그들 중에서 우리가 원하는 정보를 담고 있는 태그를 찾아서, 그 태그에서 우리가 원하는 정보를 추출하고 저장하는 것입니다.

## Python을 이용한 Web scraping의 주요 절차

Python을 이용한 Web scraping의 주요 절차는 아래와 같습니다.

① 얻고자하는 데이터의 설정

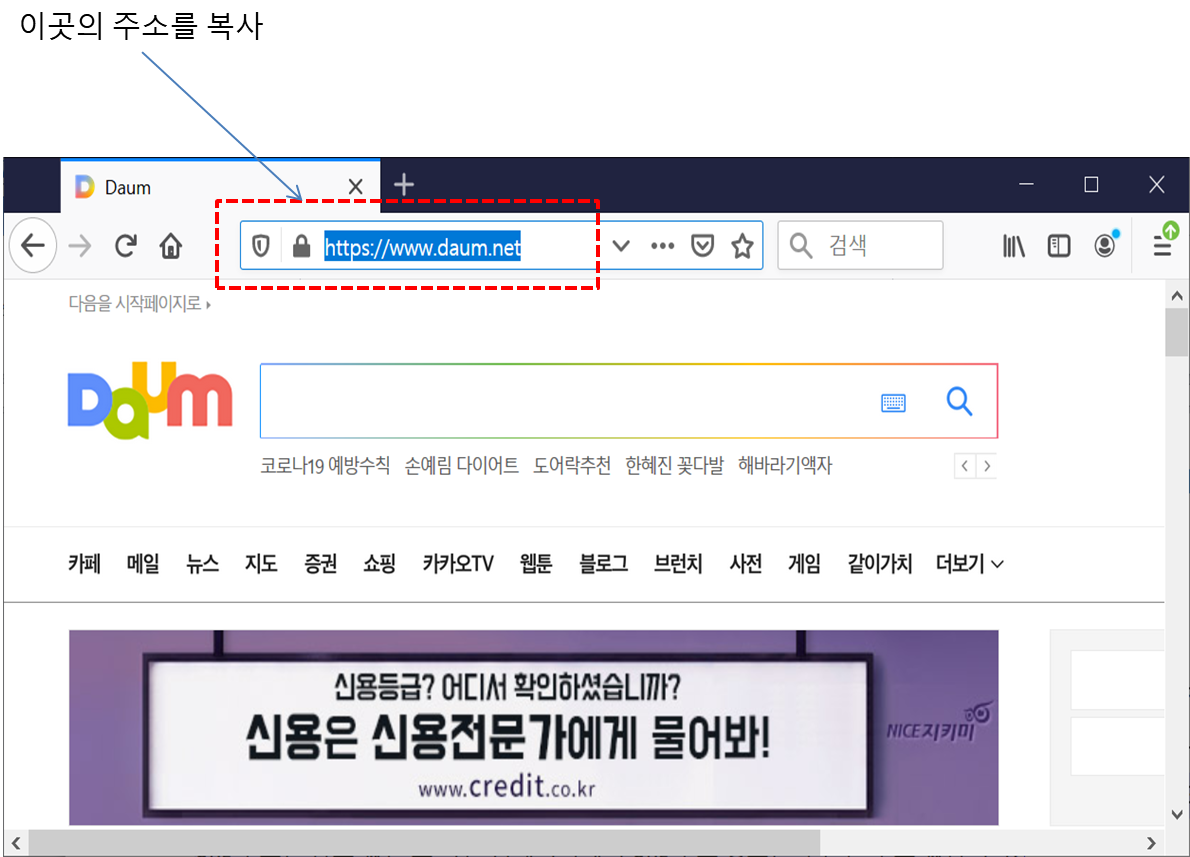
데이터 수집을 시작하기 전에 본인이 얻고자 하는 데이터가 무엇인지 명확하게 설정하는 것이 필요합니다. Web에는 굉장히 많은 형태의 데이터가 존재를 하는데, 그 중에서 본인이 필요로하는 데이터가 구체적으로 무엇인지, 그리고 그러한 데이터를 웹에서 획득할 수 있는지 여부를 미리 판단해야 합니다.

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 데이터를 수집할 수 있는지에 대해 생각해야 합니다.

③ 웹페이지의 정보 가져오기

원하는 정보를 담고있는 혹은 제공하는 웹페이지를 파악한 후에, 우리가 해야할 것은 Python 을 통해서 실제로 해당 페이지에서 정보를 스크래핑 하는 것입니다. 이를 위해서는 우선 해당 페이지의 URL 주소를 알아야 합니다. URL 주소는 해당 페이지로 이동 후에 브라우저의 주소창에서 복사해서 가져옵니다 (아래 그림 참고).



또는 우리가 원하는 페이지의 URL주소를 담고 있는 또 다른 웹페이지에서 URL 주소 정보를 추출할 수도 있습니다 (이는 이후에 좀 더 자세히 설명합니다).

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

1. 원하는 정보를 담고 있는 페이지의 소스 코드를 다운로드 받는다.
2. 원하는 정보를 담고 있는 태그를 찾아서 정보를 추출한다.

각 과정에 사용될 수 있는 모듈과 클래스 정보를 정리하면 아래와 같습니다.

|  |  |
| --- | --- |
| **Role** | **Related module** |
| Web communication  => download the (raw) data (i.e., source codes) | requests 또는 urllib |
| Information extraction  => extract the information from the source codes and saves it | BeautifulSoup |

각 과정에 대해서 자세히 살펴보도록 하겠습니다.

### URL 주소 정보를 사용해서 해당 페이지의 소스 코드 다운로드 하기

- 필요 모듈: requests[[2]](#footnote-2)  
 예를 들어, URL 주소가 ‘http://www.daum.net/’라고 하면, 이 주소 정보를 이용해서 해당 웹페이지의 정보를 담고 있는 소스 코드를 서버로부터 다운로드 받을 수 있습니다. 이를 위해서는 requests라고 하는 모듈을 사용합니다. requests 모듈에는 다른 컴퓨터와의 통신에 필요한 여러가지 함수가 저장되어 있습니다. 여기서는 requests 모듈에서 제공하는 여러개의 함수들 중에서 get이라는 함수를 사용합니다.

Tip!

requests의 get() 함수는 request message를 보낼 때 GET이라는 방식을 사용하는 것입니다. request message를 서버에 전송할 때는HTTP (Hypertext Transfer Protocol) 라는 protocol을 사용하는데, 구체적은 전송 방식에는 GET, POST, PUT, DELETE, HEAD 등의 5가지 방식이 있습니다. 이중에서 가장 일반적으로 사용되는 방식이 GET 방식입니다. 그 다음으로 자주 사용되는 방식이 POST 방식입니다. POST방식으로 request message를 서버에 보낼 때는 requests 모듈에서 제공되는 post()함수를 사용합니다. post() 함수 사용 방법은<https://requests.readthedocs.io/en/master/user/quickstart/#more-complicated-post-requests> 을 참고할 수 있습니다.

GET 방식과 POST 방식의 차이는 request message를 보내는데 필요한 세부 파라미터 정보를 URL 주소에 포함시켜 전송하느냐 (GET 방식) 아니면 URL 주소에 보이지 않게 숨겨서 전송하느냐 (POST 방식)의 차이입니다. 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 참고할 수 있습니다.

get이라는 함수는 URL주소를 인자로 입력 받습니다. 다음과 같이 사용합니다.

import requests

url = 'http://www.daum.net/' #우리가 원하는 정보를 담고 있는 웹페이지의 URL 주소

r = requests.get(url)

이렇게 하면 r에는 url에 해당하는 서버에서 보낸 응답 메시지의 내용이 저장됩니다. 여기에는 소스 코드 이외도 다양한 정보 (서버 정보, 통신 성공 여부 등)가 저장이 됩니다. 이러한 여러가지 정보들 중에서 우리가 필요한 정보는 소스 코드입니다. 소스 코드는 r.text라는 곳에 저장되어 있습니다.

**소스 코드에서 우리가 원하는 정보를 담고 있는 tag 확인하기**

requests.get(url)을 통해서 소스 코드를 다운로드 받은 다음에는 해당 소스 코드에서 우리가 원하는 정보를 담고 있는 태그가 무엇인지 확인해야 합니다. 이를 위해서는 브라우저를 사용합니다. 원하는 정보를 담고 있는 웹페이지에서 마우스 오른쪽 버튼을 클릭하여 나오는 메뉴 중에 ‘소스 보기’ (혹은 이와 비슷한 것)를 선택합니다. 이를 통해 보여지는 소스 코드는 많은 경우에 requests.get(url)을 통해서 다운로드 받은 소스 코드와 동일합니다. 브라우저에서 보이는 소스 코드에서 찾기 명령 (Ctrl+F 사용해서 실행)을 통해서 우리가 추출하고자 하는 정보가 어떠한 태그에 저장되어 있는지 확인할 수 있습니다.

### 원하는 정보를 담고 있는 태그 찾기: BeautifulSoup 사용하기

원하는 정보를 담고 있는 태그가 무엇인지 확인했으면 requests.get(url)을 통해서 다운로드 받은 소스 코드 (즉 r.text에 저장되어 있는 소스 코드)에서 해당 태그를 찾아야 합니다. 이를 위해서는 Python에서 제공되는 BeautifulSoup이라는 클래스를 사용합니다.[[3]](#footnote-3) 아래와 같이 bs4라는 모듈에서 해당 클래스를 import할 수 있다.

from bs4 import BeautifulSoup

BeautifulSoup 클래스를 이용해서 원하는 태그를 찾고자 할 때는 다음과 같이 BeautifulSoup 클래스의 생성자 함수를 사용해서 해당 클래서의 복사본인 객체를 먼저 생성해야 합니다.

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

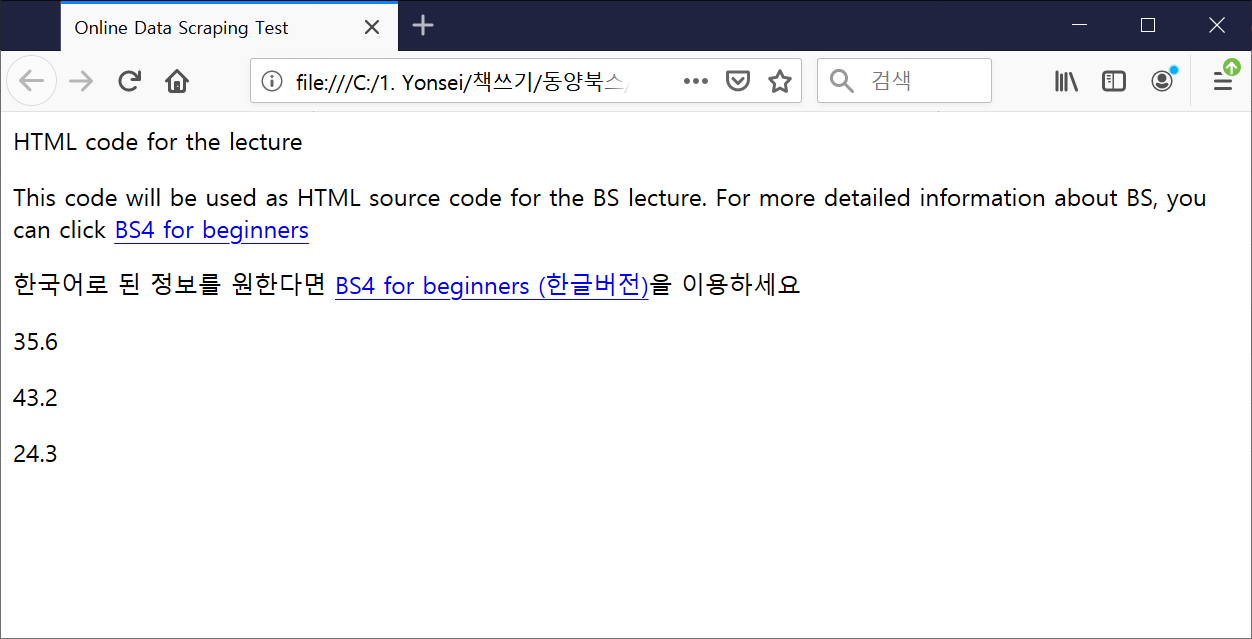
이렇게 하면, soup이라는 BeautifulSoup의 객체를 통해서 r.text에 담겨 있는 소스 코드의 태그를 찾고 접근할 수 있습니다. 생성자 함수의 두번째 인자로 제공된 'lxml'은 BeautifulSoup이 태그를 찾을 때 사용하는 파서 (parser)의 이름입니다. 소스 코드에서 태그를 찾는 것을 파싱(parsing)이라고 하고 파싱에 사용되는 것이 파서 (parser)입니다. BeautifulSoup에서 사용할 수 있는 파서의 종류에는 크게 3가지가 있습니다. “lxml”, “html5lib”, “html.parser”가 그것입니다.[[4]](#footnote-4) 이 중에서 가장 일반적으로 많이 사용되는 것이 lxml입니다.

위의 코드와 같이 입력하면 soup을 이용해서 r.text에 저장되어 있는 태그들을 탐색할 수 있는데, 이때 lxml이라는 파서를 사용해서 탐색한다는 것을 의미합니다.

BeautifulSoup을 이용해서 태그를 찾는 방법에는 크게 아래와 같이 3가지 정도가 있습니다.

* 태그의 이름 사용하기
* BeautifulSoup에서 제공되는 find() 또는 find\_all() 함수 사용하기
* BeautifulSoup의 네비게이션 (navigation) 기능 사용하기

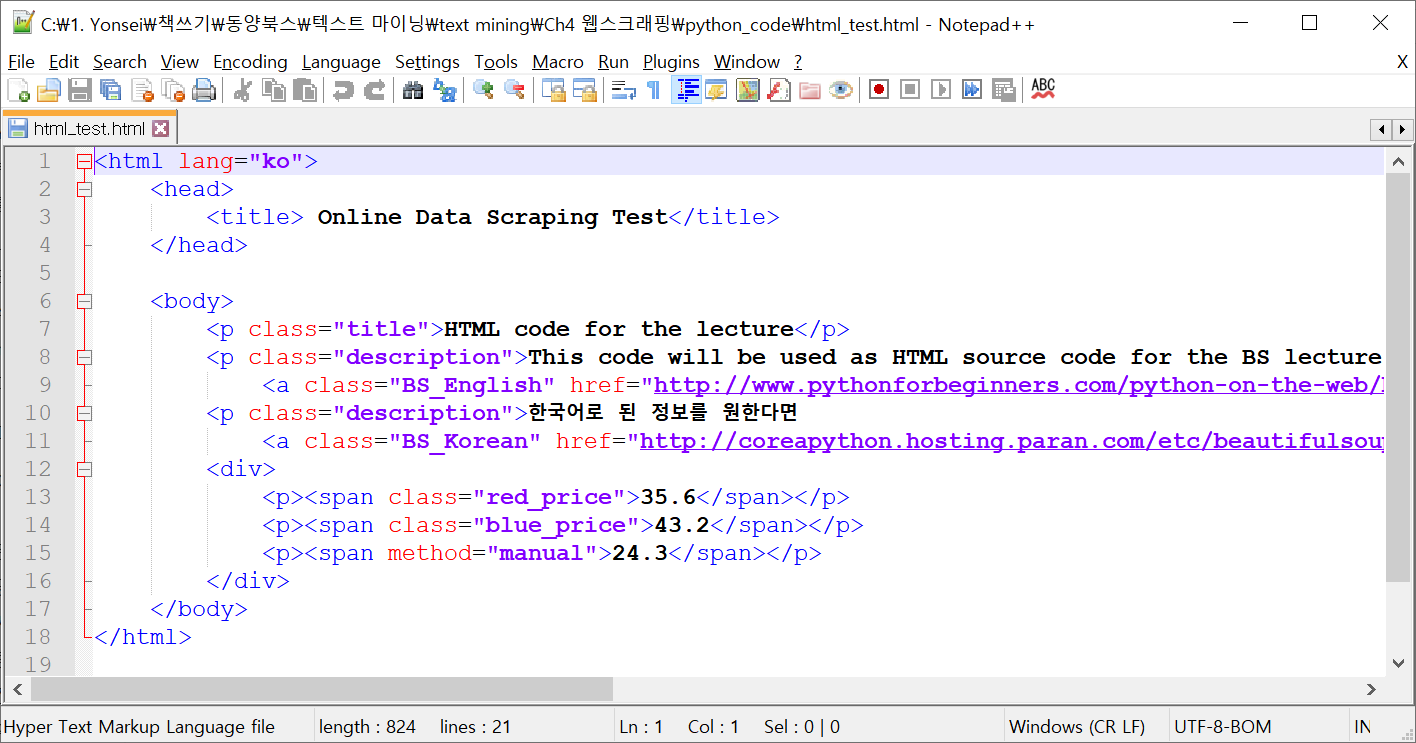
BeautifulSoup 을 사용해서 태그 찾는 방법을 설명하기 위해서 여기서는 저자가 작성한 간단한 웹페이지 파일을 사용하도록 하겠습니다. 해당 파일의 이름은 html\_test.html 입니다. 해당 파일에는 여러개의 태그들이 저장되어 있습니다. 해당 파일을 브라우저에서 열면 다음과 같이 보입니다.



해당 페이지에서 소스 코드 보기를 하면 아래와 같이 소스 코드가 구성되어 있는 것을 알 수 있습니다.



위의 내용이 실제 html\_test.html 파일에 저장되어 있는 내용입니다. 즉, 이러한 소스 코드가 저장되어 있는 파일 (확장자가 html인 파일)을 브라우저에서 열면 정해진 규칙에 따라 사용자가 보기 편하게 표시되는 것입니다. 해당 파일을 노트패드와 같은 텍스트 에디터에서 열면 그 내용이 같음을 알 수 있습니다 (아래 그림 참고).



여기서는 설명을 위해서 해당 파일에 저장되어 있는 소스 코드를 사용하겠습니다. 이를 위해서 다음과 같은 코드를 이용해서, 소스 코드를 주피터 노트북 상으로 불러 옵니다.

with open('html\_test.html', 'r', encoding = 'utf-8') as f:

html= f.read()

html 변수에 해당 소스 코드가 저장됩니다 (아래 참고). 여러분들은 원래 requests.get()함수를 사용해서 특정 웹페이지의 소스 코드를 직접 웹으로부터 다운로드 받아야 하는데, 여기서는 그 과정이 대신, 로컬 드라이브에 있는 웹페이지 파일에 저장되어 있는 소스 코드를 불러온 것입니다.



해당 소스 코드를 BeautifulSoup을 이용해서 탐색하기 위해서 아래와 같이 코딩합니다.

soup=BeautifulSoup(html, 'lxml')

그러면 soup 객체를 사용해서 html에 저장되어 있는 태그들을 탐색할 수 있습니다.

**태그 찾기: 1) 태그의 이름 사용하기**

태그를 찾는 가장 간단한 방법은 (찾고자 하는) 태그의 이름을 사용하는 것입니다. 예를 들어서, 우리가 원하는 정보가 title 태그에 저장되어 있는 경우, 해당 정보를 추출하기 위해서 BeautifulSoup을 사용해서 일단 먼저 해당 태그를 찾아야 합니다. 이름을 사용해서 해당 태그를 찾고자 하는 경우에는 간단히 아래와 같이 코딩 합니다. 즉, soup 다음에 점(.)을 찍고, 찾고자 하는 태그의 이름 (여기서는 title)을 입력합니다.

In: soup.title

위의 코드가 반환하는 값은 아래와 같습니다.

Out: <title> Online Data Scraping Test</title>

즉 title 태그의 전체 내용을 반환합니다. 이중에서 우리가 최종적으로 추출하고자 하는 정보는 title 태그의 시작 태그와 종료 태그 사이에 저장되어 있는 텍스트 정보입니다. 해당 정보만을 추출하기 위해서 아래와 같이 text라는 키워드를 사용합니다.

In: soup.title.text # 태그 이름을 사용해서 태그를 찾고, 텍스트 정보 추출하기

Out: 'Online Data Scraping Test'

이번에는 소스 코드에 저장되어 있는 태그들 중에서 첫번째 p태그를 이름을 이용해서 찾아 보도록 하겠습니다. 아래와 같이 할 수 있습니다.

In: soup.p

위의 코드가 반환하는 값은 아래와 같습니다. 즉, 우리가 원하는 첫번째 p 태그가 찾아진 것입니다.

Out: <p class="title">HTML code for the lecture</p>

그런데, html에 저장되어 있는 소스 코드를 보면 p라는 이름을 갖는 태그가 두개 이상 존재하는 것을 알 수 있습니다. 이렇게 동일한 이름을 갖는 태그가 두개 이상인 경우에 우리가 찾고자 하는 태그가 첫번째 태그가 아닌 경우에는 어떻게 할 수 있을까요? 이러한 경우에는 태그의 이름을 사용할 수 없습니다. 위의 코드에서 본 것 처럼 동일한 이름을 갖는 태그가 여러개 존재하는 경우, 이름을 사용하게 되면 가장 먼저 출현한 태그에만 접근하게 됩니다 (title 태그의 경우에는 title이라는 이름을 갖는 태그가 하나이기 때문에 이름을 사용해서 쉽게 해당 태그를 찾을 수 있습니다). 동일한 이름을 갖는 태그가 여러개 존재하는 경우에는 BeautifulSoup에서 제공되는 find()나 find\_all() 함수를 사용합니다.

**태그 찾기: 2) find() 또는 find\_all() 사용하기**

find()나 find\_all() 함수는 비슷한 방식으로 작동합니다. 즉, 찾고자 하는 태그에 대한 조건을 함수의 인자로 전달하여, 해당 조건을 만족하는 태그를 찾습니다. 하지만, find() 함수는 특정한 조건을 만족시키는 단 하나의 태그를 찾는 것이고, find\_all() 함수는 해당 조건을 만족시키는 (소스 코드에 존재하는) 모든 태그를 다 찾는 것입니다. find\_all()은 해당 조건을 만족 시키는 모든 태그를 찾아서 list의 형태로 반환합니다. find() 함수 같은 경우에는 인자로 전달된 조건을 만족하는 태그가 소스 코드에 여러개 존재하는 경우 가장 먼저 출현하는 태그만을 찾게됩니다. 따라서, 특정 조건을 만족시키는 태그가 소스 코드안에 하나 밖에 없는 경우는find()함수를, 2개 이상 존재하는 경우는 find\_all()함수를 사용하여 반환되는 리스트에 대해서 인덱싱 기능을 사용하여 원하는 태그를 찾을 수 있습니다.

가장 간단하게는 태그의 이름을 조건으로 전달할 수 있습니다 (아래 참고).

In: soup.find('title')

위와 같은 경우, title이라고 하는 태그의 이름을 찾고자 하는 태그의 조건으로 find()함수의 인자로 전달하였습니다. 이렇게 하면 find() 함수는 입력 받은 조건을 만족하는 태그를 찾게 됩니다. 즉, 이름이 title인 태그를 찾는 것입니다. 해당 조건을 만족하는 태그는 아래와 title 태그 밖에 없기 때문에 위의 코드가 반환하는 결과는 아래와 같습니다.

Out: <title>Online Data Scraping Test</title>

앞에서 했던 것과 마찬가지로 시작 태그와 종료 태그 사이에 있는 텍스트 정보만을 추출하기 위해서는 아래와 같이 text 키워드를 붙여 줍니다.

In: soup.find('title').text

태그의 이름만을 찾고자 하는 태그의 조건으로 제공하는 경우에는 앞에서 살펴본 이름을 사용해서 태그를 찾는 방법과 동일하게 작동합니다. 하지만, find() 함수 (또는 find\_all()함수)에는 이름 이외에 태그가 갖고 있는 추가적인 정보를 조건으로 입력할 수 있습니다. 이를 위해서 태그가 이름 이외에 갖는 추가적인 정보에는 무엇이 있는지 알아야 합니다. 아래의 태그를 예로 살펴보겠습니다.

<span class="red\_price">35.6</span>

위 태그의 이름은 span입니다. 하지만 span 태그의 시작 태그를 보면 이름 이외에 추가적인 내용이 포함되어 있는 것을 알 수 있습니다. class="red\_price"가 바로 그것입니다. 이러한 정보를 태그의 속성 (attribute) 정보라고 합니다. 사람의 경우, 사람의 이름 이외에도 해당 사람의 특성을 구분 짓는 추가적인 속성 정보를 갖는데, 태그의 속성 정보도 이와 비슷한 역할을 한다고 생각할 수 있습니다. 사람의 속성 정보에는 나이, 성별, 직장 이름, 교육 정도 등이 존재합니다. 즉, 사람의 이름과 그외의 여러가지 속성 정보가 그 사람의 특성을 정의하게 됩니다. 태그도 마찬가지라고 생각할 수 있습니다. 위의 예에서는 span 태그가 class라고 하는 속성을 갖고, 그 속성의 값이 "red\_price"가 되는 것입니다. 우리는 이러한 속성 정보 (즉, 어떠한 속성을 갖고, 해당 속성의 구체적인 값이 무엇인지)를 사용해서 우리가 찾고자 하는 태그를 좀 더 명확하게 찾을 수 있습니다.

예를 들어서, html\_test.html 파일의 소스 코드에는 아래와 같이 서로 다른, 하지만 이름이 동일한 3개의 span 태그가 있습니다.

<span class="red\_price">35.6</span>

<span class="blue\_price">43.2</span>

<span method="manual">24.3</span>

이 중에서 두번째 span태그에 접근하기 위해서는 어떻게 하면 될까요? 일단 태그의 이름만을 이용해서는 해당 태그에 접근할 수 없습니다. 우리가 찾고자 하는 태그가 동일 이름을 갖는 여러개의 태그들 중에서 첫번째 태그가 아니기 때문입니다. 그리고 find()함수를 사용하더라도 태그의 이름만을 찾고자 하는 태그의 조건으로 사용해서는 안됩니다. 왜냐하면 이름을 사용하는 경우와 마찬가지로 첫번째 태그만을 찾을 수 있기 때문입니다. 두번째 span 태그의 경우 이름 이외에도 추가적인 속성이 있고, 그 속성의 값이 고유하기 때문에 해당 정보를 이용해서 찾을 수가 있습니다. 즉, 두번째 span 태그는 class라는 속성이 있습니다. 첫번째 span 태그도 class 속성이 있지만, 그 값이 두번째 span 태그의 class 속성 값 (="blue\_price")과 다르기 때문에 해당 속성 정보를 사용해서 우리는 두번째 속성 태그를 직접 찾을 수 있습니다. 이러한 경우는 아래와 같이 코딩합니다.

In: soup.find('span',attrs={'class':'blue\_price'})

find()함수에는 첫번째 인자로 찾고자 하는 태그의 이름을 제공하고 ('span'), 두번째 인자로로 해당 태그의 추가적인 속성 정보를 제공합니다. 속성 정보를 인자로 제공하기 위해서 attrs라는 파라미터 이름을 사용합니다. attrs는 attributes를 의미합니다. 그리고 해당 파라미터는 사전 형태의 데이터를 입력 받습니다. 즉, 찾고자 하는 태그의 속성 이름이 키(key)가 되고, 속성의 값이 키의 값 (value)이 됩니다. soup.find('span',attrs={'class':'blue\_price'})는 span이라는 이름을 갖는 태그들 중에서 class 속성을 가지고 있고, 그 속성의 구체적인 값이 'blue\_price'인 태그를 찾아라는 뜻입니다. 해당 코드가 반환하는 값은 아래와 같습니다.

Out: <span class="blue\_price">43.2</span>

즉, 우리가 원하는 두번째 span 태그가 찾아진 것입니다. 특정한 태그를 찾은 다음에 태그가 저장하고 있는 텍스트 정보 (위의 예에서는 43.2)를 추출하기 위해서는 text키워드를 사용하면 됩니다. 아래와 같이 할 수 있습니다.

In: soup.find('span',attrs={'class':'blue\_price'}).text

Out: '43.2'

**find\_all() 함수 사용하기**

속성 정보를 사용해서 우리가 원하는 특정 태그를 찾을 수 있다면, find() 함수를 사용하면 됩니다. 하지만, 속성 정보까지 동일한 여러개의 태그가 존재하는 경우가 있는데 이러한 경우에 find()함수를 사용하게 되면, 첫번째로 출현하는 태그만을 찾을 수 있습니다. html\_test.html에 저장되어 있는 태그들 중에서 두번째 p태그와 세번째 p태그가 그러한 경우입니다. 두 p태그 모두 class 속성을 가지고 있고, 그 값이 동일하게 "description"입니다. 이러한 경우 아래와 같이 find() 함수를 사용하면 해당 p 태그들 중에서 첫번째로 나오는 p 태그에만 접근할 수 있습니다.

In: soup.find('p', attrs={'class':'description'})

만약 우리가 접근하고자 하는 p 태그가 해당 조건 만족하는 여러개의 p 태그들 중에서 두번째 이후에 나오는 p 태그라면, find()가 아닌 find\_all() 함수를 사용해야 합니다 (아래 참고).

In: soup.find\_all('p', attrs={'class':'description'})

위의 코드는 이름이 p이면서 class 속성을 갖고 있고, 그 속성의 값이 'description'인 태그를 모두 찾습니다. 그리고 html\_test.html 파일의 소스 코드에는 그러한 p태그가 두개 존재합니다. 즉, 두번째 p태그와 세번째 p 태그입니다. 이 중에서 우리가 찾고자 하는 p태그가 세번째 인 경우, 일단 find\_all()을 사용해서 해당 조건을 만족하는 태그를 모두 찾은 후 그 다음 find\_all() 함수가 반환하는 리스트 데이터에 인덱싱 방법을 적용하여 우리가 원하는 태그를 찾을 수 있습니다.

soup.find\_all('p', attrs={'class':'description'})은 인자로 전달된 조건을 만족하는 두 개의 p 태그에 대한 정보를 리스트 형태로 반환합니다. 우리가 찾고자 하는 p 태그는 그 중 두번째이기 때문에 인덱스 값 1을 사용해서 접근할 수 있습니다.

In: soup.find\_all('p', attrs={'class':'description'})[1]

아래와 같이 우리가 원하는 태그의 정보가 반환됩니다.

Out: <p class="description">한국어로 된 정보를 원한다면 <a class="BS\_Korean" href="http://coreapython.hosting.paran.com/etc/beautifulsoup4.html"> BS4 for beginners (한글버전)</a>을 이용하세요</p>

그렇다면 아래와 같은 코드가 반환하는 값은 무엇일까요?

In: soup.find\_all('span')[-1].text

위의 코드가 반환하는 값을 알기 위해서는 먼저 soup.find\_all('span')가 반환하는 값이 무엇인지 알아야 합니다. 해당 함수는 소스 코드에 포함되어 있는 모든 span 태그를 찾습니다. 소스 코드에는 총 3개의 span 태그가 있습니다. 그리고 이러한 결과가 아래와 같이 리스트 형태로 반환됩니다.

Out: [<span class="red\_price">35.6</span>, <span class="blue\_price">43.2</span>, <span method="manual">24.3</span>]

여기에 -1 인덱스를 적용하게 되면, 제일 마지막 원소인 3번째 span 태그에 접근하게 됩니다. soup.find\_all('span')[-1]가 반환하는 값은 <span method="manual">24.3</span>, 즉, 우리가 찾고자 하는 3번째 태그가 됩니다. 여기에서 태그가 저장하고 있는 텍스트 정보인 24.3을 추출하기 위해서는 text 키워드를 사용합니다. 결과적으로 soup.find\_all('span')[-1].text는 24.3을 반환합니다 (사실, 해당 span 태그의 경우에는 method라는 속성을 갖고 그 값이 고유하기 때문에 find() 함수를 사용해서 직접 접근할 수 있습니다. 하지만, 여기에서는 find\_all() 함수의 설명을 위해 find\_all()함수를 사용해서 접근해 보았습니다).

만약 우리가 찾고자 하는 태그가 속성 정보를 두개 이상 가지고 있다면 아래와 같이 추가적으로 입력해서 해당 태그를 찾을 수 있습니다.

soup.find('tag\_name', attrs={'attribute1':'value1', 'attribute2':'value2'})

**태그 찾기: 3) BeautifulSoup의 Navigation 기능 사용하기**

requests.get()을 이용해서 다운로드한 소스 코드에 존재하는 많은 태그를 지금까지 배운 방법을 통해서 찾을 수 있습니다. 하지만, 간혹 앞의 방법들을 사용해서 찾을 수 없는 태그가 있는데, 그러한 경우에는 BeautifulSoup에서 제공되는 navigation 기능을 사용해야 합니다. Navigation 이란 특정한 태그를 중심으로 앞, 뒤 태그 등으로 이동해서 또 다른 태그를 찾는 방법을 말합니다. BeautifulSoup이 어떻게 소스 코드를 navigate하는지를 알기 위해서는 소스 코드, 즉, 태그들이 어떻게 구성되어 있는지를 먼저 이해할 필요가 있습니다. 설명을 위해 html\_test.html파일의 head태그를 보겠습니다. 해당 태그는 아래와 같이 구성되어 있습니다.

<head>

<title>Online Data Scraping Test</title>

</head>

위의 예에서 볼 수 있듯이 소스 코드를 구성하고 있는 태그들은 위계적으로(hierarchically) 구성되어 있습니다. 태그들 간의 관계는 그 구조에 따라서 부모 (parent), 자식(contents)[[5]](#footnote-5), 형제자매 (siblings)로 이뤄져 있습니다. 위의 소스 코드를 예로 설명해 보겠습니다. 첫번째로 head태그와 title 태그 간의 관계를 보면, title 태그가 head 태그 안에 포함된 것을 볼 수 있는데, 이러한 경우 head 태그는 title 태그의 부모 태그가 되고 반대로 title 태그는 head 태그의 자식 태그가 됩니다.

Navigation 기능 설명을 위해서, find()나 find\_all() 함수를 통해서 title태그에 직접적으로 접근할 수 없다라고 가정하겠습니다. 하지만, head태그 (즉, title의 부모 태그)에는 직접적으로 접근할 수 있다라고 한다면, 이러한 상황에서는 find()함수를 통해서 head 태그에 먼저 접근하고 navigation 기능을 통해서 head태그의 자식 태그인 title태그에 접근하는 것이 가능합니다. BeautifulSoup에서는 특정 태그의 자식 태그에 접근하기 위해서 contents 라는 키워드를 사용합니다. head 태그를 찾은 다음에 그 뒤에 .contents를 붙여서 html 태그의 자식 태그인 title 태그 에 접근할 수 있습니다.

이를 위해 아래와 같은 코드를 실행하면,

In: soup.head.contents

soup.head.contents는 head 태그가 갖고 있는 모든 자식 원소(elements)를 리스트의 형태로 반환합니다. 즉, 그 결과는 아래와 같습니다.

Out: ['\n', <title>Online Data Scraping Test</title>, '\n']

위의 결과를 보면 head의 자식 원소에는 태그들만 아니라는 것을 알 수 있습니다. 즉, \n 문자가 들어 있습니다. 이는 html 태그가 포함하고 있는 태그 뿐만 아니라 다른 문자들도 자식 원소에 해당이 된다는 것을 의미합니다. 위의 html\_test.html를 보면 <head> 다음에 줄바꿈 (new line)이 포함되어 있는 것을 알 수 있습니다. 즉, <head> 다음에 한줄 밑으로 내려 간다음에 <title> 태그가 시작됩니다. 이러한 공백문자 (\n)도 한 태그의 자식이 되는 것입니다. 그래서soup.head.contents 가 리턴하는 결과의 첫번째 원소와 세번째 원소가 ‘\n’이 됩니다. 리턴된 결과에서 우리가 관심있는 것은 titile 태그 즉, 두번째 원소가 됩니다. 따라서 우리는 해당 태그에 다음과 같이 인덱싱 방법을 사용해서 접근할 수가 있습니다.

In: soup.head.contents[1]

위의 코드는 다음 값을 반환합니다.

Out: <title>Online Data Scraping Test</title>

반대로 우리는 parent라는 키워드를 사용해서 특정 태그의 부모 태그에 접근할 수가 있습니다. 예를 들어서 title 태그에 먼저 접근한 다음에 .parent 키워드를 사용해서 title의 부모 태그인 head에 접근합니다 (아래 코드 참고).

In: soup.title.parent

위 코드의 결과는 아래와 같습니다. 즉, html 태그 (body 태그의 부모 태그)의 모든 내용이 반환됩니다.

Out: <head>

<title>Online Data Scraping Test</title>

</head>

**형제자매 태그에 접근하기**

이번에는 특정한 태그의 형제자매 (sibling) 태그에 접근하는 방법에 대해서 살펴보도록 하겠습니다. html\_test.html 파일 소스 코드에서 <body> 태그 안에 포함되어 있는 여러개의 p 태그들은 다음과 같이 배열 되어 있습니다.

<body>

<p class="title">HTML code for the lecture</p>

<p class="description">This code will be used as HTML source code for the BS lecture. For more detailed information about BS, you can click<a class="BS\_English" href="http://www.pythonforbeginners.com/python-on-the-web/beautifulsoup-4-python/">BS4 for beginners</a>

</p>

<!-- 이후 생략 -->

</body>

위의 소스 코드에서 첫번째 p 태그와 두번째 p 태그를 살펴보면, 두 태그는 들여쓰여진 정도가 같은데, 이는 두 태그가 서로 형제자매(sibling)이라는 것을 의미합니다. 즉, 같은 부모 (body 태그) 태그를 가지고 있는 것이다. 만약에 우리가 첫번째 p 태그에 find() 함수를 사용해서 접근할 수 있고, 두번째 태그는 find()를 사용해서 접근할 수 없다라고 한다면, 그러한 경우에 find() 함수를 사용해서 첫번째 p 태그에 먼저 접근한 다음에 navigation 기능을 사용해서 두번째 p 태그에 접근할 수 있습니다. 특정 태그의 다음 형제자매 태그에 접근하기 위해서는 BeautifulSoup에서 제공되는 next\_sibiling키워드를 사용하고 이전 형제자매 태그에 접근하기 위해서는 previous\_sibling 키워드를 사용할 수 있다. 첫번째 p 태그를 찾은 다음에 그 다음 p 태그에 접근하기 위해서는 다음과 같이 코드를 작성할 수 있다.

In: soup.find('p',attrs={'class':'title'}).next\_sibiling

하지만 이렇게만 코드를 작성하면 그 결과로 리턴되는 값은 ‘\n’이 됩니다. 왜냐하면 위에서 설명한 것 처럼 태그만이 아니라 문자도 (공백문자 포함) 하나의 원소로 간주되기 때문입니다. 즉, ‘\n’도 첫번째 p 태그의 형제자매가 되는 것입니다. 두번째 p 태그에 접근하기 위해서는 ‘\n’을 찾은 다음 다시 한번 .next\_sibling을 붙여 줘야 합니다. 즉, 두번째 p 태그는 ‘\n’의 다음 형제자매가 되는 것입니다.

In: soup.find('p',attrs={'class':'title'}).next\_sibling.next\_sibling

**get() 함수 사용하기**

우리가 원하는 정보는 보통 시작 태그와 종료 태그 사이에 텍스트 형태로 저장되어 있습니다. 하지만, 우리가 원하는 정보가 태그의 속성 정보로 저장되어 있는 경우가 있습니다. 설명을 위해 html\_test.html 파일의 소스 코드에 저장되어 있는 첫번째 <a> 태그를 보도록 하겠습니다 (a 태그는 웹페이지에 대한 링크를 만드는 역할을 합니다. 이를 위해 href라는 속성의 값으로 해당 링크를 클릭했을 때 이동하는 웹페이지의 URL 주소를 저장합니다).

<a class="BS\_English" href="http://www.pythonforbeginners.com/python-on-the-web/beautifulsoup-4-python/"> BS4 for beginners</a>

시작 태그와 종료 태그 사이에 저장된 텍스트 정보는BS4 for beginners입니다. 하지만, 우리가 원하는 정보는 이 텍스트 정보가 아니라 시작 태그에 저장되어 있는 href 속성의 값인 특정 웹페이지의 URL 주소라고 하겠습니다. 이를 위해서는 해당 a 태그를 찾은 다음에 text 키워드를 붙이는 것이 아니라, get() 함수를 사용합니다. 이 get() 함수는 BeautifulSoup에서 제공되는 함수로, requests 모듈에서 제공되었던 get() 함수와 다른 함수입니다. a 태그의 href 속성의 값을 추출하기 위해서는 다음과 같이 코딩합니다 (여기서는 간단하게 태그의 이름을 사용해서 첫번째 a 태그에 접근하였습니다. 즉, soup.a 부분). get() 함수에 우리가 추출하고자 하는 값을 저장하고 있는 속성의 이름을 입력해 줍니다. 여기서는 우리가 원하는 URL 정보가 href 속성의 값으로 저장되어 있기 때문에 해당 속성의 이름인 href를 문자열의 형태로 get() 함수의 인자로 입력하였습니다.

soup.a.get('href')

위의 코드가 반환하는 값은 'http://www.pythonforbeginners.com/python-on-the-web/beautifulsoup-4-python/' 이 됩니다.

1. 우리는 이러한 컴퓨터를 보통 서버 (server) 컴퓨터 혹은 간단히 서버라고 합니다. 그리고 여러분의 컴퓨터와 같이 서버에 접속하는 컴퓨터를 클라이언트 (client)라고 합니다. [↑](#footnote-ref-1)
2. 보다 자세한 내용은 <https://requests.readthedocs.io/en/master/>를 참고하세요. [↑](#footnote-ref-2)
3. https://www.crummy.com/software/BeautifulSoup/bs4/doc/ [↑](#footnote-ref-3)
4. 자세한 내용은 https://www.crummy.com/software/BeautifulSoup/bs4/doc/#specifying-the-parser-to-use 을 참고하세요. [↑](#footnote-ref-4)
5. BeautifulSoup에서 children은 다른 의미를 갖습니다. 자세하나 내용은 https://www.crummy.com/software/BeautifulSoup/bs4/doc/을 참고하세요. [↑](#footnote-ref-5)