웹 스크래핑 기초

강의 목표

- 웹에서 데이터를 스크랩하는 방법을 배울 예정
- Beautiful Soup 라이브러리를 사용하여 진행할 예정
- 많은 웹 사이트가 API를 제공하지 않거나, 제공하는 API로 원하는 작업을 수행하기 어려울 때 웹 스크래핑을 고려
- 웹 스크래핑은 웹사이트의 HTML 코드를 분석하여 필요한 정보를 추출하는 과정

웹 스크래핑 & 웹 크롤링 정의

웹 스크래핑

- 웹 스크래핑은 특정 웹 페이지로부터 필요한 데이터를 추출하는 과정
- 이 방식은 주로 구조화된 데이터를 수집하는 데 사용
- 예를 들어, 온라인 쇼핑 사이트에서 제품 정보, 가격, 리뷰 등을 수집할 때 웹 스크래핑 기술 사용
  - **목적**: 웹 페이지로부터 특정 데이터를 추출하여 사용자가 원하는 형식으로 가공
  - **도구**: Beautiful Soup, Scrapy, Selenium 등 다양한 파이썬 라이브러리가 사용
  - **방식**: HTML의 특정 요소를 선택하여 그 안의 데이터만을 추출

웹 크롤링

- 웹 크롤링은 인터넷상의 웹 페이지를 체계적으로 탐색하여 정보를 수집하는 과정

- 크롤링은 스크래핑보다 더 광범위한 데이터 수집을 목표
  - **목적**: 웹 페이지의 링크를 따라가며 정보를 수집하고, 검색 엔진의 데이터베이스에 저장하여 사용자 검색에 활용
  - **도구**: Googlebot과 같은 검색 엔진 크롤러 또는 사용자 지정 크롤러
  - **방식**: 웹의 링크 구조를 따라 전체 웹 사이트 또는 특정 부분의 페이지를 체계적으로 방문하고 정보를 수집

차이점

- **범위**
  - 스크래핑은 특정 페이지에서 필요한 데이터를 선택적으로 추출하는 반면,
  - 크롤링은 웹 사이트 전체 또는 큰 범위의 페이지를 대상으로 함
- **목적**
  - 스크래핑은 데이터 추출에 중점을 두고,
  - 크롤링은 데이터 인덱싱 및 링크 탐색에 초점을 맞춤
- **결과 활용**
  - 스크래핑 결과는 데이터 분석, 시장 조사 등 특정한 목적으로 직접 사용되며,
  - 크롤링 결과는 주로 검색 엔진 최적화(SEO)나 다른 웹 서비스의 기반 데이터로 사용

Beautiful Soup

- Beautiful Soup은 HTML과 XML 파일로부터 데이터를 추출하기 위한 파이썬 라이브러리
- 사용자가 선호하는 파서와 함께 작동하여, 파싱 트리를 탐색하고, 검색하며, 수정하는 방식을 제공
  - "파서"는 문서를 분석하고 구조화된 형태로 변환하는 프로그램을 의미
  - 파서는 원시 텍스트 형태의 HTML/XML 문서를 읽고, 그 구조를 분석하여 계층적인 트리 구조로 변환
- 웹 페이지에서 원하는 정보를 쉽게 가져올 수 있게 해주며, 복잡하고 규칙이 없는 웹의 데이터를 구조화된 형태로 변환하는 데 매우 유용

Beautiful Soup Documentation — Beautiful Soup 4.12.0 documentation

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

라이브러리 설치

- Beautiful Soup를 사용하기 위해 라이브러리를 설치 필요

```
pip install beautifulsoup4
```

웹 스크래핑 기본 프로세스

1. **HTML 가져오기**: 웹 페이지의 HTML을 가져옵니다. 이는 `requests` 라이브러리를 통해 수행
2. **Beautiful Soup 객체 생성**: 가져온 HTML로부터 Beautiful Soup 객체 생성. 이 객체를 통해 데이터를 탐색하고 조작
3. **데이터 추출**: 태그 이름, 속성 등을 사용하여 필요한 데이터 추출

Beautiful Soup 라이브러리를 사용하여 HTML 문서 파싱 실습

- 아래 예제에서는 HTML 문서를 파싱하고, 특정 태그를 검색하며, 내용을 수정하는 과정

```
from bs4 import BeautifulSoup html_doc = """ <html> <head> <title>The Dormouse's story</title> </head> <body> <div data-role="page" data-last-modified="2022-01-01" data-foo="value">This is a div with data attributes. </div> <p class="title"><b>The Dormouse's story</b></p> <p class="story">Once upon a time there were three little sisters; and their names were <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>, <a href="http://example.com/lacie" class="sister" id="link2" data-info="more info">Lacie</a> and <a href="http://example.com/tillie" class="sister" id="link3" data-info="even more info">Tillie abcd</a> ; and they lived at the bottom of a well.</p> <p class="story">...</p> </body> </html> """ soup = BeautifulSoup(html_doc, 'html.parser') print(soup.prettify())
```

파서 종류

html.parser

- **표준 Python 라이브러리**: `html.parser`는 Python의 표준 라이브러리에 포함되어 있어서 별도의 설치가 필요 없음
- **간단하고 쉬운 사용**: 비교적 단순한 HTML 파싱에 적합하며, 복잡하지 않은 웹 페이지를 처리할 때 사용하기 좋음
- **속도와 유연성**: `html.parser`는 `lxml`이나 `html5lib`보다 느릴 수 있으며, 매우 복잡하거나 깨진 HTML에 대해서는 `lxml`이나 `html5lib`만큼 강력하지 않음

lxml

- **속도**: `lxml`은 매우 빠른 HTML과 XML 파서로, 대량의 데이터나 매우 복잡한 웹 페이지를 처리하는 데 적합
- **유연성과 기능**: `lxml`은 광범위한 XML 처리 기능을 지원하며, 손상된 HTML 문서에 대해서도 강력한 파싱 성능 제공
- **설치 필요**: `lxml`은 별도의 외부 라이브러리이므로 사용하기 위해 설치 필요

```
pip install lxml
```

html5lib

- **웹 브라우저 모방**: `html5lib`는 웹 브라우저가 HTML5를 처리하는 방식을 모방하여, 웹 브라우저가 작동하는 방식과 매우 유사하게 HTML 문서를 해석하고, 심지어 잘못된 마크업도 올바르게 처리할 수 있음을 의미
- **오류 처리**: 이 파서는 잘못된 HTML을 웹 브라우저처럼 우아하게 처리할 수 있어, 심각하게 깨진 HTML 문서에서도 작동 가능, `html.parser`나 `lxml`보다 더 나은 오류 관리 능력을 제공
- **성능**: `html5lib`는 정확도를 위해 성능이 떨어질 수 있고, 웹 브라우저처럼 동작하기 때문에 처리 속도가 느리지만, 가장 정확한 HTML 해석을 제공

```
pip install html5lib
```

파서 종류

html.parser

- **표준 Python 라이브러리**: `html.parser`는 Python의 표준 라이브러리에 포함되어 있어서 별도의 설치가 필요 없음
- **간단하고 쉬운 사용**: 비교적 단순한 HTML 파싱에 적합하며, 복잡하지 않은 웹 페이지를 처리할 때 사용하기 좋음
- **속도와 유연성**: `html.parser`는 `lxml`이나 `html5lib`보다 느릴 수 있으며, 매우 복잡하거나 깨진 HTML에 대해서는 `lxml`이나 `html5lib`만큼 강력하지 않음

lxml

- **속도**: `lxml`은 매우 빠른 HTML과 XML 파서로, 대량의 데이터나 매우 복잡한 웹 페이지를 처리하는 데 적합
- **유연성과 기능**: `lxml`은 광범위한 XML 처리 기능을 지원하며, 손상된 HTML 문서에 대해서도 강력한 파싱 성능 제공
- **설치 필요**: `lxml`은 별도의 외부 라이브러리이므로 사용하기 위해 설치 필요

```
pip install lxml
```

html5lib

- **웹 브라우저 모방**: `html5lib`는 웹 브라우저가 HTML5를 처리하는 방식을 모방하여, 웹 브라우저가 작동하는 방식과 매우 유사하게 HTML 문서를 해석하고, 심지어 잘못된 마크업도 올바르게 처리할 수 있음을 의미
- **오류 처리**: 이 파서는 잘못된 HTML을 웹 브라우저처럼 우아하게 처리할 수 있어, 심각하게 깨진 HTML 문서에서도 작동 가능, `html.parser`나 `lxml`보다 더 나은 오류 관리 능력을 제공
- **성능**: `html5lib`는 정확도를 위해 성능이 떨어질 수 있고, 웹 브라우저처럼 동작하기 때문에 처리 속도가 느리지만, 가장 정확한 HTML 해석을 제공

```
pip install html5lib
```

데이터 구조 탐색

- Beautiful Soup 객체를 사용하여 문서 내 데이터를 다양한 방식으로 접근

```
# <title> 태그 접근 print(soup.title) # <title>The Dormouse's story</title>
# <title> 태그의 이름 print(soup.title.name) # 'title' # <title> 태그의 텍스트 내용 print(soup.title.string) # 'The Dormouse's story' # <title> 태그의 부모 태그 이름 print(soup.title.parent.name) # 'head' # 첫 번째 <p> 태그 접근 print(soup.p) # <p class="title"><b>The Dormouse's story</b></p> # <p> 태그의 클래스 속성 값 print(soup.p['class']) # ['title'] # 첫 번째 <a> 태그 접근 print(soup.a) # <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a> # 모든 <a> 태그를 찾아 리스트로 반환 print(soup.find_all('a')) # 출력: 모든 <a> 태그 리스트 # id가 "link3"인 <a> 태그 찾기 print(soup.find(id="link3")) # <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

웹 페이지에서 모든 URL 추출

```
for link in soup.find_all('a'): print(link.get('href')) # http://example.com/elsie # http://example.com/lacie # http://example.com/tillie
```

Beautiful Soup 메서드

find_all() 메서드

- `find_all()` 메소드는 지정된 태그의 모든 자손을 검색하고, 설정된 필터에 맞는 모든 결과를 검색하여 `리스트로 반환`

사용 예제

- 태그 이름으로 검색

```
soup.find_all("title") # [<title>The Dormouse's story</title>]
```

- **태그의 속성을 이용한 검색**
  - 첫 번째 인자는 태그 이름, 두 번째 인자는 클래스 이름

```
soup.find_all("p", "title") # [<p class="title"><b>The Dormouse's story</b
></p>] # soup.find_all("p", class_="title") 코드로 같은 동작 가능 # class는
예약어이기 때문에 class_로 사용
```

- 모든 `<a>` 태그 검색

```
soup.find_all("a") # [<a class="sister" href="http://example.com/elsie" id
="link1">Elsie</a>, # <a class="sister" href="http://example.com/lacie" id
="link2">Lacie</a>, # <a class="sister" href="http://example.com/tillie" i
d="link3">Tillie</a>]
```

- **태그의 ID 속성을 이용한 검색**

```
soup.find_all(id="link2") # [<a class="sister" href="http://example.com/la
cie" id="link2">Lacie</a>]
```

- 정규 표현식을 이용한 문자열 검색
  - find 메서드의 string 파라미터 사용

```
import re soup.find(string=re.compile("sisters")) # 'Once upon a time ther
e were three little sisters; and their names were\n'
```

find() 메서드

- `find_all()` 메소드는 문서 전체를 검색하여 모든 결과를 찾지만, 때로는 단 하나의 결과만 찾고 싶을 때 `find()` 메소드를 사용
- 만약 문서에 단 하나의 `<body>` 태그만 존재한다면, 전체 문서를 스캔하는 것은 시간 낭비
- 이런 경우 `find_all()` 대신, `find()`를 사용 가능

코드 비교

- 다음 두 줄의 코드는 거의 동일
- 유일한 차이점은 `find_all()`이 단일 결과를 포함하는 리스트를 반환하는 반면, `find()`는 결과 자체를 반환

```
soup.find_all('title', limit=1) # [<title>The Dormouse's story</title>] so
up.find('title') # <title>The Dormouse's story</title>
```

반환 값

- `find_all()`이 아무 것도 찾지 못하면 `빈 리스트`를 반환
- `find()`가 아무 것도 찾지 못하면 `None`을 반환

```
print(soup.find("abcdefg")) # None
```

태그 이름을 사용한 탐색

- `soup.head.title` 같은 방법은 태그 이름을 사용하여 반복적으로 `find()`를 호출하는 방식으로 작동

```
soup.head.title # <title>The Dormouse's story</title> soup.find("head").fi
nd("title") # <title>The Dormouse's story</title>
```

get_text() 메서드

- 문서나 태그 안에 있는 인간이 읽을 수 있는 텍스트만을 원할 경우, `get_text()` 메서드를 사용
- 이 메서드는 문서나 특정 태그 아래에 있는 모든 텍스트를 단일 유니코드 문자열로 반환
- 기본적으로 페이지의 모든 텍스트 출력

- 기본적으로 `get_text()`를 호출하면 모든 텍스트 조각이 연결되어 하나의 문자열로 반환

```
from bs4 import BeautifulSoup markup = '<a href="http://example.com/">\nI linked to <i>example.com</i>\n</a>' soup = BeautifulSoup(markup, 'html.parser') print(soup.get_text()) # 출력: '\nI linked to example.com\n' print(soup.i.get_text()) # 출력: 'example.com'
```

CSS selectors

- Beautiful Soup 및 Tag 객체는 `.css` 속성을 통해 CSS 선택자를 지원
- 실제 선택자 구현은 PyPI에서 soupsieve로 사용 가능한 Soup Sieve 패키지에 의해 처리됨
- pip을 통해 Beautiful Soup을 설치했다면 Soup Sieve도 동시에 설치되었기 때문에 추가 작업을 할 필요가 없음

select() 메서드 (=find_all())

- **인자 값**: CSS 선택자 문자열
- 반환 값: 일치하는 모든 요소의 ResultSet 객체. 만약 일치하는 요소가 없다면 빈 ResultSet 객체를 반환

```
soup.select("p.sister") # 'sister' 클래스를 가진 모든 <p> 태그 선택 soup.sel
ect("#first") # ID가 'first'인 요소 선택 soup.select("a[href]") # 'href' 속
성을 가진 모든 <a> 태그 선택
```

select_one() 메서드

- **인자 값**: CSS 선택자 문자열
- **반환 값**: 일치하는 요소 중 첫 번째 요소. 만약 일치하는 요소가 없다면 `None`을 반환

```
soup.select_one(".sister") # 'sister' 클래스를 가진 첫 번째 요소 선택 soup.s
elect_one("title") # 문서의 첫 번째 <title> 요소 선택
```

연습 예제

- **태그 이름을 사용하여 태그를 찾기**

```
# soup.css.select("title"): HTML 문서에서 <title> 태그를 찾기 soup.css.sele
ct("title") # [<title>The Dormouse's story</title>] # soup.css.select("p:n
th-of-type(3)"): 세 번째 <p> 태그를 찾기 # :nth-of-type() 선택자는 같은 유형
의 형제 중 특정 순서에 있는 요소를 선택 soup.css.select("p:nth-of-type(3)")
# [<p class="story">...</p>]
```

- **ID로 태그 찾기**

```
# soup.css.select("#link1"): id 속성이 "link1"인 요소를 찾기 soup.css.selec
t("#link1") # [<a class="sister" href="http://example.com/elsie" id="link
1">Elsie</a>] # id 속성이 "link2"인 <a> 태그 찾기 # 태그 이름과 id를 함께 사
용하여 더 구체적인 선택 가능 soup.css.select("a#link2") # [<a class="sister"
href="http://example.com/lacie" id="link2">Lacie</a>]
```

- **다른 태그 내부에 포함된 태그 찾기**

```
# soup.css.select("body a"): <body> 태그 내부의 모든 <a> 태그 찾기 soup.css.
select("body a") # [<a class="sister" href="http://example.com/elsie" id
="link1">Elsie</a>, # <a class="sister" href="http://example.com/lacie" id
="link2">Lacie</a>, # <a class="sister" href="http://example.com/tillie" i
d="link3">Tillie</a>] # <html> 태그 안의 <head> 태그 내부에 있는 <title> 태
그 찾기 soup.css.select("html head title") # [<title>The Dormouse's story</
title>]
```

- **태그를 직접 포함하는 태그 찾기**

```
# <head> 태그 바로 아래에 있는 <title> 태그 찾기 soup.css.select("head > tit
le") # [<title>The Dormouse's story</title>] # <p> 태그 바로 아래에 있는 모
든 <a> 태그 찾기 soup.css.select("p > a") # [<a class="sister" href="htt
p://example.com/elsie" id="link1">Elsie</a>, # <a class="sister" href="htt
p://example.com/lacie" id="link2">Lacie</a>, # <a class="sister" href="htt
p://example.com/tillie" id="link3">Tillie</a>]
```

- **CSS 클래스로 태그 찾기**

```
# .sister 클래스를 가진 모든 태그 찾기 soup.css.select(".sister") # [<a clas
s="sister" href="http://example.com/elsie" id="link1">Elsie</a>, # <a clas
s="sister" href="http://example.com/lacie" id="link2">Lacie</a>, # <a clas
s="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

- **속성 값으로 태그 찾기**

```
# href 속성 값이 특정 URL을 가진 <a> 태그 찾기 # 속성 선택자를 사용하여 특정 속성 값을 가진 요소 찾기 soup.css.select('a[href="http://example.com/elsie"]') # [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>]
```