BeautifulSoup

BeautifulSoup 설치

```
conda install beautifulsoup4
pip install beautifulsoup4
```

In [2]:
```
import bs4
from bs4 import BeautifulSoup
bs4.__version__
```

Out[2]:
```
'4.12.2'
```

0. 코딩 패턴

1. 조회할 HTML 내용 전달, BeautifulSoup 객체 생성
2. BeautifulSoup 객체 메소드 이용 문서 내 필요한 정보 조회
    - 태그 이용 조회
    - css selector 이용 조회
    - .표기법 이용 탐색 (tree 구조 순서대로 탐색)

1. BeautifulSoup 객체 생성

```
BeautifulSoup(html str [,파서])
```

매개변수

- html.parser : default
- lxml : 빠른 속도. html, xml(lxml만 가능) 파싱 가능

- 사용 시 install 필요
- conda install lxml

In [7]:
```
# example.html로부터 내용 조회
# beautifulsoup에 넣으려면 string 값으로 있어야 함
with open('example.html','rt', encoding='utf-8') as fr:
    docs = fr.read()

print(type(docs))   # str type
soup = BeautifulSoup(docs)    # html str을 넣어서 객체 생성
soup.find()
```

```
<class 'str'>
```

Out[7]:
```
<html>
<head>
<title>다양한 HTML 예제</title>
</head>
<body>
<h1 id="main-title">웹 크롤링 연습</h1>
<p>여기는 간단한 HTML 예제입니다. 이 내용을 BeautifulSoup과 Requests를 사용하여 가져올 수 있습니다.</p>
<h2>리스트</h2>
<ul>
<li>리스트 아이템 1</li>
<li>리스트 아이템 2</li>
<li>리스트 아이템 3</li>
</ul>
<div class="content" id="content-div">
<p>이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.</p>
<p>또 다른 문장이 있습니다.</p>
</div>
<p id="special-paragraph">이 단락은 특별한 id를 가지고 있습니다.</p>
</body>
</html>
```

2. BeautifulSoup 객체 메소드 이용 원하는 정보 검색

Tag 객체

- 하나의 태그(element)에 대한 정보를 다루는 객체
  - BeautifulSoup 조회 메소드들의 **조회 결과의 반환 타입**
  - 조회 함수들이 찾은 element가 하나인 경우 Tag 객체를, 여러개일 경우 Tag 객체들을 담은 List(ResultSet) 반환
- 주요 속성/메소드
  - **태그의 속성값 조회**

    tag객체.get('속성명') tag객체['속성명']
  - **태그 내 text값 조회**

    tag객체.get_text() tag객체.text
  - **contents 속성**: 조회한 태그의 모든 자식 요소들을 리스트로 반환

    child_list = tag객체.contents

3. 조회 함수

1. 태그 이름으로 조회

- **find_all(name=태그명, attrs={속성명:속성값,...})**
  - 이름의 모든 태그 elements들을 리스트에 담아 반환
  - 여러 이름의 태그를 조회할 경우 List에 태그명들을 묶어서 전달
  - 태그의 attribute 조건으로만 조회할 경우 name 생략

- **find(name=태그명, attrs={속성명:속성값,...})**
  - 이름의 태그 중 첫 번째 태그 element 반환

In [11]:
```
# 문서 내 모든 <p>
soup.find_all('p')
```

Out[11]:
```
[<p>여기는 간단한 HTML 예제입니다. 이 내용을 BeautifulSoup과 Requests를 사용하여 가져올 수 있습니다.</p>,
 <p>이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.</p>,
 <p>또 다른 문장이 있습니다.</p>,
 <p id="special-paragraph">이 단락은 특별한 id를 가지고 있습니다.</p>]
```

In [13]:
```
# 여러 태그를 찾고 싶을 경우 리스트로 전달
soup.find_all(['h1', 'h2'])
```

Out[13]:
```
[<h1 id="main-title">웹 크롤링 연습</h1>, <h2>리스트</h2>]
```

In [14]:
```
# class='content'인 div 태그 찾기
soup.find_all('div', {'class':'content'})
```

Out[14]:
```
[<div class="content" id="content-div">
 <p>이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.</p>
 <p>또 다른 문장이 있습니다.</p>
 </div>]
```

In [16]:
```
print(type(soup.find_all(['h1', 'h2'])))     # <class 'bs4.element.ResultSet'>
print(type(soup.find_all(['h1', 'h2'])[0]))  # <class 'bs4.element.Tag'>
```

```
<class 'bs4.element.ResultSet'>
<class 'bs4.element.Tag'>
```

In [17]:
```
# 첫 번째 p태그 element
soup.find('p')
```

Out[17]:
```
<p>여기는 간단한 HTML 예제입니다. 이 내용을 BeautifulSoup과 Requests를 사용하여 가져올 수 있습니다.</p>
```

In [31]:
```
result = soup.find('div')
print(result.get('class'))  # tag의 class 속성 조회
print(result.get_text())    # tag 내 text 조회

# 하위 element 조회
print(result.find('p').text)

# contents 사용 > 엔터까지 나옴(불필요)
print(result.contents)
for txt in result.contents:
    if isinstance(txt, bs4.element.Tag):  # 태그 타입만 실행
        print(type(txt))
```

```
['content']

이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.
또 다른 문장이 있습니다.

이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.
['\n', <p>이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.</p>, '\n', <p>또 다른 문장이 있습니다.</p>,
'\n']
<class 'bs4.element.Tag'>
<class 'bs4.element.Tag'>
```

2. css selector 이용 조회

- **select(selector='css셀렉터')**
  - css 셀럭터와 일치하는 tag들 반환
- **select_one(selector='css셀렉터')**
  - css 셀럭터와 일치하는 첫 번째 tag 반환

In [34]:
```
print(soup.select('p'))
print(soup.select('h1, h2'))  # 여러 태그 조회: 쉼표로 연결
```

```
[<p>여기는 간단한 HTML 예제입니다. 이 내용을 BeautifulSoup과 Requests를 사용하여 가져올 수 있습니다.</p>, <p>이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.</p>, <p>또 다른 문장이 있습니다.</p>, <p id="special-paragraph">이 단락은 특별한 id를 가지고 있습니다.</p>]
[<h1 id="main-title">웹 크롤링 연습</h1>, <h2>리스트</h2>]
```

In [36]:
```
# class 속성명 검색 > .클래스 속성값
# 태그명.클래스 속성값 : 특정 태그 클래스 한정
print(soup.select('.content'))
print(soup.select('div.content'))
```

```
[<div class="content" id="content-div">
<p>이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.</p>
<p>또 다른 문장이 있습니다.</p>
</div>]
[<div class="content" id="content-div">
<p>이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.</p>
<p>또 다른 문장이 있습니다.</p>
</div>]
```

In [38]:
```
# id 속성으로 조회 > #id속성값
#    태그명.#id속성값
soup.select_one('#content-div')
```

Out[38]:
```
<div class="content" id="content-div">
<p>이 부분은 클래스가 'content'인 div 안에 들어있는 내용입니다.</p>
<p>또 다른 문장이 있습니다.</p>
</div>
```

3. .표기법 (dot notation)
   - dom tree 구조의 계층 순서대로 조회
   - 위 두 방식으로 찾은 tag를 기준으로 그 주위의 element들을 찾을 때 사용

```
# 자식 태그(노트) 표현
#    parent > child ==> 찾는 대상: child
result = soup.select('#animal2 > .animal')

# result = soup.select('li > a')
#    조상 자손 ==> 찾는 대상: 자손 태그
result = soup.select('ul a')    # ul:조상 -> li -> a:자손

# div 중에서 두 번째 자식 태그 조회
result = soup.select('div:nth-child(2)')
```

Requests

- HTTP 요청 처리 파이썬 패키지
- get/post 방식 모두 지원

```
pip install requests
conda install -c conda-forge requests
```

0. 코딩 패턴

1. get()/post()함수 이용 url 전달해 서버 요청
2. 응답받은 내용 처리
    - text(HTML)은 BeautifulSoup에 넣어 parsing
    - binary 파일의 경우 파일 출력을 이용해 local에 저장

1. 요청 함수 (get/post)

1. **requests.get(URL)**

- GET 방식 요청
- 매개변수
  - params : 요청 파라미터를 dictionary로 전달
  - headers : HTTP 요청 header를 dictionary로 전달
    - 'User-Agent', 'Referer' 등 헤더 설정 : client 정보
  - cookies : 쿠키 정보 전달
- return value : response(응답 결과)

2. **requests.post(URL)**

- POST 방식 요청
- 매개변수
  - datas : 요청 파라미터를 dictionary로 전달
  - files : 업로드할 파일을 dictionary로 전달
    - key: 이름, value: 파일과 연결된 InputStream(TextIOWrapper)
  - headers: HTTP 요청 header를 dictionary로 전달
    - 'User-Agent', 'Referer' 등 헤더 설정 : client 정보
  - cookies : 쿠키 정보 전달
- return value : response(응답 결과)

In [1]:
```python
import requests
from bs4 import BeautifulSoup
url = 'https://www.naver.com'

res = requests.get(url)       # 요청 ~ 응답
print(type(res.text))
print(res.status_code)

if res.status_code == 200:
    print(res.headers)
    soup = BeautifulSoup(res.text)
else:
    print('문제 발생')


soup = BeautifulSoup(res.text, "html.parser")
# print(soup)
```

```
<class 'str'>
200
{'Server': 'NWS', 'Date': 'Mon, 04 Dec 2023 13:55:16 GMT', 'Content-Type': 'text/html; charset=UTF-8', 'Trans
fer-Encoding': 'chunked', 'Connection': 'keep-alive', 'Set-Cookie': 'PM_CK_loc=95ca7532295d92eebd7e23af26d32e
020261a8f31510b2b35457b26e498a7c78; Expires=Tue, 05 Dec 2023 13:55:16 GMT; Path=/; HttpOnly', 'Cache-Contro
l': 'no-cache, no-store, must-revalidate', 'Pragma': 'no-cache', 'X-Frame-Options': 'DENY', 'X-XSS-Protectio
n': '1; mode=block', 'Content-Encoding': 'gzip', 'Strict-Transport-Security': 'max-age=63072000; includeSubdo
mains', 'Referrer-Policy': 'unsafe-url'}
```

2. Response 객체 - 응답 데이터

- get()/post() 의 요청에 대한 서버의 응답 결과를 Response에 담아 반환
  - Response의 속성을 이용해 응답 결과를 조회
- 주요 속성(Attribute)
  - **url** : 응답 받은(요청한) url

    - **status_code :** HTTP 응답 상태 코드
    - **headers :** 응답 header 정보를 dictionary로 반환

- **응답 결과 조회**
    - **text :** 응답 내용(html을 str로 반환)
    - **content :** 응답 내용(응답 결과가 binary-image, 동영상 등- 일 경우 사용하며 bytes로 반환)
    - **json() :** 응답 결과가 JSON 인 경우 dictionary로 변환해서 반환

HTTP 응답 상태코드

- 2XX: 성공
    - 200: OK
- 3XX: 다른 주소로 이동 (이사)
    - 300번대이면 자동으로 이동해 준다. 크롤링 시는 볼일이 별로 없다.
- 4XX: 클라이언트 오류 (사용자가 잘못한 것)
    - 404: 존재하지 않는 주소
- 5XX: 서버 오류 (서버에서 문제 생긴 것)
    - 500: 서버가 처리 방법을 모르는 오류
    - 503: 서버가 다운 등의 문제로 서비스 불가 상태