빅데이터 처리를 위한 파이썬

- 1991년에 첫 버전이 공개된 후 지금까지 많은 인기를 얻고 있는 스크립트 형태의 범용 프로그래밍 언어
- 프로그래머가 원하는 모든 작업을 할 수 있도록 설계한 범용 언어
- 명령형 언어이면서 스크립트 방식 지원
  - 프로그래밍적인 구현에 적합
  - 데이터 수집과 처리에 활용할 수 있는 다양한 라이브러리 제공
- 웹 서버 프로그래밍, 데이터 분석, 시스템 자동화,
  IOT 프로그래밍까지 활용 분야도 다양

파이썬
특징과 장점

- 윈도우, 리눅스, 맥에서 모두 실행할 수 있는 언어이다.
- Free and Open Source 언어이다.
- 인터프리터 언어이다.
- 객체지향 언어이다.
- 배우기 쉽고 간결하다.
- 동적(스크립트) 언어이다.
- IoT, 빅데이터, 인공지능, 블록체인 등의 실무 프로그램 구현에 많이 사용되는 고급 라이브러리도 제한 없이 사용할 수 있는 큰 장점도 가지고 있다.

빅데이터 처리 언어로서 파이썬의 장점

이해하기 쉽고 유연한 문법으로 좋은 접근성을 가짐

빅데이터 처리 언어로서 많은 커뮤니티가 형성되어 있음

가독성이 좋고, 간결하며, 스탠다드 라이브러리가 잘 갖춰져 있음

데이터 분석 관련 패키지가 최근 몇 년 사이 눈에 띄게 발전하여
NumPy, SciPy, Pandas, Matplotlib, Seaborn 등 데이터 분석 관련
오픈 소스 라이브러리들을 무상으로 사용할 수 있음

Anaconda

Anaconda는 세계에서 가장 유명한 파이썬(Python) 데이터 과학 플랫폼입니다. 모든 데이터 과학 패키지를 쉽게 설치하고 패키지, 종속성 및 환경을 관리할 수 있습니다. Anaconda는 conda, Python 및 150 개가 넘는 과학 패키지와 그 종속성과 함께 제공되는 파이썬 배포판입니다. 응용 프로그램 conda는 패키지 및 환경 관리자입니다.

파이썬과 수학·과학·데이터 분석 분야에서 필요한 거의 모든 패키지(NumPy, SciPy, Pandas, Matplotlib 등) 포함

urllib 패키지를 활용한 웹 페이지 요청

- URL 관련 라이브러리라는 의미의 패키지
- 파이썬의 표준 라이브러리

URL 문자열과 웹 요청에 관련된 모듈 5개 제공

- urllib.request — URL 문자열을 가지고 요청 기능 제공
- urllib.response — urllib 모듈에 의해 사용되는 응답 기능 관련 클래스들 제공
- urllib.parse — URL 문자열을 파싱하여 해석하는 기능 제공
- urllib.error — urllib.request에 의해 발생하는 예외 클래스들 제공
- urllib.robotparser — robots.txt 파일을 구문 분석하는 기능 제공

URL 문자열을 가지고 HTTP 요청을 수행하는 urllib.request 모듈

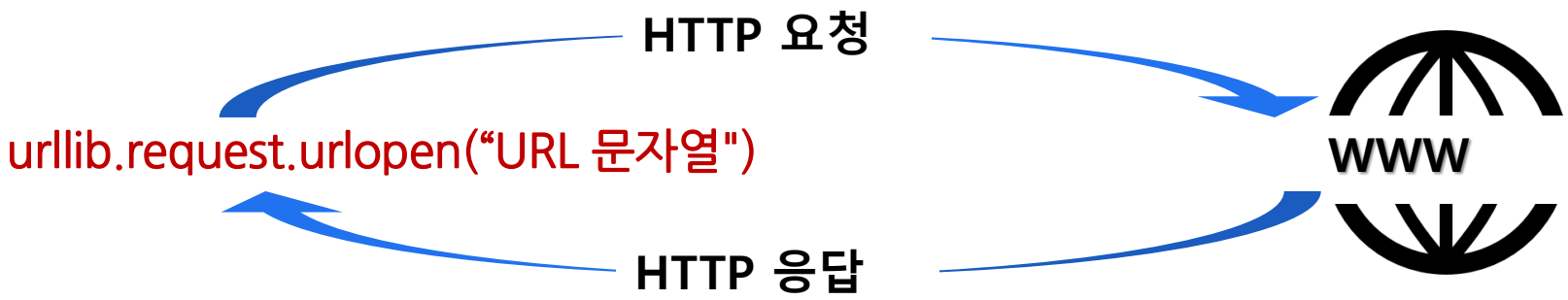
URL 문자열(주소)을 해석하는 urllib.parse 모듈

urllib 패키지를 활용한 웹 페이지 요청

[urllib.request 모듈]

- URL 문자열을 가지고 HTTP 요청을 수행
- urlopen() 함수를 사용하여 웹 서버에 페이지를 요청하고, 서버로부터 받은 응답을 저장하여 응답 객체(http.client.HTTPResponse)를 반환

```
res = urllib.request.urlopen("요청하려는 페이지의 URL 문자열")
```



http.client.HTTPResponse 클래스

- 웹 서버로부터 받은 응답을 래핑하는 객체
- 응답 헤더나 응답 바디의 내용을 추출하는 메서드 제공

- HTTPResponse.read([amt])
- HTTPResponse.readinto(b)
- HTTPResponse.getheader(name, default=None)
- HTTPResponse.getheaders()
- HTTPResponse.msg
- HTTPResponse.version
- HTTPResponse.status
- HTTPResponse.reason
- HTTPResponse.closed

http.client.HTTPResponse 객체의 read() 메서드

- read() 메서드를 실행하면 웹 서버가 전달한 데이터(응답 바디)를 바이트 열로 읽어 들임
  - 16진수로 이루어진 수열이기 때문에 읽기 어려우므로 웹 서버가 보낸 한글을 포함한 텍스트 형식의 HTML 문서의 내용을 읽을 때는 텍스트 형식으로 변환함
  - 바이트열(bytes)의 decode('문자 셋') 메서드를 실행하여 응답된 문자 셋에 알맞은 문자로 변환해야 함

res.read()

<body>₩r₩n<h1>₩xea₩
xb0₩x80₩xeb₩x82₩x98
₩xeb₩x8b₩xa4ABC</h1>
₩r₩n</body>

res.read().decode('utf-8')

<body>
<h1>가나다ABC</h1>
</body>

웹 페이지 인코딩 체크(1)

- 웹 크롤링하려는 웹 페이지가 어떠한 문자 셋으로 작성되었는지 파악하는 것이 필수
- 페이지의 소스 내용에서 <meta> 태그의 charset 정보를 체크하면 파악 가능

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>블럭스타일 태그</h1>
```

```
<!doctype html>
<!--[if lt IE 7]>   <html class="n
<!--[if IE 7]>      <html class="n
<!--[if IE 8]>      <html class="n
<!--[if gt IE 8]><!--><html class=

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compati
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
  <head>
      <title id="Title">알라딘</title>
      <meta http-equiv="Content-Type" content="text/html; charset=euc-kr">
      <meta content="Microsoft Visual Studio .NET 7.1" name="GENERATOR">
      <meta content="C#" name="CODE_LANGUAGE">
      <meta content="JavaScript" name="vs_defaultClientScript">
```

웹 페이지 인코딩 체크(2)

웹 페이지의 문자 셋 정보를 파이썬 프로그램으로도 파악할 수 있음

http.client.HTTPMessage 객체의 get_content_charset() 메서드 사용

urllib.request.urlopen() 함수의 리턴 값인 http.client.HTTPResponse 객체의 info() 메서드 호출
http.client.HTTPMessage 객체가 리턴 됨

웹 서버로부터 응답될 때 전달되는 Content-Type이라는 응답 헤더 정보를 읽고 해당 페이지의 문자 셋 정보를 추출해 줌

```
url = 'http://www.python.org/'
f = urllib.request.urlopen(url)
encoding = f.info().get_content_charset()
```

urllib.parse 모듈

웹 서버에 페이지 또는 정보를 요청할 때 함께 전달하는 데이터

- GET 방식 요청 : Query 문자열
- POST 방식 요청 : 요청 파라미터

name=value&name=value&name=value&.....

- 영문과 숫자는 그대로 전달되지만 한글은 %기호와 함께 16진수 코드 값으로 전달되어야 함
- 웹 크롤링을 할 때 요구되는 Query 문자열을 함께 전달해야 하는 경우, 직접 Query 문자열을 구성해서 전달해야 함

urllib.parse 모듈 사용

- urllib.parse.urlparse()
- urllib.parse.urlencode()

urllib.parse.urlparse("URL문자열")

- 아규먼트에 지정된 URL 문자열의 정보를 다음과 같이 구성하여 저장하는 urllib.parse.ParseResult 객체를 리턴함
- 각 속성들을 이용하여 필요한 정보만 추출할 수 있음

```
url1 = urlparse('https://movie.daum.net/moviedb/main?movieId=93252')
```

```
ParseResult(scheme='https', netloc='movie.daum.net',
            path='/moviedb/main', params='',
            query='movieId=93252', fragment='')
```

```
url1.netloc, url1.path, url1.query , url1.scheme, url1.port,
url1.fragment, url1.geturl()
```

urllib.parse.urlencode()

- 메서드의 아규먼트로 지정된 name과 value로 구성된 딕셔너리 정보를 정해진 규격의 Query 문자열 또는 요청 파라미터 문자열로 리턴 함

```
urlencode({'number': 12524, 'type': 'issue', 'action': 'show'})
```

```
number=12524&type=issue&action=show
```

```
urlencode({'addr': '서울시 강남구 역삼동'})
```

```
addr=%EC%84%9C%EC%9A%B8%EC%8B%9C+%EA%B0%95%EB%82%A8%EA%B5%AC+%EC%97%AD%EC%82%BC%EB%8F%99
```

- Query 문자열을 포함하여 요청하는 것 ➡ GET 방식 요청

urllib.parse.urlencode 함수로 name과 value로 구성되는 Query 문자열을 만듦
URL 문자열의 뒤에 '?' 기호를 추가하여 요청 URL로 사용

```
params = urllib.parse.urlencode({'name': '유니코', 'age': 10})
url = "http://unico2013.dothome.co.kr/crawling/get.php?%s" % params
```

⬇

```
http://unico2013.dothome.co.kr/crawling/get.php?name=%EC%9C
%A0%EB%8B%88%EC%BD%94&age=10
```

```
urllib.request.urlopen(url)
```

- 요청 바디안에 요청 파라미터를 포함하여 요청하는 것 ➡ POST 방식 요청

GET 방식과 같이 name과 value로 구성되는 문자열을 만듦
POST 방식 요청에서는 바이트 형식의 문자열로 전달해야 하므로, encode('ascii') 메서드를 호출하여 바이트 형식의 문자열로 변경
urllib.request.urlopen() 호출 시 바이트 형식의 문자열로 변경된 데이터를 두 번째 아규먼트로 지정

```
data = urllib.parse.urlencode({'name': '유니코', 'age': 10})
data = data.encode('ascii')
```

```
b'name=%EC%9C%A0+%EB%8B%88%EC%BD%94&age=10'
```

```
url = "http://unico2013.dothome.co.kr/crawling/post.php"
urllib.request.urlopen(url, data)
```

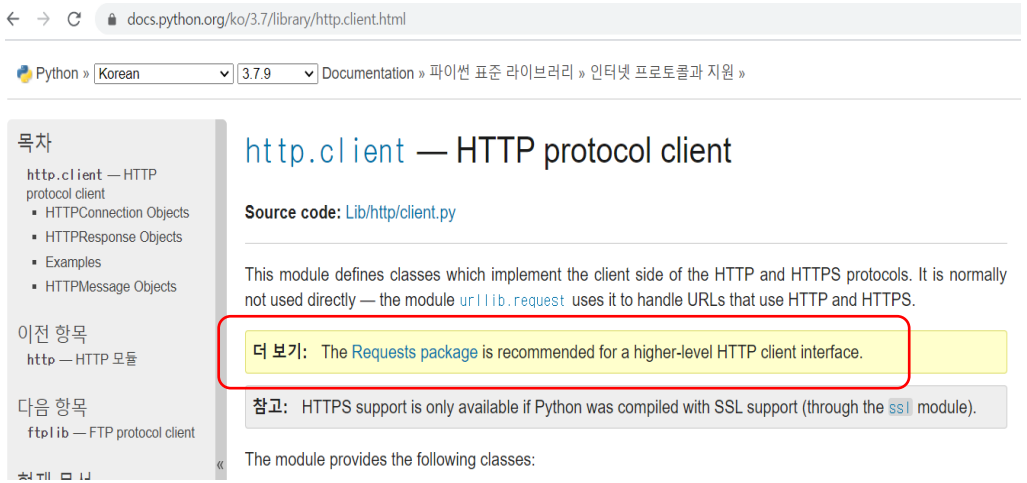
- URL 문자열과 요청 파라미터 문자열을 지정한 urllib.request.Request 객체 생성
- urllib.request.urlopen() 함수 호출 시 URL 문자열 대신 urllib.request.Request 객체 지정

```
data = urllib.parse.urlencode({'name': '유니코', 'age': 10})
postdata = data.encode('ascii')
req = urllib.request.Request(url='http://unico2013.dothome.co.kr/
crawling/post.php',data=postdata)
urllib.request.urlopen(req)
```

requests 패키지를 활용한 웹 페이지 요청

requests 패키지란?

- Kenneth Reitz에 의해 개발된 파이썬 라이브러리
- HTTP 프로토콜과 관련된 기능 지원



requests 패키지의 공식 홈페이지 소개

Requests is an elegant and simple HTTP library for Python, built for human beings. You are currently looking at the documentation of the development release.

〈 https://2.python-requests.org/en/master/ 〉

- 아나콘다에는 requests 패키지가 site-packages로 설치되어 있음
- 만일 설치를 해야 한다면 다음 명령으로 설치

conda install requests 또는 pip install requests

requests.request() 함수

- requests 패키지의 대표 함수
- HTTP 요청을 서버에 보내고 응답을 받아오는 기능 지원

urllib 패키지	requests 패키지
인코딩하여 바이너리 형태로 데이터 전송	딕셔너리 형태로 데이터 전송
요청 방식(GET, POST)에 따라서 구현 방법이 달라짐	request() 함수 호출시 요청 메소드(GET, POST)를 명시하여 요청

requests.request(method, url, **kwargs)

- method : 요청 방식 지정(GET, POST, HEAD, PUT, DELETE, OPTIONS)
- url : 요청할 대상 URL 문자열 지정
- params : [선택적] 요청 시 전달할 Query 문자열 지정
  (딕셔너리, 튜플리스트, 바이트열 가능)
- data : [선택적] 요청 시 바디에 담아서 전달할 요청 파라미터 지정
  (딕셔너리, 튜플리스트, 바이트열 가능)
- json : [선택적] 요청 시 바디에 담아서 전달할 JSON 타입의 객체 지정
- auth : [선택적] 인증처리(로그인)에 사용할 튜플 지정

requests.request() 함수에 요청 방식을 지정하여 호출하는 것과 동일

- requests.get(url, params=None, **kwargs)
- requests.post(url, data=None, json=None, **kwargs)
- requests.head(url, **kwargs)
- requests.put(url, data=None, **kwargs)
- requests.patch(url, data=None, **kwargs)
- requests.delete(url, **kwargs)

- GET 방식 요청은 다음 두 가지 함수 중 하나를 호출하여 처리 가능

requests.request('GET', url, **kwargs)
[kwargs]
params - (선택적) 요청 시 전달할 Query 문자열을 지정합니다.
requests.get(url, params=None, **kwargs)

- Query 문자열을 포함하여 요청 : params 매개변수에 딕셔너리, 튜플리스트, 바이트열(bytes) 형식으로 전달
- Query 문자열을 포함하지 않는 요청: params 매개변수의 설정 생략

- POST 방식 요청은 다음 두 가지 함수 중 하나를 호출하여 처리 가능

requests.request('POST', url, **kwargs)
[kwargs]
data - (선택적) 요청 시 바디에 담아서 전달할 요청 파라미터를 지정합니다.
json - (선택적) 요청 시 바디에 담아서 전달할 JSON 타입의 객체를 지정합니다.
requests.post(url, params=None, **kwargs)

data 매개변수나 json 매개변수로 요청 파라미터를 지정하여 요청하는 것이 일반적

data 매개변수 : 딕셔너리, 튜플리스트 형식, 바이트열(bytes) 형식으로 지정

json 매개변수 : JSON 객체 형식 지정

requests.request(), requests.get(), requests.head(), requests.post() 함수
모두 리턴 값은 requests.models.Response 객체임

Text
- 문자열 형식으로 응답 콘텐츠 추출
- 추출 시 사용되는 문자 셋은 'ISO-8859-1'이므로
  'utf-8' 이나 'euc-kr' 문자 셋으로 작성된 콘텐츠 추출 시 한글이 깨지는 현상 발생
- 추출 전 응답되는 콘텐츠의 문자 셋 정보를 읽고 r.encoding = 'utf-8'와 같이 설정한 후 추출

Content
- 바이트열 형식으로 응답 콘텐츠 추출
- 응답 콘텐츠가 이미지와 같은 바이너리 형식인 경우 사용
- 한글이 들어간 문자열 형식인 경우 r.content.decode('utf-8')를 사용해서 디코드 해야 함

BeautifulSoup

- HTML 및 XML 파일에서 데이터를 추출하기 위한 파이썬 라이브러리
- 파이썬에서 기본적으로 제공하는 라이브러리가 아니므로 별도의 설치가 필요하지만 Anaconda에는 BeautifulSoup 패키지가 Site-packages로 설치되어 있음
- HTML 및 XML 파일의 내용을 읽을 때 다음 파서(Parser) 이용

html.parser	lxml
lxml-xml	html5lib

- 파이썬이 내장하고 있는 파서를 사용해도 되고, 좀 더 성능이 좋은 파서를 추가로 설치해서 사용해도 됨

HTML 파싱

1. BeautifulSoup의 메인 API인 bs4 모듈에서 BeautifulSoup() 함수임포트

2. 파싱할 HTML 문서와 파싱에 사용할 파서(구문 분석기)를 지정하여 호출하면, bs4.BeautifulSoup 객체 리턴

3. HTML 문서에 대한 파싱이 끝나면 트리 구조 형식으로 DOM 객체들이 생성되며, bs4.BeautifulSoup 객체를 통해 접근 가능

```
from bs4 import BeautifulSoup
bs = BeautifulSoup(html_doc, 'html.parser')
bs = BeautifulSoup(html_doc, 'lxml')
bs = BeautifulSoup(html_doc, 'lxml-xml')
bs = BeautifulSoup(html_doc, 'html5lib')
```

```
conda install lxml
conda install html5lib
```

bs4.BeautifulSoup 객체의 태그 접근 방법

HTML 문서를 파싱하고 bs4.BeautifulSoup 객체 생성

```
bs = BeautifulSoup(html_doc, 'html.parser')
```

<html>, <head> 태그와 <body> 태그는 제외하고 접근하려는 태그에 계층 구조를 적용하여 태그명을 . 연산자와 함께 사용

```
bs.태그명
bs.태그명.태그명
bs.태그명.태그명.태그명
bs.태그명.태그명.태그명.태그명
```

bs4.BeautifulSoup 객체의 태그 접근 방법

HTML 문서의 내용을 파싱하여 BeautifulSoup 객체 생성

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Test BS</title>
  </head>
  <body>
    <h1>테스트</h1>
  </body>
</html>
```

```
bs = BeautifulSoup(HTML문서, 'html.parser')
```

계층 구조를 적용한 DOM 객체 생성

```
html
  head
    meta
    title
      Test BS
  body
    h1
      테스트
```

<h1> 태그 접근 방법

```
bs.html.body.h1
bs.html.h1
bs.h1
```

태그명 추출

```
bs.태그명.name
```

속성 추출

```
bs.태그명['속성명']
bs.태그명.attrs
```

콘텐츠 추출

bs.태그명.string
bs.태그명.text
bs.태그명.contents
bs.태그명.strings
bs.태그명.stripped_strings

bs.태그명.string.strip()
bs.태그명.text.strip()
bs.태그명.get_text()
bs.태그명.get_text(strip=True)

부모 태그 추출

bs.태그명.parent

자식 태그들 추출

bs.태그명.children

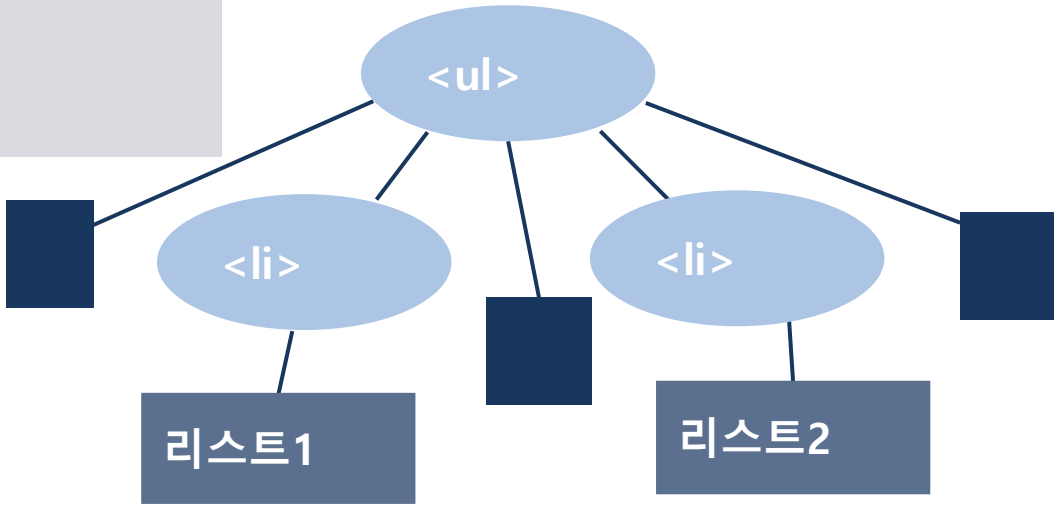
형제 태그 추출

bs.태그명.next_sibling
bs.태그명. previous_sibling
bs.태그명.next_siblings
bs.태그명. previous_siblings

자손 태그들 추출

bs.태그명.descendants

```
<ul>
  <li>리스트1</li>
  <li>리스트2</li>
</ul>
```



- HTML 문서에 대한 파싱이 끝나고 생성된 트리 구조 형식의 DOM 객체들은 bs4.BeautifulSoup 객체의 속성으로도 접근 가능하지만 다음 메서드로도 가능

태그 찾기 기능의 주요 메서드

- find_all()
- find()
- select()

태그 찾기 기능의 기타 메서드

- find_parents()와 find_parent()
- find_next_siblings()과 find_next_sibling()
- find_previous_siblings()와 find_previous_sibling
- find_all_next()와 find_next()
- first_all_previous()와 first_previous()

bs.find_all()

주어진 기준에 맞는 모든 태그들을 찾아오며 결과는 bs4.element.ResultSet 객체로 리턴

```
find_all(name=None, attrs={}, recursive=True, text=None, limit=None, **kwargs)
```

태그명,
정규표현식을
적용한 태그명,
태그명 리스트,
속성 정보, 함수,
논리값

```
find_all('div')
find_all(['p','img'])
find_all(True)
find_all(re.compile('^b'))
find_all(id='link2')
find_all(id=re.compile("para$")
find_all(id=True)
find_all('a', class_="sister")
find_all(src=re.compile("png$"), id='link1')
```

```
find_all(attrs={'src':re.compile('png$'), 'id':'link1'})
find_all(text='example')
find_all(text=re.compile('example'))
find_all(text=re.compile('^test'))
find_all(text=['example', 'test'])
find_all('a', text='python')
find_all('a', limit=2)
find_all('p', recursive=False)
```

bs.find()

주어진 기준에 맞는 태그 한 개를 찾아오며 결과는 bs4.element.Tag 객체로 리턴하며 결과값이 없으면 None을 리턴

find()는 find_all()에 limit=1로 설정한 것과 동일하게 수행,
find_all()에서 사용하는 아규먼트 값을 find()에서도 동일하게 사용 가능

```
find(name=None, attrs={}, recursive=True, text=None, **kwargs)
```

태그명,
정규표현식을
적용한 태그명,
태그명 리스트,
속성정보, 함수,
논리값

```
find('div') == find_all('div', limit=1)
find(re.compile('^b')) == find_all(re.compile('^b'), limit=1)
```

bs.select()

주어진 CSS 선택자에 맞는 모든 태그들을 찾아오며 결과는 list 객체로 리턴

```
select(selector, namespaces=None, limit=None, **kwargs)
```

CSS 선택자를 적용한 호출

```
select('태그명')
select('.클래스명')
select('#아이디명')
select('태그명.클래스명))
```

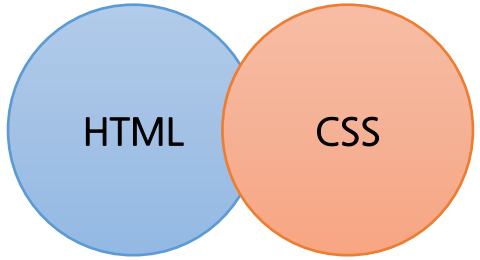
자식 선택자 및 자손 선택자를 사용하면 HTML문서의
트리 구조를 적용하여 태그를 찾을 수 있음

```
select('상위태그명 > 자식태그명 > 손자태그명')
select('상위태그명.클래스명 > 자식태그명.클래스명')
select('상위태그명.클래스명   자손태그명')
select('상위태그명 > 자식태그명  자손태그명')
select('#아이디명 > 태그명.클래스명)
select('태그명[속성]')
select('태그명[속성=값]')
select('태그명[속성$=값]')
select('태그명[속성^=값]')
select('태그명:nth-of-type(3)')
```

정적 웹 페이지와 동적 웹 페이지의 구분

정적 웹 페이지

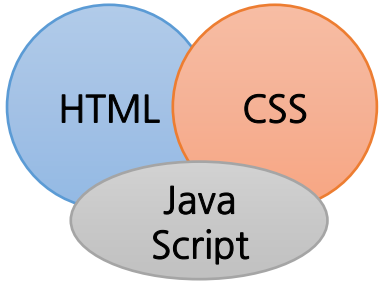
- 웹 서버에서 전송된 웹 페이지의 소스에서 화면에 렌더링된 내용을 모두 찾을 수 있는 경우
- HTML만으로 작성되거나 HTML과 CSS 기술 등으로 구현된 경우



정적 웹 페이지와 동적 웹 페이지의 구분

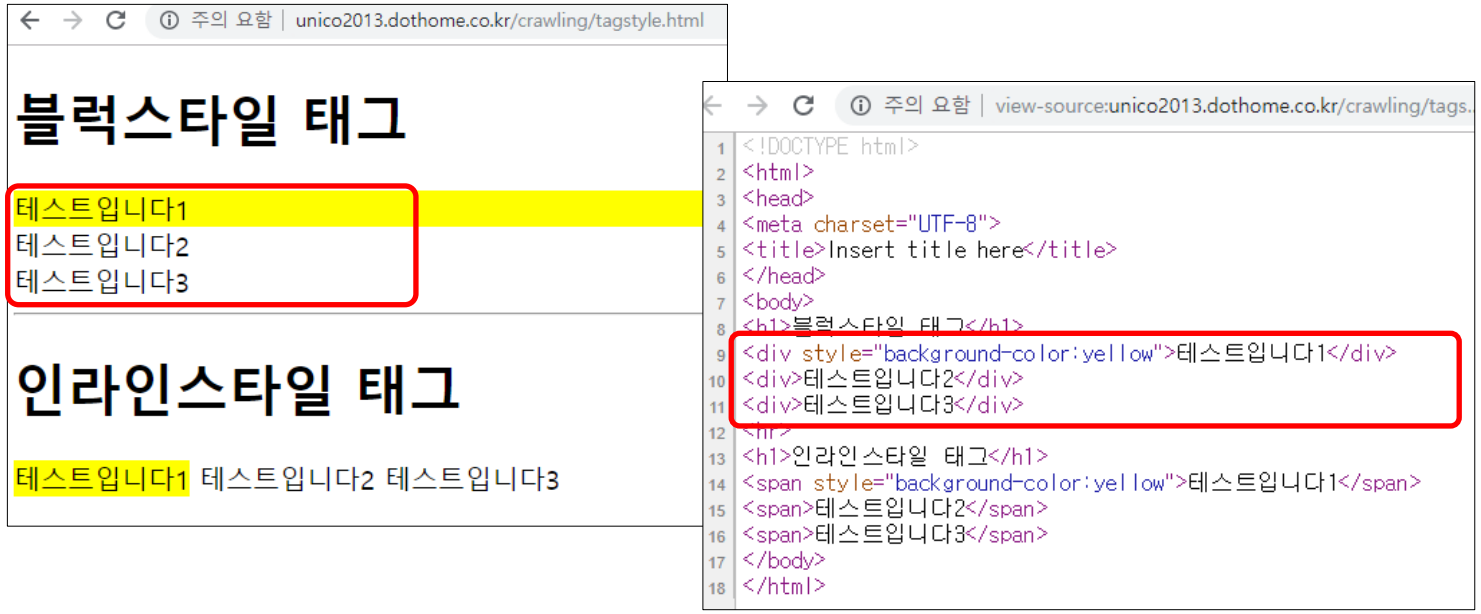
동적 웹 페이지

- 웹 서버에서 전송된 웹 페이지의 소스에서 화면에 렌더링된 내용을 일부 찾을 수 없는 경우
- HTML과 CSS 기술 외에 JavaScript 프로그래밍 언어로 브라우저에서 실행시킨 코드에 의해 웹 페이지의 내용이 렌더링 시 자동으로 생성되는 페이지



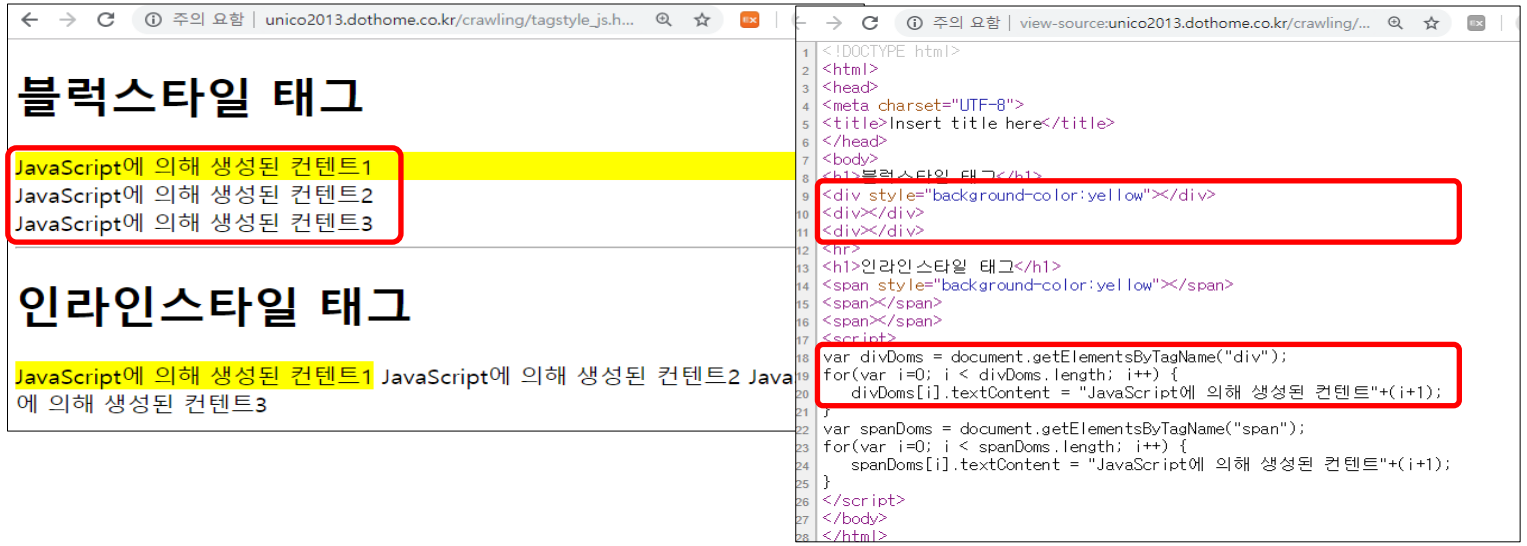
정적 웹 페이지 화면

- 화면에 렌더링된 각 태그들의 콘텐츠가 페이지의 소스에서도 모두 보여짐



동적 웹 페이지 화면

- 화면에 렌더링된 일부 태그들의 콘텐츠를 페이지의 소스에서 찾아볼 수 없음
- 태그나 태그처럼 소스코드에서 그 내용을 찾아볼 수 없음



동적 웹 페이지에 의해 렌더링된 동적 콘텐츠의 스크래핑

Selenium이라는 웹 브라우저를 자동화 하는 도구 모음을 사용

Selenium	▪ 다양한 플랫폼과 언어를 지원하는 이용하는 브라우저 자동화 도구 모음

- WebDriver라는 API를 통해 운영체제에 설치된 크롬이나 파이어폭스 등의 브라우저를 기동시키고 웹 페이지를 로드하고 제어
- 브라우저를 직접 동작시킨다는 것은 JavaScript에 의해 생성되는 콘텐츠와 Ajax 통신 등을 통해 뒤늦게 불려오는 콘텐츠를 처리할 수 있다는 것을 의미함

WebDriver API

- 간결한 프로그래밍 인터페이스를 제공하도록 설계
- 동적 웹 페이지를 보다 잘 지원할 수 있도록 개발

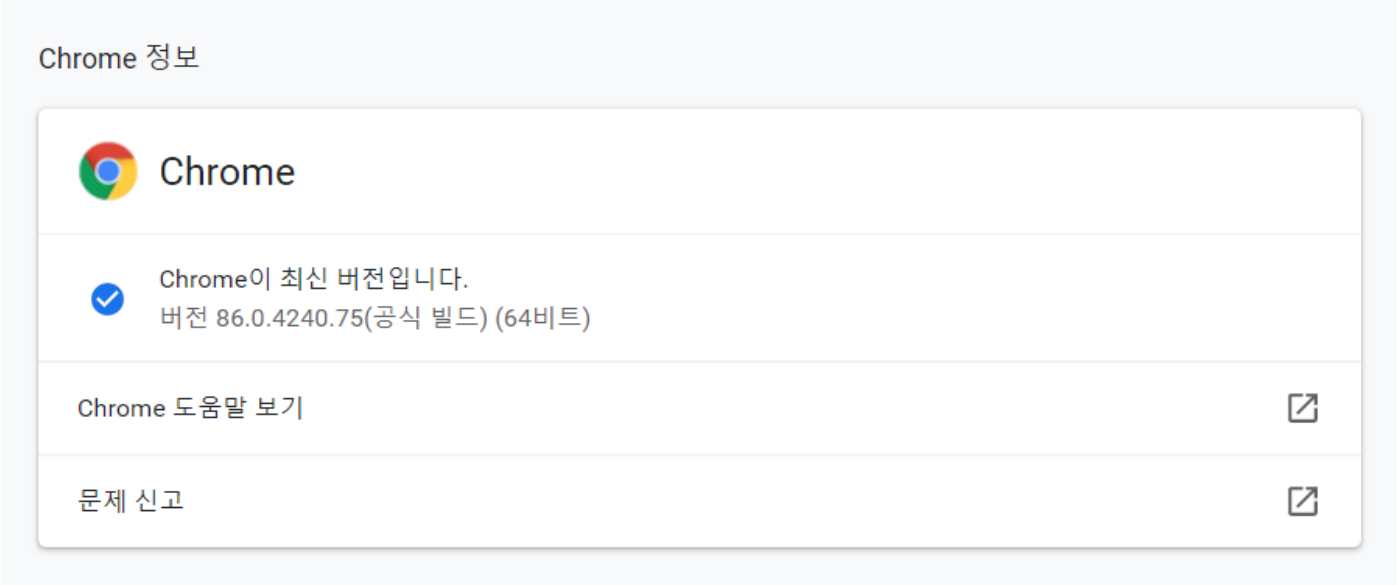
- WebDriver의 목표 : 최신 고급 웹 응용 프로그램 테스트 문제에 대한 향상된 지원을 제공하는 잘 디자인된 객체 지향 API를 제공하는 것
- Selenium-WebDriver는 자동화를 위한 각 브라우저의 기본 지원을 사용하여 브라우저를 직접 호출

Selenium은 cmd 창에서 pip 명령 또는 conda 명령을 통해서 설치 가능

```
conda install selenium
pip install selenium
```

Chrome Driver 설치

Selenium의 Web Driver에 의해 제어되는 Chrome Driver를 설치하기 위해
먼저 시스템에 설치된 크롬 브라우저의 버전 체크



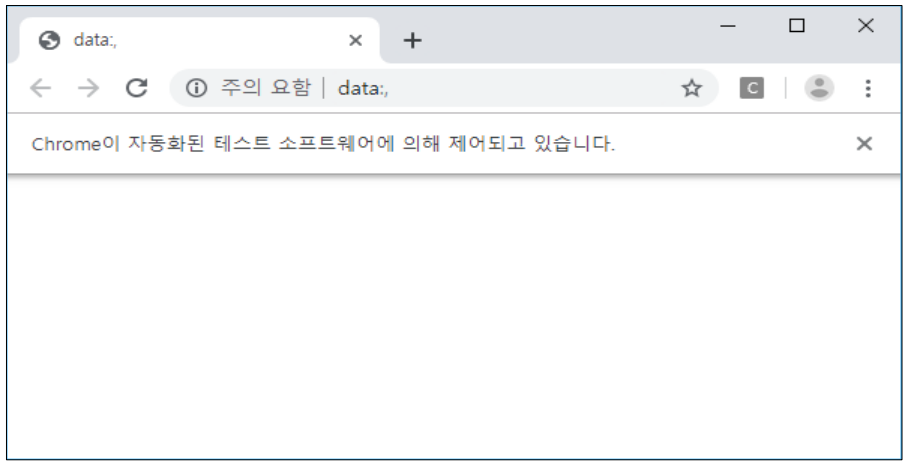
WebDriver 객체 생성

다음 코드를 수행시켜서 크롬 드라이버를 기반으로
selenium.webdriver.chrome.webdriver.WebDriver 객체 생성

```
driver = webdriver.Chrome('C:/Temp/chromedriver')
```

WebDriver 객체 생성

아규먼트로 chromedriver.exe 파일이 존재하는 디렉토리와 파일명에 대한 패스 정보를 지정하면 Selenium에 의해 관리되는 크롬 브라우저가 기동 됨



페이지 가져오기

selenium.webdriver.chrome.webdriver.WebDriver 객체의 get() 메서드를 사용하여
크롤링하려는 웹 페이지를 제어하는 크롬 브라우저에 로드하고 렌더링

```
driver.get('http://www.google.com/ncr')
```

- 경우에 따라 페이지 로드가 완료되거나 시작되기 전에 WebDriver가 제어권을 반환할 수 있음
- 견고성을 확보하려면 Explicit and Implicit Waits를 사용하여 요소가 페이지에 존재할 때까지 기다려야 함

```
driver.implicitly_wait(3)
driver.get('http://www.google.com/ncr')
```

```
driver.get('http://www.google.com/ncr', 5)
```

WebDriver의 요소 찾기는 WebDriver 객체 및 WebElement 객체에서 제공되는 다음 메서드들을 사용

태그의 id 속성 값으로 찾기

```
byId = driver.find_element_by_id('btype')
또는
from selenium.webdriver.common.by import By
byId = driver.find_element(by=By.ID, value='btype')
```

태그의 class 속성 값으로 찾기

```
target = driver.find_element_by_class_name('quickResultLstCon')
또는
target = driver.find_element(By.CLASS_NAME, "quickResultLstCon")
```

태그명으로 찾기

```
byTagName = driver.find_element_by_tag_name('h1')
또는
byTagName = driver.find_element(By.TAG_NAME, 'h1')
```

링크 텍스트로 태그 찾기

```
byLinkText = driver.find_element_by_link_text('파이썬 학습 사이트')
또는
byLinkText = driver.find_element(By.LINK_TEXT, '파이썬 학습 사이트')
```

```
<a href="https://www.python.org/">파이썬 학습 사이트</a>
```

부분 링크 텍스트로 태그 찾기

```
byLinkText = driver.find_elements_by_partial_link_text('사이트')
또는
byLinkText = driver.find_element(By.PARTIAL_LINK_TEXT, '사이트')
```

CSS 선택자로 태그 찾기

```
byCss1 = driver.find_element_by_css_selector('section>h2')
또는
byCss1 = driver.find_element(By.CSS_SELECTOR, 'section>h2')
```

Xpath로 태그 찾기

```
byXpath1 = driver.find_element_by_xpath('//*[@id="f_subtitle"]')
또는
byXpath1 = driver.find_element(By.XPATH, '//*[@id="f_subtitle"]')
```

조건에 맞는 요소 한 개 찾기 : WebElement 객체 리턴

```
driver.find_element_by_xxx("xxx에 알맞는 식")
```

조건에 맞는 모든 요소 찾기 : list 객체 리턴

```
driver.find_elements_by_xxx("xxx에 알맞는 식")
```

요소의 정보 추출

```
element = driver.find_element_by_id("element_id")
# 태그명
element.tag_name
# 텍스트 형식의 콘텐츠
element.text
# 속성값
element.get_attribute('속성명')
```