

# Web Scrapping & Web Crawling

**Bok, Jong Soon**  
**javaexpert@nate.com**  
**<https://github.com/swacademy>**

# 웹 데이터 수집 종류

## ■ 뉴스 기사 크롤링

- RSS 서비스(예: <http://news.jtbc.joins.com/Etc/RssService.aspx>)
- XML 형식

## ■ SNS 데이터 크롤링

- 트위터, 댓글 등
- Text 형식 또는 JSON 형식

## ■ 웹 데이터 수집

- 웹 페이지 URL에서 원하는 정보를 수집
- HTML 형식

## ■ 공공데이터 수집

- Open API를 이용
- URL이 존재
- CSV 형식, 정형화된 데이터
  - Pandas 패키지의 read\_csv("url")을 이용
- JSON 형식
  - Pandas 패키지의 read\_json("url")을 이용

데이터를 선택  
적으로 가져옴

전체 데이터  
를 가져옴

# Using urllib.request module

# Crawling & Scraping

## ■ Crawling

- Web site의 data를 그대로 취득하는 것

## ■ Scraping

- Crawling하여 모든 취득한 data에서 필요한 것만 추출하거나 변환하는 처리 포함.

# pandas.read\_html()

- Read HTML tables into a list of DataFrame objects.
- Syntax
  - `read_html(url)`

```
url = 'http://www.fdic.gov/bank/individual/failed/banklist.html'  
df = pd.read_html(url)  
  
print(df[0].info())
```

# pandas.read\_csv()

- Read a comma-separated values (csv) file into DataFrame.
- Syntax
  - `read_csv(url)`

```
url =  
'https://github.com/vincentarelbundock/Rdatasets/blob/master/csv/datasets/  
Titanic.csv'  
df = pd.read_csv(url)  
  
print(df.info())
```

# urllib.request Module & HTTPResponse

- Web page crawling
- 표준 library
- url을 열고 파일을 읽기 위한 module
- **urlopen(url)**
  - Open the url
  - Return **http.client.HTTPResponse**
- **HTTPResponse.read()**
  - HTML을 binary 형태로 가져온다.
  - decoding 해야 함.

# 정규표현식으로 Scraping

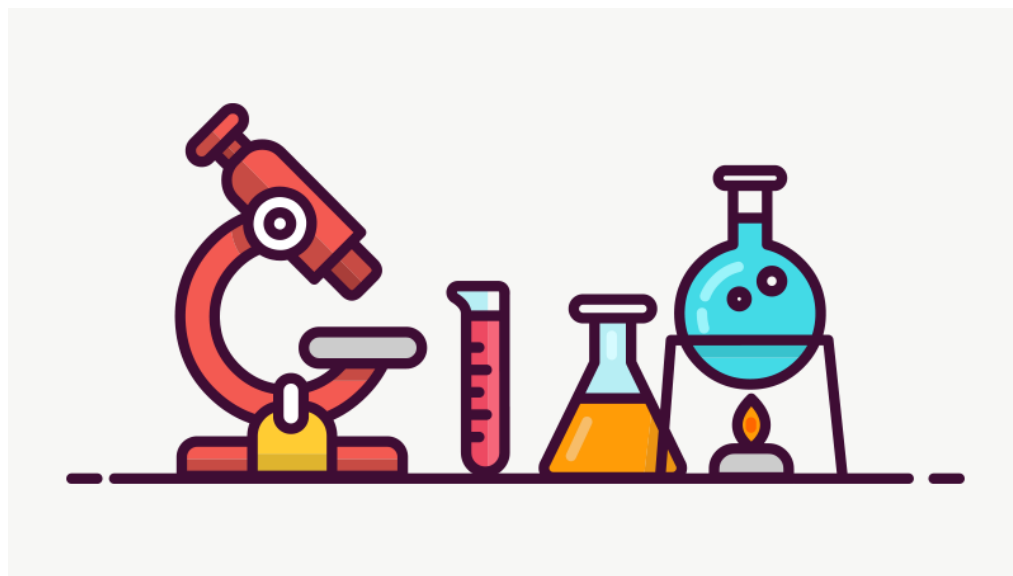
```
...
import re
from html import unescape

...
hanbit = urlopen('http://www.hanbit.co.kr/store/books/full_book_list.html')
html = hanbit.read().decode()

# re.findall()을 사용해 도서 하나에 해당하는 HTML을 추출한다.
for partial_html in re.findall(r'<td class="left"><a.*?</td>', html, re.DOTALL):
    # 도서의 URL을 추출합니다.
    url = re.search(r'<a href="(.*?)">', partial_html).group(1)
    url = 'http://www.hanbit.co.kr' + url

# Tag를 제거해서 도서의 제목을 추출한다.
title = re.sub(r'<.*?>', '', partial_html)
title = unescape(title)
```





**Lab. urllib 사용하기**

# Using requests module

# requests

- Python에서 HTTP 요청을 만들기 위한 사실상의 표준
- 학습할 내용
  - 가장 일반적인 HTTP 메소드를 사용하여 요청하기
  - 쿼리 문자열 및 메시지 본문을 사용하여 사용자의 요청과 데이터를 변경하기
  - 요청 및 응답에서 데이터 검사하기
  - 인증 된 요청 만들기
  - 응용 프로그램이 백업 또는 속도 저하를 막을 수 있도록 요청을 구성하기

# requests module

- pip 명령으로 쉽게 설치
  - `pip install requests`
- requests가 설치되면 응용 프로그램에서 사용
  - `import requests`

# GET 요청

- GET은 HTTP 요청(Request) 방식 중 하나
- `requests.get()` : 지정된 resource에서 데이터를 가져옴

```
1 import requests
```

```
1 requests.get('https://api.github.com')
```

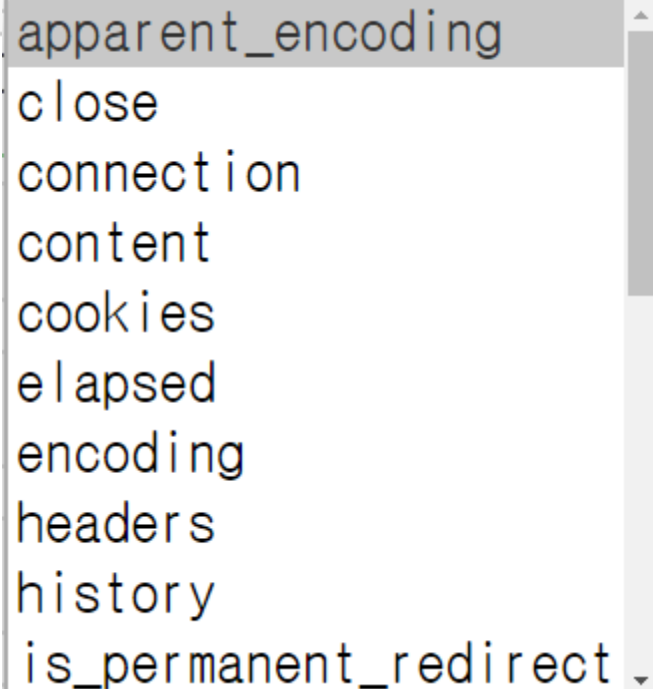
<Response [200]>

# Response 객체

- 응답(Response)은 요청의 결과를 저장한 객체

```
1 response = requests.get('https://api.github.com')
```

```
1 response.
```



- apparent\_encoding
- close
- connection
- content
- cookies
- elapsed
- encoding
- headers
- history
- is\_permanent\_redirect

# Status Code

- HTTP 응답 메시지의 상태 라인에는 상태 코드와 상태 메시지를 포함하고 있음

상태 코드	의미	내 용
100번 영역	정보전송	임시적인 응답을 나타내는 것은 Status-Line과 선택적인 헤더들로 구성되어 있고, 빈 줄로 끝을 맺는다.
200번 영역	성공	클라이언트의 요구가 성공적으로 수신되어 처리되었음을 의미 200(성공), 201(POST요청 처리), 204(전송할 데이터 없음)
300번 영역	리다이렉션	해당 요구사항을 처리하기 위해서는 사용자 에이전트에 의해 수행되어야 할 추가적인 동작이 있음을 의미
400번 영역	클라이언트 측 오류	클라이언트가 서버에게 보내는 요구 메시지를 완전히 처리하지 못한 경우와 같이 클라이언트에서 오류가 발생한 경우에 사용 401(사용자인증), 403(접근권한 없음), 404(요청한 URL 없음)
500번 영역	서버측 오류	서버 자체에서 발생한 오류상황이나 요구사항을 제대로 처리할 수 없을 때 사용.

# Status Code (Cont.)

```
1 response.status_code
```

200

```
1 if response.status_code == 200:
2     print('Success!')
3 elif response.status_code == 404:
4     print('Not Found.')
```

상태코드 정보를 사용하여 코드에서 의사 결정을 내릴 수 있음

Success!

Response 인스턴스를 사용하면 상태 코드가 200에서 400 사이 이면 **True**로 평가되고 그렇지 않으면 **False**로 평가

```
1 if response:
2     print('Success!')
3 else:
4     print('An error has occurred.')
```

Success!



# Content

- GET 요청의 응답의 메시지 본문에는 중요한 정보가 포함되어있는 경우가 많음
- 응답 내용을 바이트 단위로 보려면 **.content**를 사용

```
1 response = requests.get('https://api.github.com')  
2 response.content
```

```
b'{"current_user_url": "https://api.github.com/user", "current_user  
_authorizations_html_url": "https://github.com/settings/connection  
s/applications{/client_id}", "authorizations_url": "https://api.git  
hub.com/authorizations", "code_search_url": "https://api.github.co  
m/search/code?q={query}{&page,per_page,sort,order}", "commit_searc  
h_url": "https://api.github.com/search/commits?q={query}{&page,per
```

# Content (Cont.)

- **.text** 속성에 액세스 할 때 문자열 인코딩을 지정할 수 있음
- 내부적으로 추론해서 자동 지정됨

```
1 response = requests.get('https://api.github.com')
2 response.encoding = 'utf-8'
3 response.text
```

```
'{"current_user_url": "https://api.github.com/user", "current_user_
authorizations_html_url": "https://github.com/settings/connections
c/applications/clients/{client_id}/authorizations_url": "https://api.git
```

```
1 response = requests.get('http://jvaspecialist.co.kr')
2 response.content
```

```
b'<!DOCTYPE html>
<html manifest="index.manifest">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="robots" content="index, follow">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="google-site-verification" content="RPK2fDgygxeiosW4a4sE8zcla1193rlzAf90zQjTtvc" />
  <title>자바전문가</title>
  <!-- Favicon -->
  <link href="/favicon.png" rel="icon">
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-md-12">
        <h1>자바전문가</h1>
      </div>
    </div>
  </div>
</body>
</html>
```

```
1 response = requests.get('http://jvaspecialist.co.kr/emp')
2 response.text
```

```
'<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="robots" content="index, follow">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="google-site-verification" content="RPK2fDgygxeiosW4a4sE8zcla1193rlzAf90zQjTtvc" />
  <title>자바전문가</title>
  <!-- Favicon -->
  <link href="/favicon.png" rel="icon">
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-md-12">
        <h1>자바전문가</h1>
      </div>
    </div>
  </div>
</body>
</html>
```

## Content (Cont.)

- **.text** 속성에 액세스 할 때 문자열 인코딩을 지정할 수 있음

```
1 response = requests.get('https://api.github.com')
2 response.json()
```

```
{'current_user_url': 'https://api.github.com/user',
 'current_user_authorizations_html_url': 'https://github.com/settings/connections/applications{/client_id}',
 'authorizations_url': 'https://api.github.com/authorizations',
 'code_search_url': 'https://api.github.com/search/code?q={query}{&page,per_page,sort,order}',
 'commit_search_url': 'https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}',
```

# Response Headers

```
response.headers
```

```
{'Date': 'Sat, 08 Jun 2019 11:52:45 GMT', 'Content-Type':  
'application/json; charset=utf-8', 'Transfer-Encoding': 'chunked',  
'Server': 'GitHub.com', 'Status': '200 OK', 'X-RateLimit-Limit': '60',  
'X-RateLimit-Remaining': '58', 'X-RateLimit-Reset': '1559998358',  
'Cache-Control': 'public, max-age=60, s-maxage=60', 'Vary': 'Accept,  
Accept-Encoding', 'ETag': 'W/"7dc470913f1fe9bb6c7355b50a0737bc"',  
'X-GitHub-Media-Type': 'github.v3; format=json',  
'Access-Control-Expose-Headers': 'ETag, Link, Location, Retry-After,  
X-GitHub-OTP, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset,  
X-OAuth-Scopes, X-Accepted-OAuth-Scopes, X-Poll-Interval,  
X-GitHub-Media-Type', 'Access-Control-Allow-Origin': '*',  
'Strict-Transport-Security': 'max-age=31536000; includeSubdomains;  
preload', 'X-Frame-Options': 'deny', 'X-Content-Type-Options': 'nosniff',  
'X-XSS-Protection': '1; mode=block', 'Referrer-Policy':  
'origin-when-cross-origin, strict-origin-when-cross-origin',  
'Content-Security-Policy': "default-src 'none'", 'Content-Encoding':  
'gzip', 'X-GitHub-Request-Id': 'DC9A:300B:149E1B4:18FE584:5CFBA18D'}
```

```
1 response.headers['Content-Type']
```

```
'application/json; charset=utf-8'
```

# Request Parameter 사용하기

```
1 import requests
2
3 # 깃허브 저장소에서 검색 파라미터 사용하기
4 response = requests.get(
5     'https://api.github.com/search/repositories',
6     params={'q': 'requests+language:python'},
7 )
8
9 # 'requests' 저장소의 속성 검사
10 json_response = response.json()
11 repository = json_response['items'][0]
12 print(f'Repository name: {repository["name"]}')
13 print(f'Repository description: {repository["description"]}')
```

params 매개변수를 이용  
해서 tuple 형식으로 지정  
가능

Repository name: requests

Repository description: Python HTTP Requests for Humans™ 🌟🍰🌟

# Request Headers

```
1 import requests
2
3 response = requests.get(
4     'https://api.github.com/search/repositories',
5     params={'q': 'requests+language:python'},
6     headers={'Accept': 'application/vnd.github.v3.text-match+json'},
7 )
8
9 json_response = response.json()
10 repository = json_response['items'][0]
11 print(f'Text matches: {repository["text_matches"]}')

```

get() 메소드에 headers 매개 변수를 사용하여 HTTP 헤더를 딕셔너리 형식으로 전달

Text matches: [{ 'object\_url': 'https://api.github.com/repositories/1362490', 'object\_type': 'Repository', 'property': 'description', 'fragment': 'Python HTTP Requests for Humans™ 🌟🍰🌟', 'matches': [{ 'text': 'Requests', 'indices': [12, 20]}] }]



# 다른 HTTP 요청 방법

```
1 requests.post('https://httpbin.org/post', data={'key': 'value'})
```

<Response [200]>

```
1 requests.put('https://httpbin.org/put', data={'key': 'value'})
```

<Response [200]>

```
1 requests.delete('https://httpbin.org/delete')
```

<Response [200]>

```
1 requests.head('https://httpbin.org/get')
```

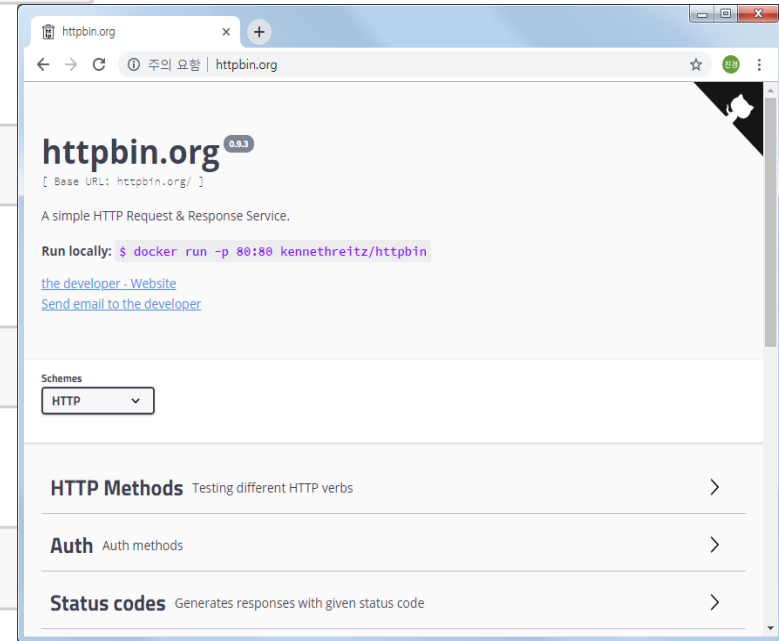
<Response [200]>

```
1 requests.patch('https://httpbin.org/patch', data={'key': 'value'})
```

<Response [200]>

```
1 requests.options('https://httpbin.org/get')
```

<Response [200]>



httpbin은 테스트 요청을 받아  
들이고 요청에 대한 데이터로  
응답하는 서비스

# 메시지 본문

- POST, PUT 요청은 메시지 본문을 통해 데이터를 전달
- 요청을 사용하여 결과 데이터를 해당 함수의 data 매개변수로 전달

```
1 requests.post('https://httpbin.org/post', data={'key': 'value'})
```

<Response [200]>

데이터를 dict로 보낼 수 있음

```
1 requests.post('https://httpbin.org/post', data=[('key', 'value')])
```

<Response [200]>

데이터를 tuple 목록으로 보낼 수도 있음



## 메시지 본문 (Cont.)

- JSON 데이터의 경우 json 매개변수 이용
- Content-Type 헤더에 '**application/json**' 이 추가됨

```
1 response = requests.post('https://httpbin.org/post',  
2                             json={'key': 'value'})  
3 json_response = response.json()  
4 json_response['data']
```

```
'{"key": "value"}'
```

```
1 json_response['headers']['Content-Type']
```

```
'application/json'
```

# Request 검사하기

- **.request** 속성에 액세스하여 **PreparedRequest**를 볼 수 있음

```
1 response = requests.post('https://httpbin.org/post',  
2                             json={'key': 'value'})
```

```
1 response.request.headers['Content-Type']
```

'application/json'

```
1 response.request.url
```

'https://httpbin.org/post'

```
1 response.request.body
```

b'{"key": "value"}'

# Authentication

- 인증 정보를 전달할 수 있는 **auth** 매개변수 제공

```
1 from getpass import getpass
2 requests.get('https://api.github.com/user',
3              auth=(' ', getpass()))
```

. . . . .

↑ 아이디는 본인의 것으로...

<Response [200]>

```
1 requests.get('https://api.github.com/user')
```

<Response [401]>

자격증명 없이 요청할 경우

# SSL 인증서

- SSL 인증서 확인을 사용하지 않으려면 요청 함수의 **verify** 매개변수로 **False**를 전달

```
1 requests.get('https://api.github.com', verify=False)
```

```
C:\Users\User\Anaconda3\lib\site-packages\urllib3\connectionpool.py:847:  
InsecureRequestWarning: Unverified HTTPS request is being made. Adding ce  
rtificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings  
InsecureRequestWarning)
```

```
<Response [200]>
```

# Timeout

- 요청의 시간 초과를 설정하려면 **timeout** 매개변수 사용

```
1 requests.get('https://api.github.com', timeout=1)
```

<Response [200]>

```
1 import requests
2 from requests.exceptions import Timeout
3
4 try:
5     response = requests.get('https://api.github.com', timeout=1)
6 except Timeout:
7     print('요청 시간을 초과했습니다.')
8 else:
9     print('요청이 정상 처리되었습니다.')
```

프로그램에서 시간 초과  
예외를 잡고 이에 따라 응  
답을 처리할 수 있음

요청이 정상 처리되었습니다.

# Session 객체

- 요청 방법에 대한 통제를 세밀하게 조정하거나 요청 성능을 향상시켜야 하는 경우에는 **Session** 객체를 직접 사용

```
1 import requests
2 from getpass import getpass
3
4 with requests.Session() as session:
5     session.auth = ('hjk7902', getpass())
6     response = session.get('https://api.github.com/user')
7
8 print(response.headers)
9 print(response.json())
```

```
. . . . .
{'Date': 'Mon, 22 Jul 2019 02:55:02 GMT', 'Content-Type': 'application/js
on; charset=utf-8', 'Transfer-Encoding': 'chunked', 'Server': 'GitHub.co
m', 'Status': '200 OK', 'X-RateLimit-Limit': '5000', 'X-RateLimit-Remaini
ng': '4999', 'X-RateLimit-Reset': '1563767702', 'Cache-Control': 'privat
e, max-age=60, s-maxage=60', 'Vary': 'Accept, Authorization, Cookie, X-Gi
tHub-OTP, Accept-Encoding', 'ETag': 'W/"f6c828a183c7dc0bb6fd6b55c18503a
```

# 재시도 제한

- 전송 어댑터를 만들고 **max\_retries** 매개변수를 설정한 다음 기존 세션에 마운트

```
1 import requests
2 from requests.adapters import HTTPAdapter
3 from requests.exceptions import ConnectionError
4
5 github_adapter = HTTPAdapter(max_retries=3)
6
7 session = requests.Session()
8
9 session.mount('https://api.github.com', github_adapter)
10
11 try:
12     session.get('https://api.github.com')
13 except ConnectionError as ce:
14     print(ce)
```

# requests Exercise. 환율 정보 가져오기

```
1 import requests
2 from bs4 import BeautifulSoup
```

```
1 url = 'https://finance.naver.com/marketindex/' # 환율 정보
2 market_index = requests.get(url)
```

```
1 market_index
```

<Response [200]>

```
1 soup = BeautifulSoup(market_index.content, "html.parser")
```

```
1 price = soup.select_one("div.head_info > span.value")
2 print(price)
```

<span class="value">1,189.70</span>

```
1 print("usd/krw=", price.text)
```

usd/krw= 1,189.70



# requests Exercise. 영화 랭킹 출력하기

```
1 import requests
2 from bs4 import BeautifulSoup
```

```
1 url = 'https://movie.naver.com/movie/sdb/rank/rmovie.nhn'
2 movie_ranking = requests.get(url)
3 movie_ranking
```

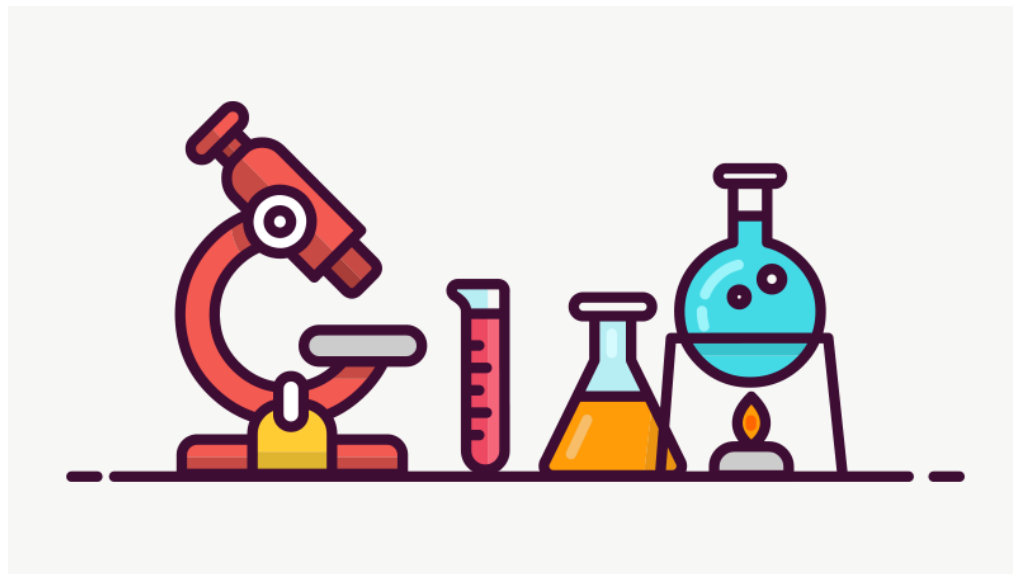
<Response [200]>

```
1 soup = BeautifulSoup(movie_ranking.content, 'html.parser')
```

```
1 title_list = soup.select('div.tit3 > a')
2 title_list[0]
```

<a href="/movie/bi/mi/basic.nhn?code=177967" title="악인전">악인전</a>

```
1 for i, title in enumerate(title_list):
2     print('{}위: {}'.format(i+1, title.text))
```



**Lab. requests 사용하기**

# Beautiful Soup & Parser

# Beautiful Soup

- BeautifulSoup은 Screen-scraping Project를 위해 설계된 Python Library.
- 구문 분석, 트리 탐색, 검색 및 수정을 위한 몇 가지 간단한 방법과 Python 관용구를 제공하며 문서를 분석하고 필요한 것을 추출하는 도구
- 들어오는 문서를 Unicode로 보내고 문서를 UTF-8로 자동 변환
- BeautifulSoup은 lxml 및 html5lib과 같은 Python Parser Library를 사용할 수 있음
  - 이를 이용하면 속도를 개선시키거나 호환성이 높은 응용프로그램을 만들 수 있음
- 공식 사이트
  - <https://www.crummy.com/software/BeautifulSoup/>
- Documentation
  - <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

# Parser Libraries

Parser	사용법	장점	단점
Python의 html.parser	BeautifulSoup(markup, "html.parser")	<ul style="list-style-type: none"><li>보통 속도</li><li>Python 2.7.3, 3.2.2 이상 버전에서 호환됨</li></ul>	<ul style="list-style-type: none"><li>Python 2.7.3 또는 3.2.2 이전버전에서 호환되지 않음</li></ul>
lxml's HTML parser	BeautifulSoup(markup, "lxml")	<ul style="list-style-type: none"><li>매우 빠름</li><li>호환성</li></ul>	<ul style="list-style-type: none"><li>외부 C에 의존함</li></ul>
lxml's XML parser	BeautifulSoup(markup, "lxml-xml") BeautifulSoup(markup, "xml")	<ul style="list-style-type: none"><li>매우 빠름</li></ul>	<ul style="list-style-type: none"><li>XML Parser만 지원</li><li>외부 C에 의존함</li></ul>
html5lib	BeautifulSoup(markup, "html5lib")	<ul style="list-style-type: none"><li>호환성이 높음</li><li>Web Browser와 같은 방식으로 page parsing</li><li>유효한 HTML5를 만들</li></ul>	<ul style="list-style-type: none"><li>매우 느림</li></ul>

# Selector API

- BeautifulSoup은 가장 일반적으로 사용되는 CSS 선택자 지원
- BeautifulSoup의 Selector API는 `select()`와 `select_one()`, `find()` 등
- `select()`와 `select_one()`만 알아도 원하는 요소를 찾기에 충분
- `soup.select("CSS 선택자")`
  - CSS 선택자에 해당하는 모든 요소를 반환
- `soup.select_one("CSS 선택자")`
  - CSS 선택자에 맞는 오직 첫 번째 태그 요소만 반환
- <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

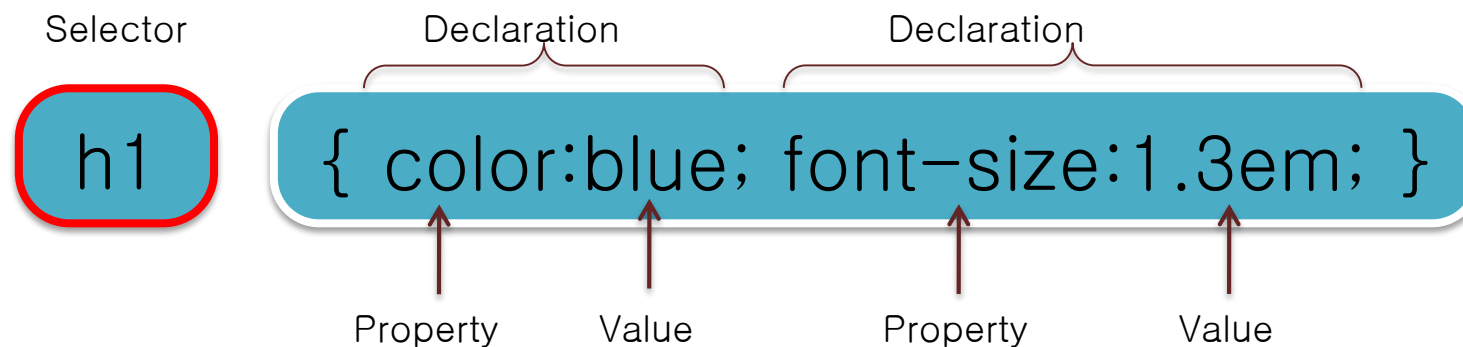
# CSS Selector

## ■ CSS(Cascading Style Sheet)

- CSS는 문서의 contents와 layout, 글꼴 및 시각적 요소들로 표현되는 문서의 외관(design)을 분리하기 위한 목적으로 만들어졌음

## ■ CSS 선택자

- CSS를 이용해서 HTML 문서에 시각적 요소를 부여할 때 문서 내의 어느 요소에 부여할지를 결정하기 위해 사용하는 것
- CSS 선택자(Selector)는 HTML 문서의 태그 이름, class 속성, id 속성 등을 이용해서 작성할 수 있음



# 실습을 위한 준비

```
1 import requests
2 from requests_file import FileAdapter
3
4 s = requests.Session()
5 s.mount('file://', FileAdapter())
```

```
1 res = s.get('file:///sample.html')
```

```
1 res
```

<Response [200]>

```
1 from bs4 import BeautifulSoup
```

```
1 soup = BeautifulSoup(res.content, "html.parser")
```

```
1 el = soup.select_one("h1")
2 print(el)
```

<h1>Hello CSS</h1>

```
1 <html>
2 <head><title>HTML Sample</title>
3 </head>
4 <body>
5     <h1>Hello CSS</h1>
6     <div id="subject">선택자</div>
7     <div class="contents">선택자를 어떻게 작성하느냐에 따라
8 <span>다른 <b>요소가 반환</b></span> 됩니다.</div>
9     <div>CSS 선택자는 다양한 곳에서 <b>활용</b>됩니다.</div>
9 </body>
10 </html>
```



# 선택자 종류

## ■ Tag 선택자 ("element")

- Tag 선택자는 일반적으로 스타일 정의하고 싶은 html 태그 이름을 사용
- 요소 안의 텍스트는 **text**, tag 이름은 **name** 그리고 tag의 속성들은 **attrs**를 이용해 조회
- **soup.select("h1")**

## ■ 다중(그룹) 선택자 ("selector1, selector2, selectorN")

- 선택자를 ", "(comma)로 분리하여 선언하면 여러 개 선택자 적용.
- 해당하는 모든 선택자의 요소를 찾기 때문에 **select\_one()** 아닌 **select()** 메서드를 이용
- **soup.select("h1, p")**

## 선택자 종류 (Cont.)

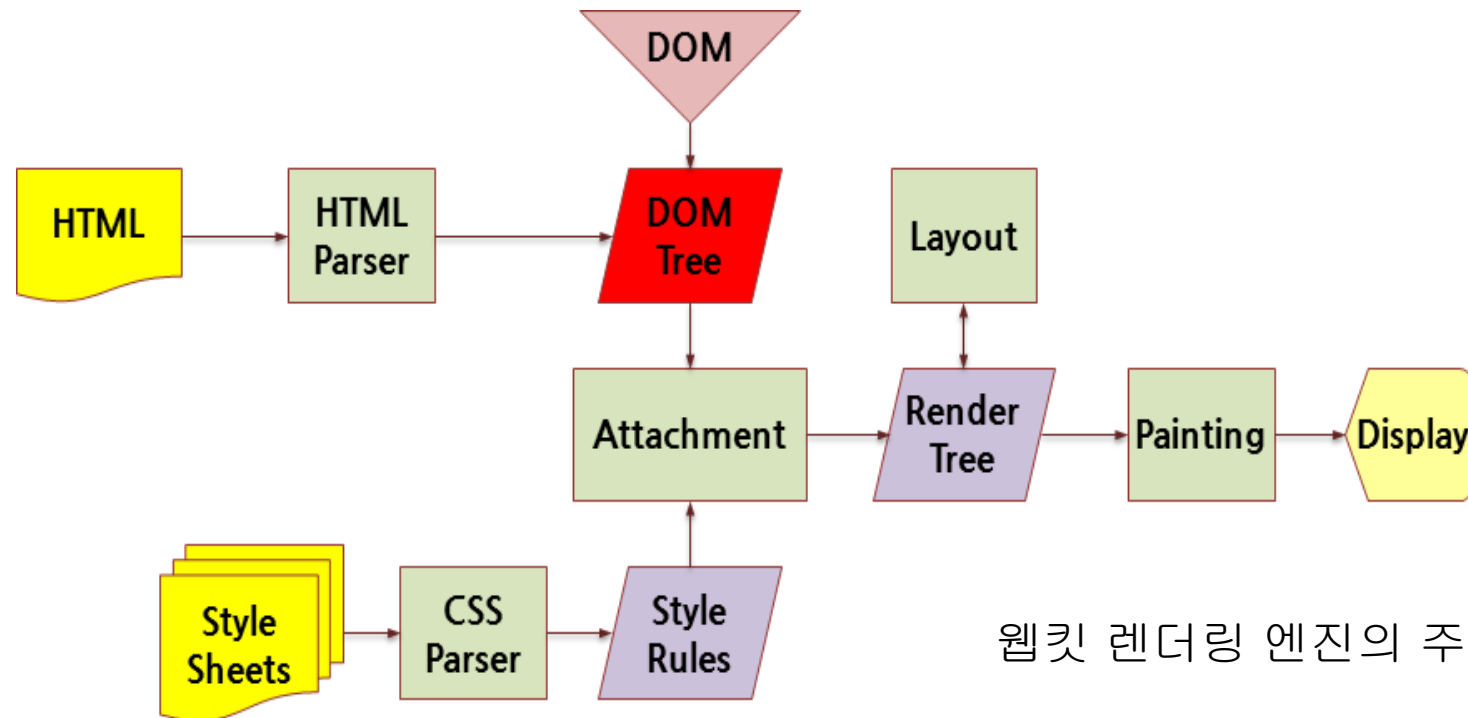
- 자손 선택자 ( " ancestor descendant " )
  - 요소가 자손 관계가 있을 때 적용시키기 위한 선택자.
  - 선택자와 선택자 사이를 공백으로 띄우고 나열
  - `soup.select("div b")`
- 자식 선택자 ("parent > child")
  - 선택자와 선택자 사이에 `>`를 입력하며 반드시 부모자식간의 관계에만 스타일이 적용되도록 함.
  - 두 단계 이상 건너뛴 관계에서는 자식 선택자가 작동하지 않음
  - `soup.select("div > b")`

# 선택자 종류 (Cont.)

- 클래스(class) 선택자 (".class")
  - HTML 문서에서 class 속성의 값과 일치하는 요소를 선택
  - 선택자 이름 앞에 "."을 이용하여 선언.
  - `soup.select("div.contents")`
- 아이디(id) 선택자 (" #id ")
  - HTML 문서에서 id 속성의 값과 일치하는 요소를 선택
  - id 선택자는 #으로 정의합니다.
  - `soup.select_one("#subject")`
- 속성 선택자 [name="value"]
  - 특정한 속성을 갖는 요소만 선택.
  - 속성 선택자는 [와 ] 사이에 속성의 이름과 값을 지정
  - `soup.select_one("[id=subject]")`

# DOM(Document Object Model, 문서 객체 모델)

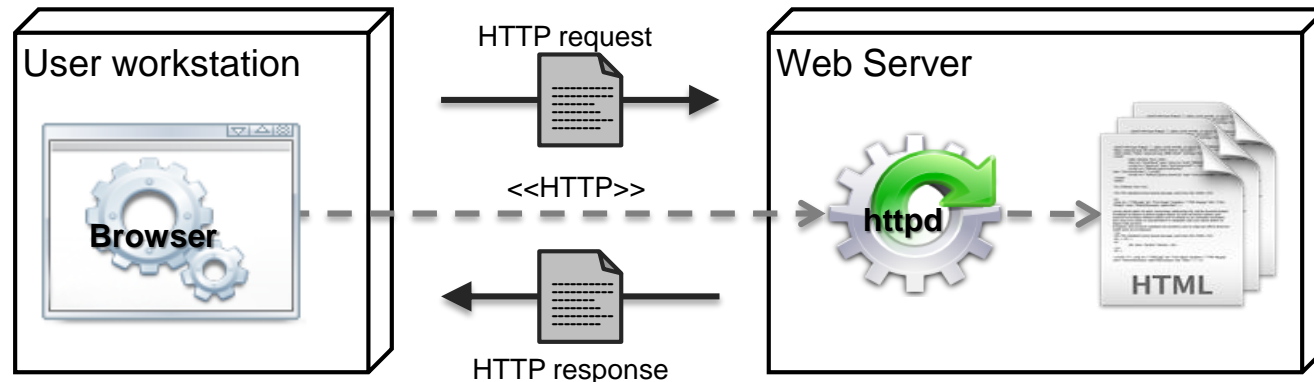
- HTML 문서를 parsing해서 만들어진 객체
- HTML 문서에서 원하는 요소를 찾기 위해서는 요소를 찾는 메소드를 실행시키기 전에 먼저 DOM이 만들어져 있어야 함



웹킷 렌더링 엔진의 주요 흐름

# HTTP Request와 HTTP Response

- Browser가 서버의 페이지를 요청하면 서버는 해당 파일을 찾은 다음 HTTP응답을 통해 클라이언트에 전송
- Browser는 응답된 페이지를 해석하여 화면에 보여줌
- request : 사용자가 원하는 파일 또는 리소스의 위치와 브라우저에 관한 정보를 포함
- response : 요청한 자원에 관한 정보를 가지고 있으며, 일반적으로 텍스트 형태이며, 그래픽 등을 바이너리 정보를 포함할 수도 있음.



# HTTP Message

- HTTP Message는 크게 요청과 응답 메시지로 구분
- HTTP Message는 크게 라인, header, body 영역으로 나뉘며, 또 여기서 header와 body 영역은 공백으로 구분됨
- Body 영역을 제외한 나머지 영역(요청, header)의 각 라인은 연속된 CR/LF(₩r₩n) 문자로 구분

HTTP Request 메시지의 구성



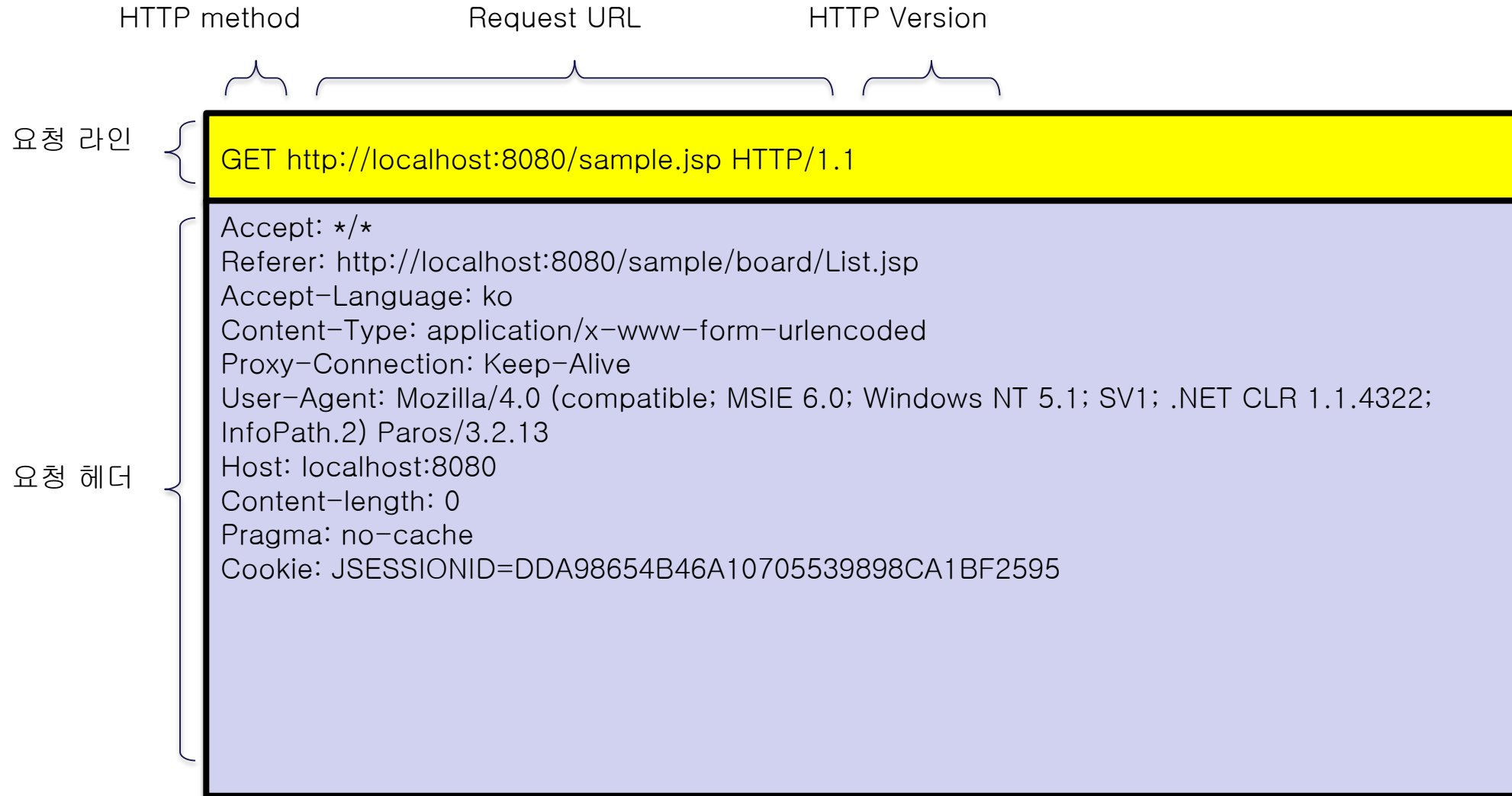
HTTP Response 메시지의 구성



# HTTP 요청 방식

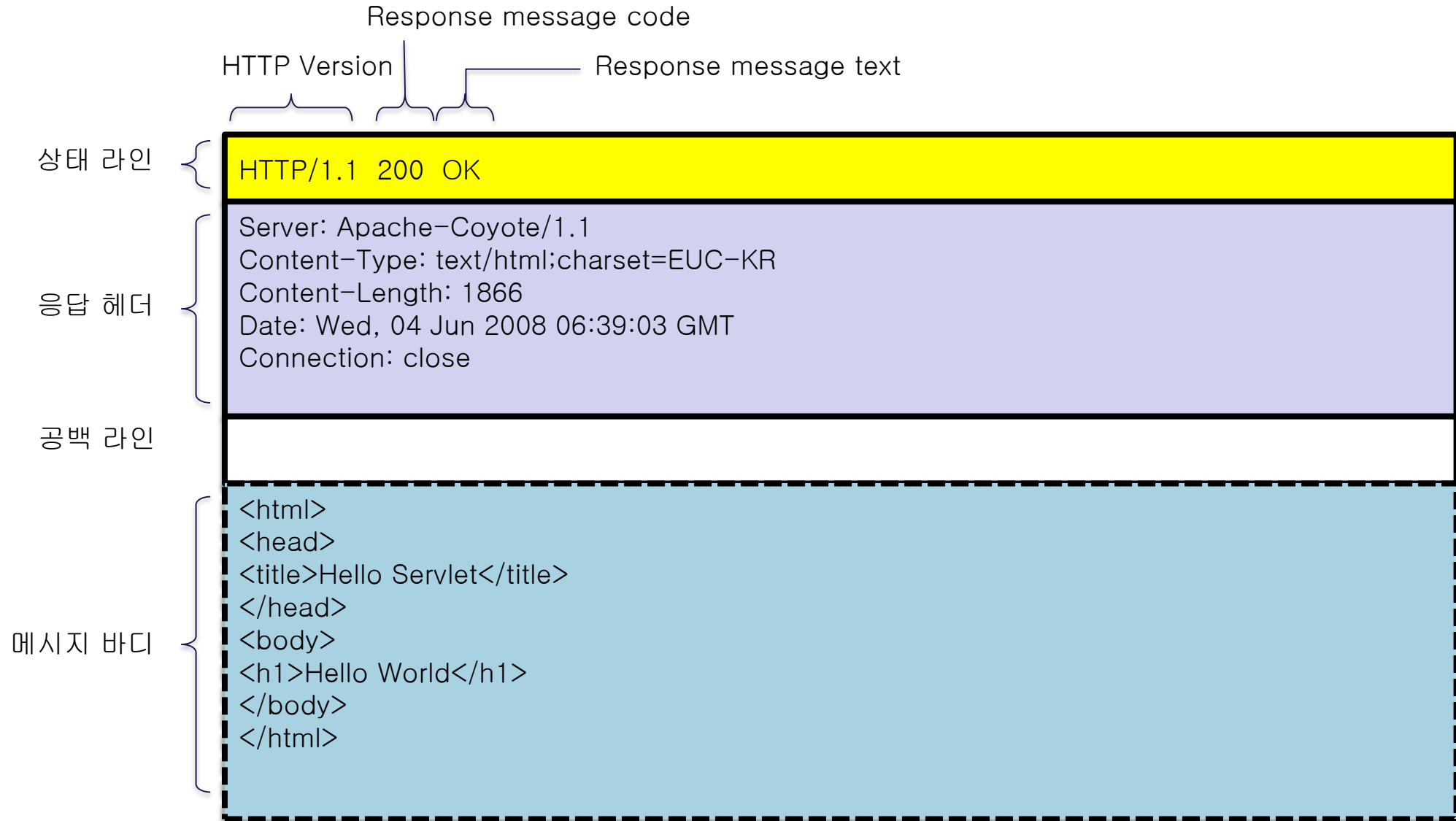
요청 방식	설 명
OPTIONS	서버 자원 및 기능검색을 위한 요청. 보통 Web 서버가 어떤 Method를 지원하는지 알기 위해 사용
GET	요청하는 주소(URL)에 의해 서버의 자원을 요청. 서버에 전달되는 해당 데이터(파라미터)가 요청 URL에 포함.
HEAD	바디 영역을 제외한 서버 정보를 요청. 요청하는 URI의 자원 유/무를 알아내기 위해 사용.
POST	GET 요청과는 달리 해당 데이터(파라미터)가 요청 URL에 포함되지 않으며 요청 바디 영역에 포함.
PUT	바디 영역의 내용을 실제 서버 URI 위치에 파일로 저장.
DELETE	서버에 저장된 자원을 삭제.
TRACE	요청 자원이 수신되는 경로를 보여줌.
CONNECT	프록시와 같은 중간 서버에 터널을 형성하여 요청하기 위해 사용

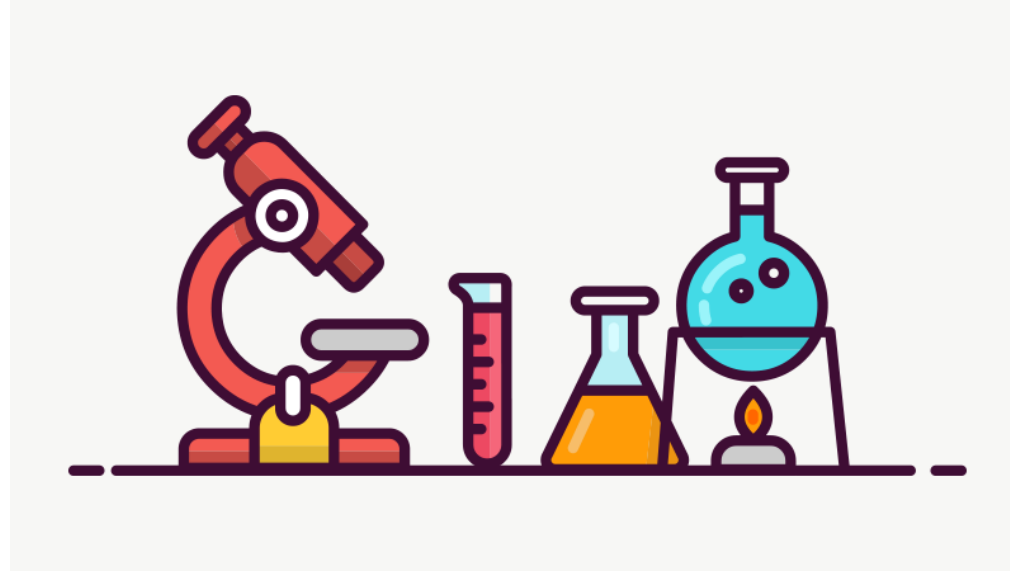
# HTTP GET Request Message





# HTTP Response Message





**Lab. Beautiful Soup 이용하기**

# Selenium을 이용한 웹 데이터 수집

# Selenium Python Binding

- 셀레니움(Selenium, <https://www.seleniumhq.org/>)은 브라우저의 동작을 자동화 해주는 프로그램
- Selenium Python bindings은 Selenium WebDriver를 사용하여 파이어폭스(Firefox), 인터넷 익스플로러(IE), 크롬(Chrome) 등 브라우저에 접근하고 브라우저의 동작을 제어 할 수 있는 편리한 API를 제공
- 설치
  - `pip install selenium`

# WebDriver

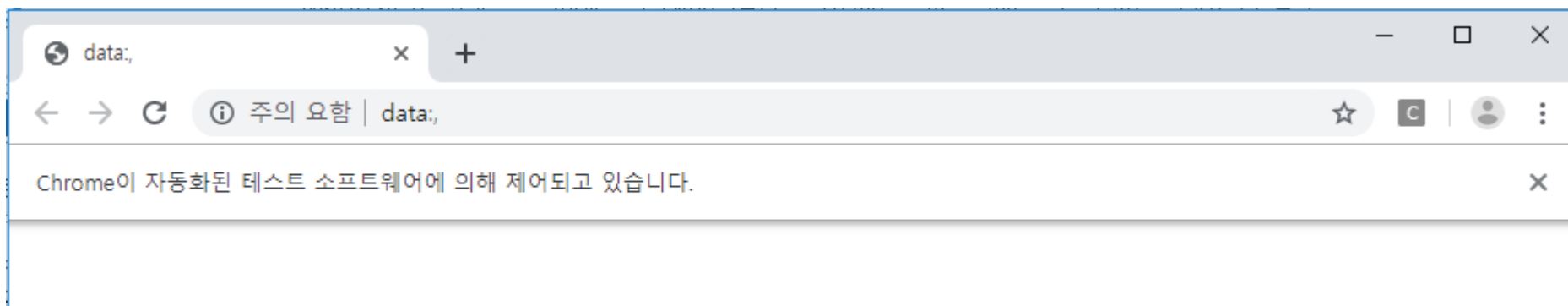
- Selenium은 WebDriver라는 것을 통해 브라우저를 제어
- WebDriver는 자신이 사용하고 싶은 브라우저에 맞는 것으로 다운로드

브라우저	주소
Chrome	<a href="https://sites.google.com/a/chromium.org/chromedriver/downloads">https://sites.google.com/a/chromium.org/chromedriver/downloads</a>
Edge	<a href="https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/">https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/</a>
Firefox	<a href="https://github.com/mozilla/geckodriver/releases">https://github.com/mozilla/geckodriver/releases</a>
Safari	<a href="https://webkit.org/blog/6900/webdriver-support-in-safari-10/">https://webkit.org/blog/6900/webdriver-support-in-safari-10/</a>

# WebDriver 실행

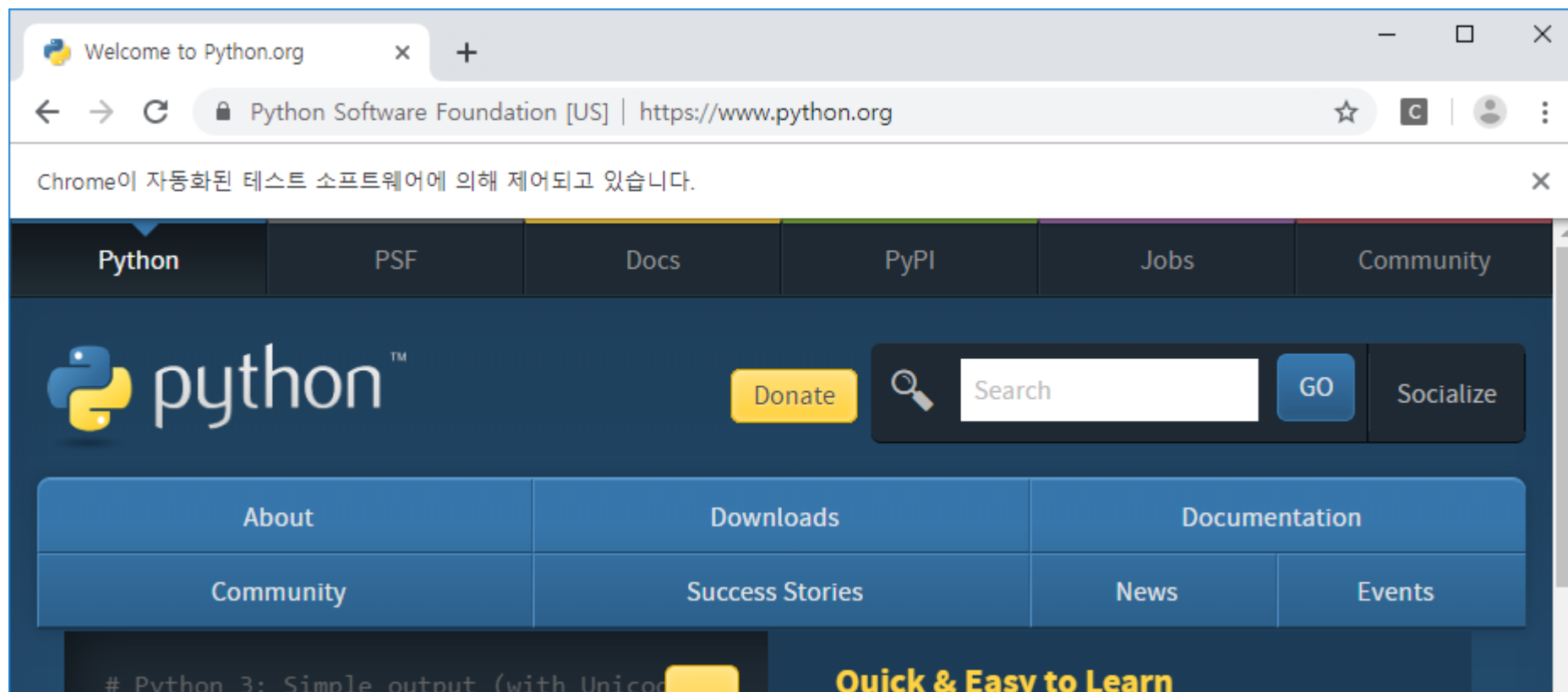
```
1 from selenium import webdriver
```

```
1 driver = webdriver.Chrome("C:/Users/user/Downloads/chromedriver")
```



# Site 접속

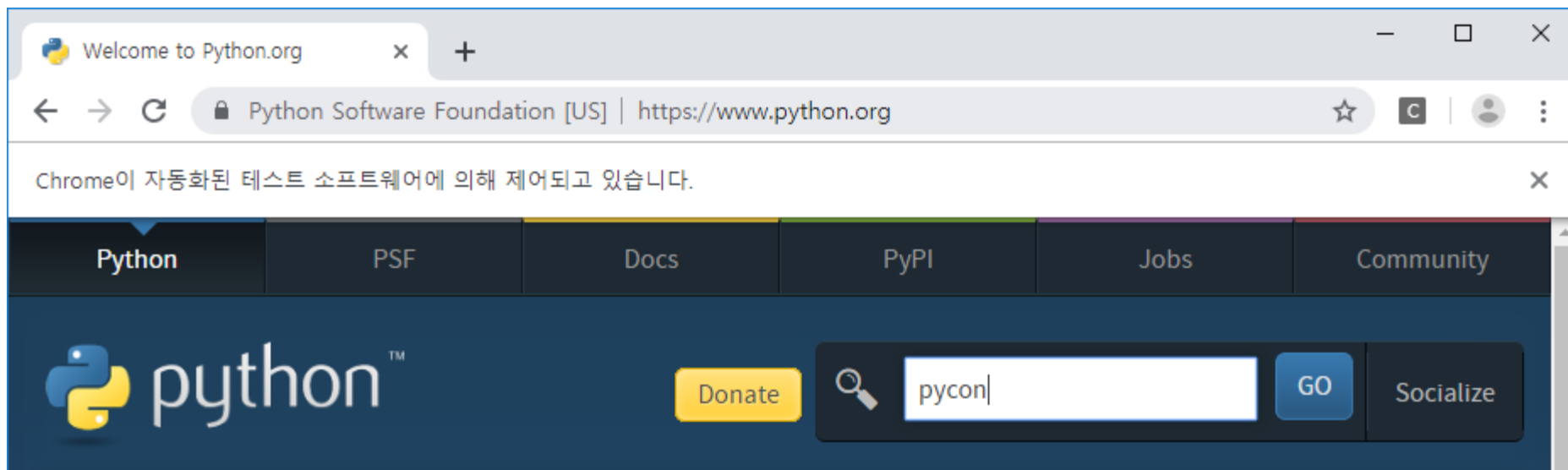
```
1 driver.get("http://www.python.org")
```



# 입력양식 채우기

```
1 elem = driver.find_element_by_name("q")
2 elem.clear()
3 elem.send_keys("pycon")
```

```
<span aria-hidden="true" class="con-search"></span>
<label class="screen-reader-text" for="id-search-field">Search This Site</label>
<input id="id-search-field" name="q" type="search" role="textbox" class="search-field" placeholder="Search" value="" tabindex="1">
<button type="submit" name="submit" id="submit" class="search-button" title="Submit this Search" tabindex="3">
  GO
</button>
```





# Event 처리하기

```
1 from selenium.webdriver.common.keys import Keys
2 elem.send_keys(Keys.RETURN)
```

Search Python.org

Search

Results

PSF PyCon Trademark Usage Policy

...PyCon AR", "PyCon Argentina" in Argentina "PyCon AU", "PyCon Australia" in Australia "PyCon BY", "PyCon Belarus" in Belarus "PyCon CA", "PyCon Canada" in Canada "PyCon CN", "PyCon China" in China "PyCon CZ", "PyCon Czech" in Czech Republic "PyCon DE" in Germany "PyCon ES", "PyCon España", "PyCon Spain" in Spain "PyCon FI", "Py...

# 결과 찾아 출력하기

```
1 result_list = driver.find_elements_by_css_selector("form h3 > a")
```

```
1 for result in result_list:  
2     print(result.text)
```

PSF PyCon Trademark Usage Policy  
Conferences and Workshops  
PyCon Italia 2016 (PyCon Sette)  
2008-04-14 PSF Board Meeting Minutes  
2012-07-16 PSF Board Meeting Minutes  
2013-02-06 PSF Board Meeting Minutes  
PyCon Home at python.org  
PyCon Australia 2013  
PyCon AU 2019  
PyCon Australia 2014

# Browser 종료

```
1 driver.close()
```

# Page와 상호작용하기

입력 양식이 아래와 같을 경우...

```
<input type="text" name="passwd" id="passwd-id" />
```

다음 중 하나를 사용하여 찾을 수 있음

```
element = driver.find_element_by_id("passwd-id")  
element = driver.find_element_by_name("passwd")  
element = driver.find_element_by_xpath("//input[@id='passwd-id']")  
element = driver.find_element_by_css_selector("#passwd-id")
```

반환 받은 요소 객체를 이용해 입력 양식에 텍스트를 입력할 수 있음

```
element.send_keys("some text")
```

# Form 서식 채우기

```
1 element = driver.find_element_by_xpath("//select[@name='name']")
2 all_options = element.find_elements_by_tag_name("option")
3
4 for option in all_options:
5     print("Value is: %s" % option.get_attribute("value"))
6     option.click()
```

```
1 from selenium.webdriver.support.ui import Select
2 select = Select(driver.find_element_by_name('name'))
3 select.select_by_index(index)
4 select.select_by_visible_text("text")
5 select.select_by_value(value)
```

요소와 상호작용할 수 있는  
**Select** 클래스를 포함

```
1 select = Select(driver.find_element_by_id('id'))
2 select.deselect_all()
```

# Form 서식 채우기 (Cont.)

```
1 select = Select(driver.find_element_by_id('id'))  
2 select.deselect_all()
```

모든 선택된 옵션의 선택 해제

```
1 all_selected_options = select.all_selected_options
```

```
1 options = select.options
```

모든 가능한 옵션

```
1 driver.find_element_by_id("submit").click()
```

submit 아이디로 버튼 클릭

```
1 element.submit()
```

모든 요소에 존재하는 submit()

# Drag & Drop

- 드래그 앤 드롭(Drag and Drop)을 사용하여 요소를 특정 양만큼 이동하거나 다른 요소로 이동

```
1 element = driver.find_element_by_name("source")
2 target = driver.find_element_by_name("target")
```

```
1 from selenium.webdriver import ActionChains
2 action_chains = ActionChains(driver)
3 action_chains.drag_and_drop(element, target).perform()
```

# Window와 Frame 이동

- `switch_to_window()` 메소드를 사용해 이름이 있는 윈도우 사이 이동

```
1 driver.switch_to_window("windowName")
```

```
1 <a href="somewhere.html" target="windowName">Click</a>
```

```
1 for handle in driver.window_handles:  
2     driver.switch_to_window(handle)
```

윈도우의 핸들을 전달 가능

```
1 driver.switch_to_frame("frameName")
```

프레임으로 이동 가능

```
1 driver.switch_to_frame("frameName.0.child")
```

.을 이용해서 서브 프레임으로 이동 가능

```
1 driver.switch_to_default_content()
```

부모 프레임으로 이동



# Popup 창

- 팝업 대화 상자 처리 기능이 내장
- warn, confirm, prompt 에서 동일하게 작동함

```
1 alert = driver.switch_to_alert()
```

# Browser History

- 방문한 브라우저 기록에서 앞뒤로 이동하려면 **forward()**, **back()** 이용

```
1 driver.forward()
```

```
1 driver.back()
```

# Cookie

- Browser의 쿠키(Cookie)를 설정하고 출력할 수 있음

```
1 cookie = { 'name' : 'foo' , 'value' : 'bar' }  
2 driver.add_cookie(cookie)
```

```
1 driver.get_cookies()
```

# 문서 내에서 요소 찾기

- Selenium은 page에서 요소를 찾기 위해 다음과 같은 메소드를 제공

단일 요소 반환	복수 요소 반환	인수
find_element_by_id	없음	id 속성 값
find_element_by_name	find_elements_by_name	name 속성 값
find_element_by_xpath	find_elements_by_xpath	XPATH
find_element_by_link_text	find_elements_by_link_text	링크 텍스트
find_element_by_partial_link_text	find_elements_by_partial_link_text	링크 텍스트
find_element_by_tag_name	find_elements_by_tag_name	태그 이름
find_element_by_class_name	find_elements_by_class_name	class 속성 값

# 대기

- Browser에서 page를 load하면 해당 page 내의 요소가 다른 시간 간격으로 load 될 수 있음
- 요소가 아직 DOM에 없는 경우 `locate()` 함수는 `ElementNotVisibleException` 예외를 발생
- 이럴 경우 대기(Wait)를 사용하여 이 문제를 해결할 수 있음

# 명시적 대기

## ■ WebDriverWait(driver, sec)

```
1  from selenium import webdriver
2  from selenium.webdriver.common.by import By
3  from selenium.webdriver.support.ui import WebDriverWait
4  from selenium.webdriver.support import expected_conditions as EC
5
6  driver = webdriver.Firefox()
7  driver.get("http://somedomain/url_that_delays_loading")
8  try:
9      element = WebDriverWait(driver, 10).until(
10         EC.presence_of_element_located((By.ID, "myDynamicElement")))
11     )
12 finally:
13     driver.quit()
```

웹 브라우저를 자동화 할 때 자주 사용되는 몇 가지 일반적인 조건

# 일반적인 조건

- 웹 브라우저를 자동화 할 때 자주 사용되는 몇 가지 일반적인 조건들을 Python에서 메소드로 제공
- **expected\_conditions** 모듈에는 **WebDriverWait**와 함께 사용할 미리 정의 된 조건 집합이 포함
  - **title\_is**
  - **title\_contains**
  - **presence\_of\_element\_located**
  - **visibility\_of\_element\_located**
  - **visibility\_of**
  - **presence\_of\_all\_elements\_located**
  - **text\_to\_be\_present\_in\_element**
  - **text\_to\_be\_present\_in\_element\_value**

## 일반적인 조건 (Cont.)

- `frame_to_be_available_and_switch_to_it`
- `invisibility_of_element_located`
- `element_to_be_clickable`
- `staleness_of`
- `element_to_be_selected`
- `element_located_to_be_selected`
- `element_selection_state_to_be`
- `element_located_selection_state_to_be`
- `alert_is_present`





## 사용자 정의 대기 조건 (Cont.)

- **implicitly\_wait()** 함수는 WebDriver가 즉시 사용할 수 없는 요소를 찾으려고 할 때 특정 시간 동안 DOM을 폴링하도록 지시

```
1 from selenium import webdriver
2
3 driver = webdriver.Firefox()
4 driver.implicitly_wait(10) # 초 단위
5 driver.get("http://somedomain/url_that_delays_loading")
6 myDynamicElement = driver.find_element_by_id("myDynamicElement")
```

# Page 객체

- Page 객체는 상호 작용하는 웹 응용 프로그램의 사용자 인터페이스의 영역을 나타냄
- Page 객체 패턴 사용의 이점
  - 여러 테스트 케이스에 걸쳐 공유 할 수 있는 재사용 가능한 코드 작성
  - 중복 된 코드의 양 줄이기
  - 사용자 인터페이스가 변경된 경우 수정 사항은 한 곳에서만 변경해야 함

# WebDriver API

- <https://selenium-python.readthedocs.io/api.html>

모듈	설명
selenium.common.exceptions	예외 클래스가 정의되어 있음
selenium.webdriver.common.action_chains	ActionChains 클래스 구현
selenium.webdriver.common.alert	Alert 클래스 구현
selenium.webdriver.common.by	By 클래스 구현
selenium.webdriver.common.desired_capabilities	DesiredCapabilities 클래스 구현
selenium.webdriver.common.keys	Keys 클래스 구현
selenium.webdriver.common.touch_actions	TouchActions 클래스 구현
selenium.webdriver.common.utils	유틸리티 메소드 구현
selenium.webdriver.common.proxy	Proxy 클래스 구현
selenium.webdriver.common.service	Service 클래스 구현
selenium.webdriver.common.html5.application_cache	ApplicationCache 클래스 구현
selenium.webdriver.support.abstract_event_listener	AbstractEventListener 클래스 구현
selenium.webdriver.support.color	Color 클래스 구현
selenium.webdriver.support.event_firing_webdriver	EventFiringWebDriver 클래스 구현
selenium.webdriver.support.expected_conditions	예상되는 조건들을 구현한 메소드
selenium.webdriver.support.select	Select 클래스 구현
selenium.webdriver.support.wait	WebDriverWait 클래스 구현
selenium.webdriver.android.webdriver	안드로이드 WebDriver 클래스 구현

# WebDriver API (Cont.)

selenium.webdriver.chrome.options	크롬 Options 클래스 구현
selenium.webdriver.chrome.service	크롬 Service 클래스 구현
selenium.webdriver.chrome.webdriver	크롬 WebDriver 클래스 구현
selenium.webdriver.firefox.extension_connection	ExtensionConnection 클래스 구현
selenium.webdriver.firefox.firefox_binary	파이어폭스 FirefoxBinary 클래스 구현
selenium.webdriver.firefox.options	파이어폭스 Options 클래스 구현
selenium.webdriver.firefox.firefox_profile	파이어폭스 FirefoxProfile 클래스 구현
selenium.webdriver.firefox.webdriver	파이어폭스 WebDriver 클래스 구현
selenium.webdriver.ie.webdriver	익스프로러 WebDriver 클래스 구현
selenium.webdriver.opera.webdriver	오페라 WebDriver 클래스 구현
selenium.webdriver.phantomjs.service	팬텀JS Service 클래스 구현
selenium.webdriver.phantomjs.webdriver	팬텀JS브라우저 WebDriver 클래스 구현
selenium.webdriver.remote.command	Command 클래스 구현
selenium.webdriver.remote.errorhandler	ErrorCode, ErrorHandler 클래스 구현
selenium.webdriver.remote.mobile	Mobile 클래스 구현
selenium.webdriver.remote.remote_connection	RemoteConnection 클래스 구현
selenium.webdriver.remote.utils	dump_json(), format_json(), load_json(), unzip_to_temp_dir()
selenium.webdriver.remote.webdriver	WebDriver 클래스 구현
selenium.webdriver.remote.webelement	WebElement 클래스 구현
selenium.webdriver.safari.service	사파리 Service 클래스 구현
selenium.webdriver.safari.webdriver	사파리 WebDriver 클래스 구현