

4. 동적 크롤링



차례

1. 정적 웹 페이지 vs 동적 웹 페이지
2. Selenium 개요
3. Selenium을 사용한 웹 페이지 스크래핑
4. 카페 및 서점 동적 웹 페이지 스크래핑 실습

1, 정적 웹 페이지 vs 동적 웹 페이지

■ 정적 웹 페이지 VS 동적 웹 페이지

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



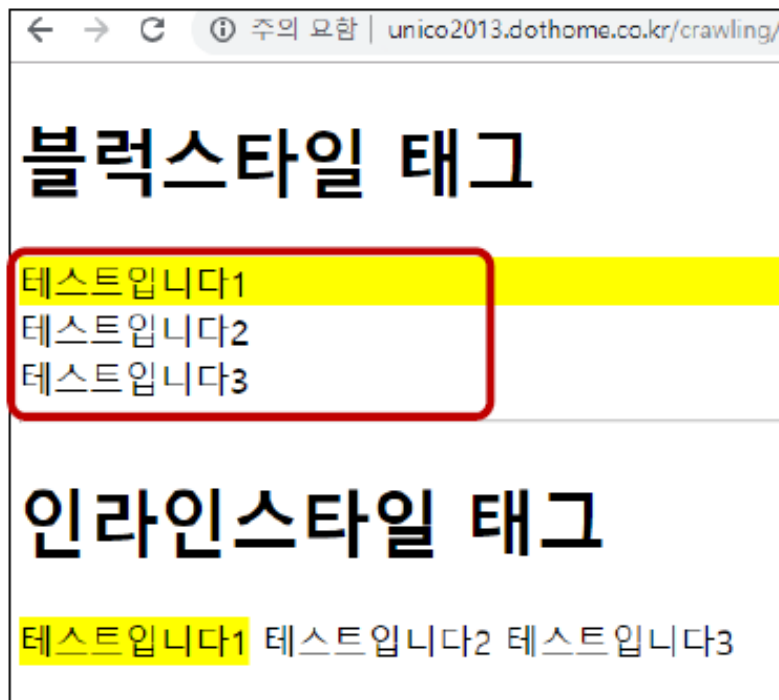
동적 웹 페이지

- 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 내용을 렌더링 시 자동 생성



1. 정적 웹 페이지 vs 동적 웹 페이지

- 정적 웹 페이지 VS 동적 웹 페이지
 - 정적 웹 페이지 화면
 - 화면에 렌더링된 각 태그의 콘텐츠가 페이지의 소스에서도 모두 보여짐

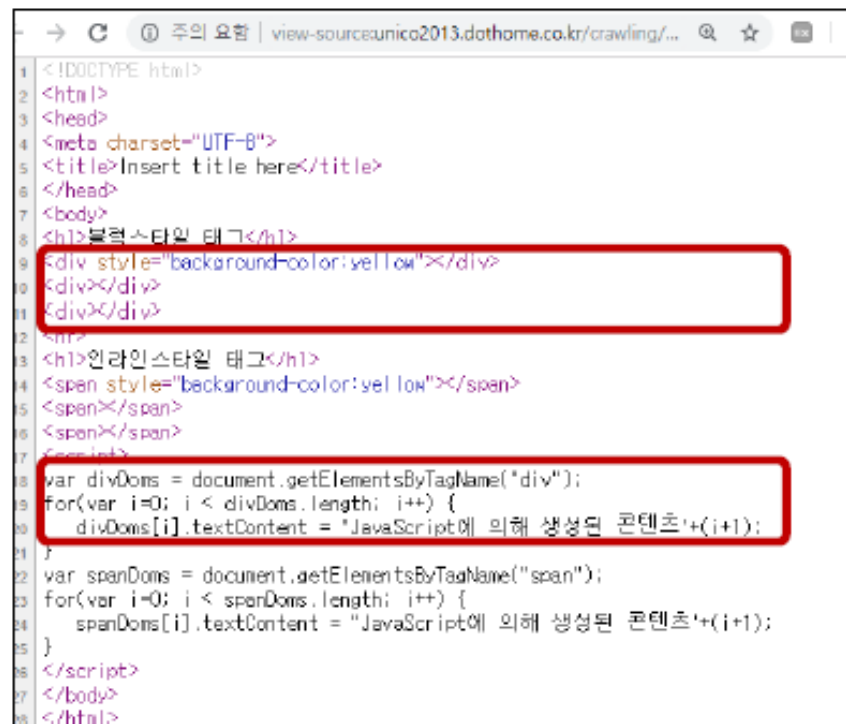


1. 정적 웹 페이지 vs 동적 웹 페이지

■ 정적 웹 페이지 VS 동적 웹 페이지

■ 동적 웹 페이지 화면

- 화면에 렌더링된 일부 태그의 콘텐츠를 페이지의 소스에서 찾아볼 수 없음
- <div> 태그나 태그처럼 소스코드에서 그내용을 찾아볼 수 없음



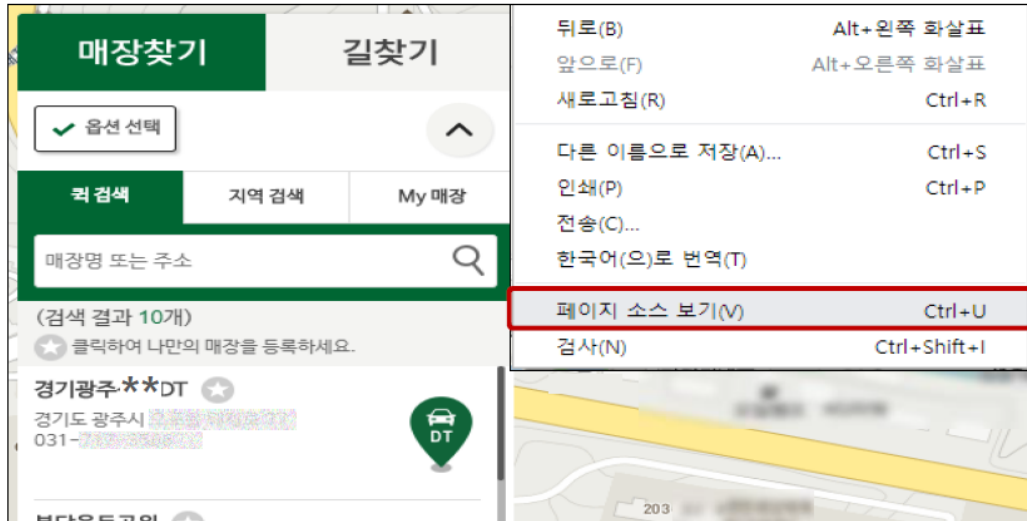
1. 정적 웹 페이지 vs 동적 웹 페이지

- 정적 웹 페이지와 동적 웹 페이지
 - 정적·동적 콘텐츠 여부 체크

소스 코드 점검을 위해 페이지에서
마우스 오른쪽 버튼 클릭

팝업 메뉴 출력

페이지 소스 보기 메뉴 클릭



오른쪽 상단의
'크롬 맞춤설정 및 제어' 메뉴 클릭

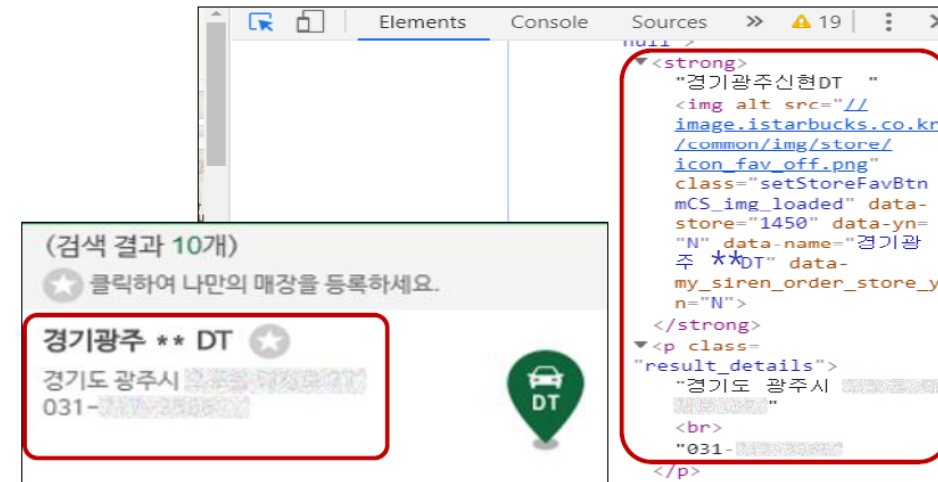
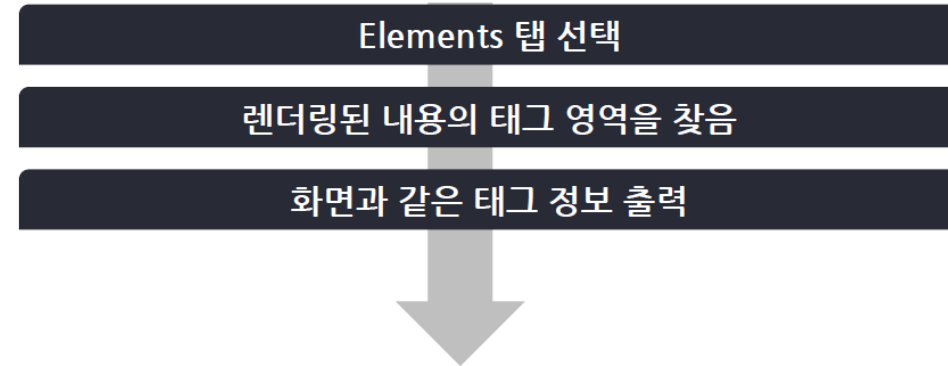
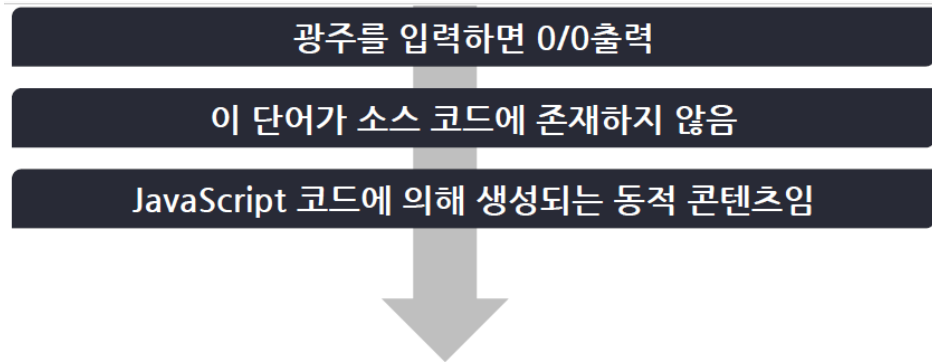
풀다운 메뉴 출력

찾기 메뉴 선택



1. 정적 웹 페이지 vs 동적 웹 페이지

- 정적 웹 페이지와 동적 웹 페이지
 - 정적·동적 콘텐츠 여부 체크



1. 정적 웹 페이지 vs 동적 웹 페이지

- 정적 웹 페이지와 동적 웹 페이지

- 정적.동적 콘텐츠 여부 체크

- 서버로부터 전송된 소스에는 없으나 렌더링된 내용에는 있는 것이 **동적 콘텐츠**
 - 이런 콘텐츠를 포함하고 있는 페이지는 **동적 웹 페이지**임

1. 정적 웹 페이지 vs 동적 웹 페이지

- 정적 웹 페이지와 동적 웹 페이지

- 동적 웹 페이지에 의해 렌더링된 동적 콘텐츠의 스크래핑

Selenium이라는 웹 브라우저를
자동화하는 도구 모음 사용

Selenium

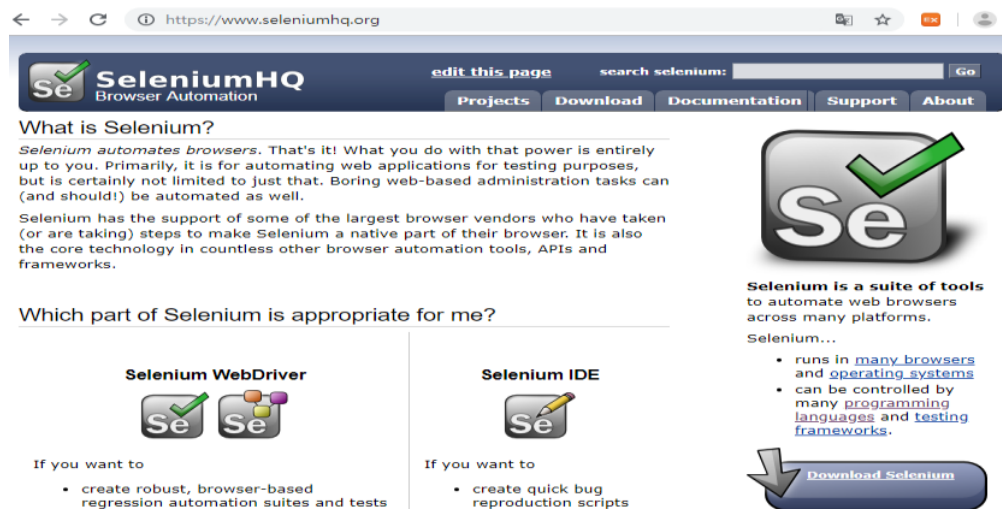
- 다양한 플랫폼과 언어 지원
- 이용하는 브라우저 자동화 도구 모음

- WebDriver라는 API를 통해 운영체제에 설치된 크롬이나 파이어폭스 등의 브라우저를 기동시키고 웹 페이지를 로드하고 제어
- 브라우저를 직접 동작시킨다는 것은 JavaScript에 의해 생성되는 콘텐츠와 Ajax 통신등을 통해 뒤늦게 브라우저에 로드되는 콘텐츠를 처리할 수 있다는것을 의미

2. Selenium 개요

■ Selenium이란?

- 주로 웹앱을 테스트하는데 이용하는 프레임워크
- Webdriver라는 API를 통해 운영체제에 설치된 Chrome 등 브라우저를 제어함
- 브라우저를 직접 동작시켜 JavaScript를 이용해 비동기적으로, 혹은 뒤늦게 불러와지는 컨텐츠들을 가져올 수 있다.
- 공식 홈페이지(<http://www.seleniumhq.org/>)
- Selenium with Python : <http://selenium-python.readthedocs.io/index.html>

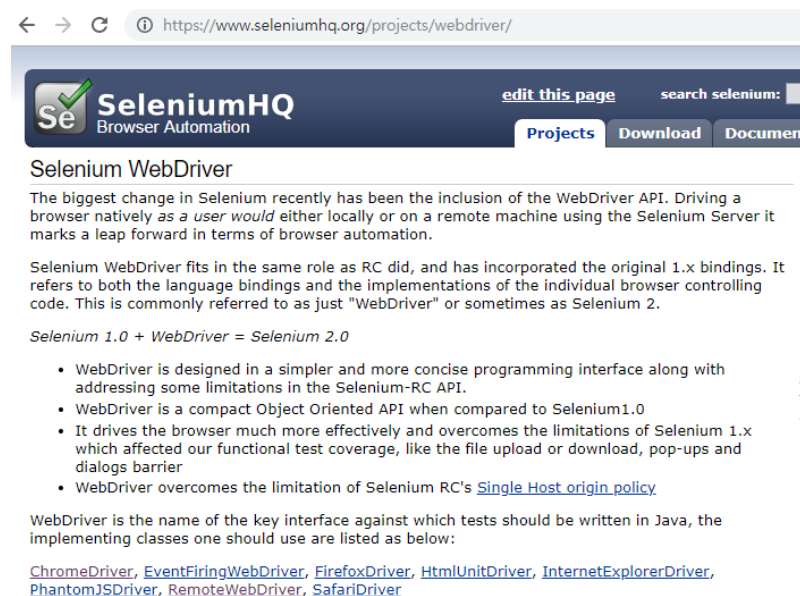


The screenshot shows the SeleniumHQ homepage. At the top, there's a navigation bar with 'SeleniumHQ Browser Automation' and links for 'Projects', 'Download', 'Documentation', 'Support', and 'About'. Below the navigation bar, the text 'What is Selenium?' is followed by a description: 'Selenium automates browsers. That's it! What you do with that power is entirely up to you. Primarily, it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) be automated as well. Selenium has the support of some of the largest browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks.'

Below this, there's a section titled 'Which part of Selenium is appropriate for me?' with two options:

- Selenium WebDriver**: If you want to create robust, browser-based regression automation suites and tests.
- Selenium IDE**: If you want to create quick bug reproduction scripts.

On the right side, there's a large 'Se' logo with a green checkmark. Below it, the text 'Selenium is a suite of tools to automate web browsers across many platforms.' is followed by a list of features: 'runs in many browsers and operating systems' and 'can be controlled by many programming languages and testing frameworks.' A 'Download Selenium' button is at the bottom.



The screenshot shows the SeleniumHQ WebDriver page. At the top, there's a navigation bar with 'SeleniumHQ Browser Automation' and links for 'Projects', 'Download', and 'Documentation'. Below the navigation bar, the text 'Selenium WebDriver' is followed by a description: 'The biggest change in Selenium recently has been the inclusion of the WebDriver API. Driving a browser natively as a user would either locally or on a remote machine using the Selenium Server it marks a leap forward in terms of browser automation.'

Below this, the text 'Selenium WebDriver fits in the same role as RC did, and has incorporated the original 1.x bindings. It refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just "WebDriver" or sometimes as Selenium 2.'

Below this, the text 'Selenium 1.0 + WebDriver = Selenium 2.0' is followed by a list of features:

- WebDriver is designed in a simpler and more concise programming interface along with addressing some limitations in the Selenium-RC API.
- WebDriver is a compact Object Oriented API when compared to Selenium1.0
- It drives the browser much more effectively and overcomes the limitations of Selenium 1.x which affected our functional test coverage, like the file upload or download, pop-ups and dialogs barrier
- WebDriver overcomes the limitation of Selenium RC's [Single Host origin policy](#)

Below this, the text 'WebDriver is the name of the key interface against which tests should be written in Java, the implementing classes one should use are listed as below:' is followed by a list of classes: [ChromeDriver](#), [EventFiringWebDriver](#), [FirefoxDriver](#), [HtmlUnitDriver](#), [InternetExplorerDriver](#), [PhantomJSDriver](#), [RemoteWebDriver](#), [SafariDriver](#)

2. Selenium 개요

- Selenium 개발환경 구축

- WebDriver API

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

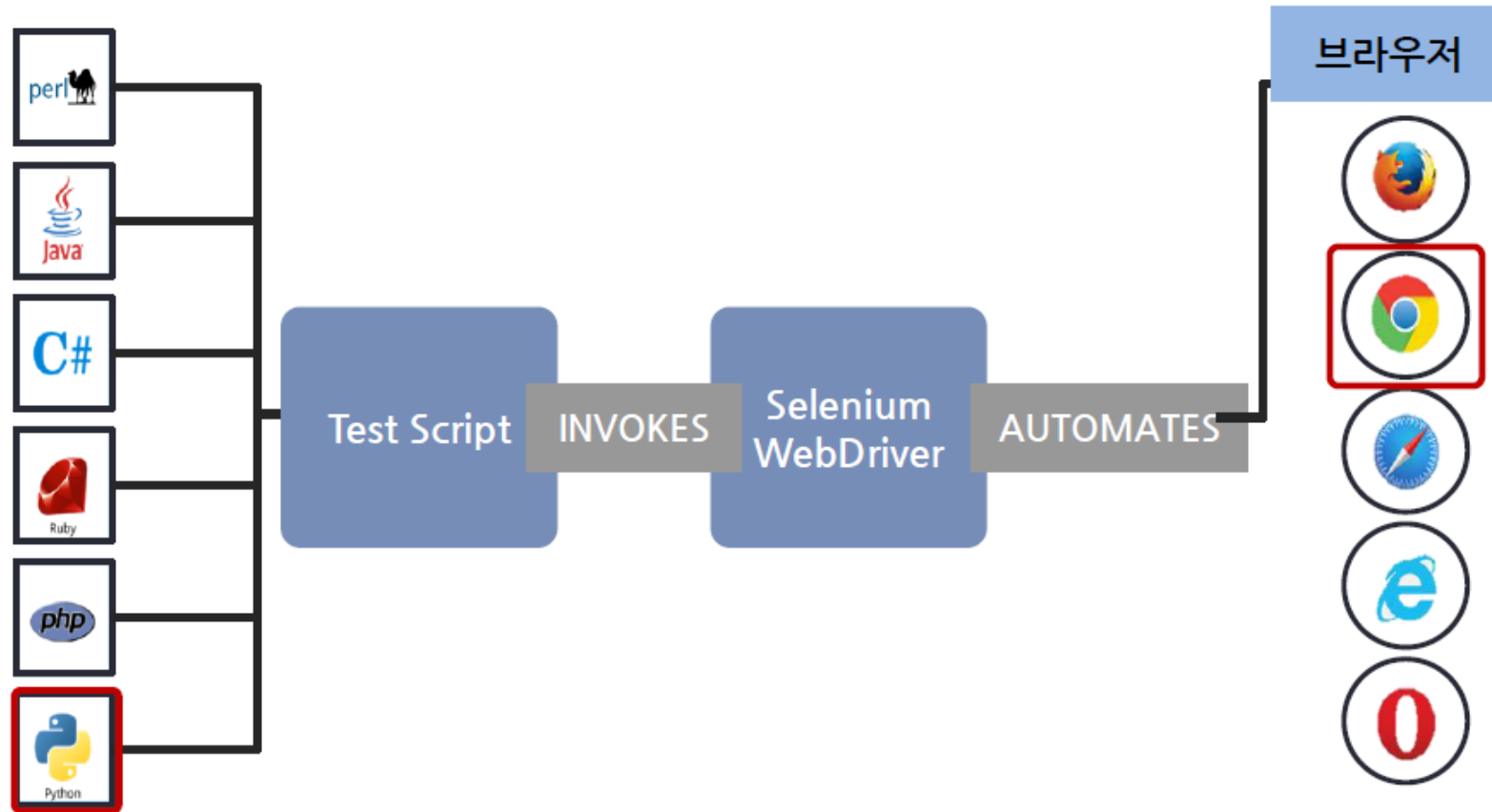
- WebDriver의 목표

- 최신 고급 웹 응용 프로그램 테스트 문제에 대한 향상된 지원과 잘 디자인된 객체지향 API 제공

- 자동화를 위한 각 브라우저의 기본 지원을 사용하여 브라우저를 직접 호출

2. Selenium 개요

- Selenium 개발환경 구축
 - WebDriver



2. Selenium 개요

- Selenium 개발환경 구축

- Selenium 설치

- cmd 창에서 pip 명령 또는 conda 명령을 통해 설치 가능

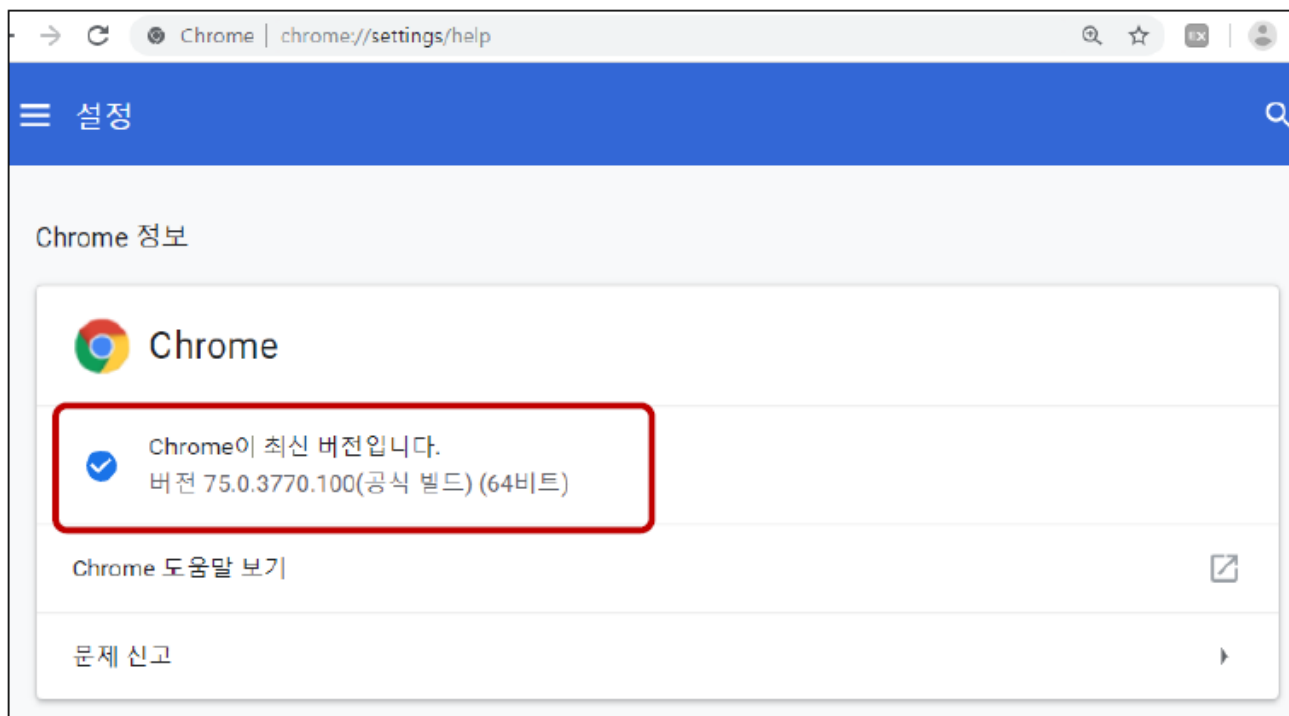
```
pip install selenium  
conda install selenium
```

- 아나콘다를 설치했으므로 conda 명령으로 설치

```
C:\Users\Samsung>conda install selenium  
Collecting package metadata: done  
Solving environment: done  
  
## Package Plan ##  
  
environment location: C:\Users\Samsung\Anaconda3  
  
added / updated specs:  
- selenium  
  
The following NEW packages will be INSTALLED:  
  
selenium           pkgs/main/win-64::selenium-3  
  
Proceed ([y]/n)? y  
  
Preparing transaction: done  
Verifying transaction: done  
Executing transaction: done
```

2. Selenium 개요

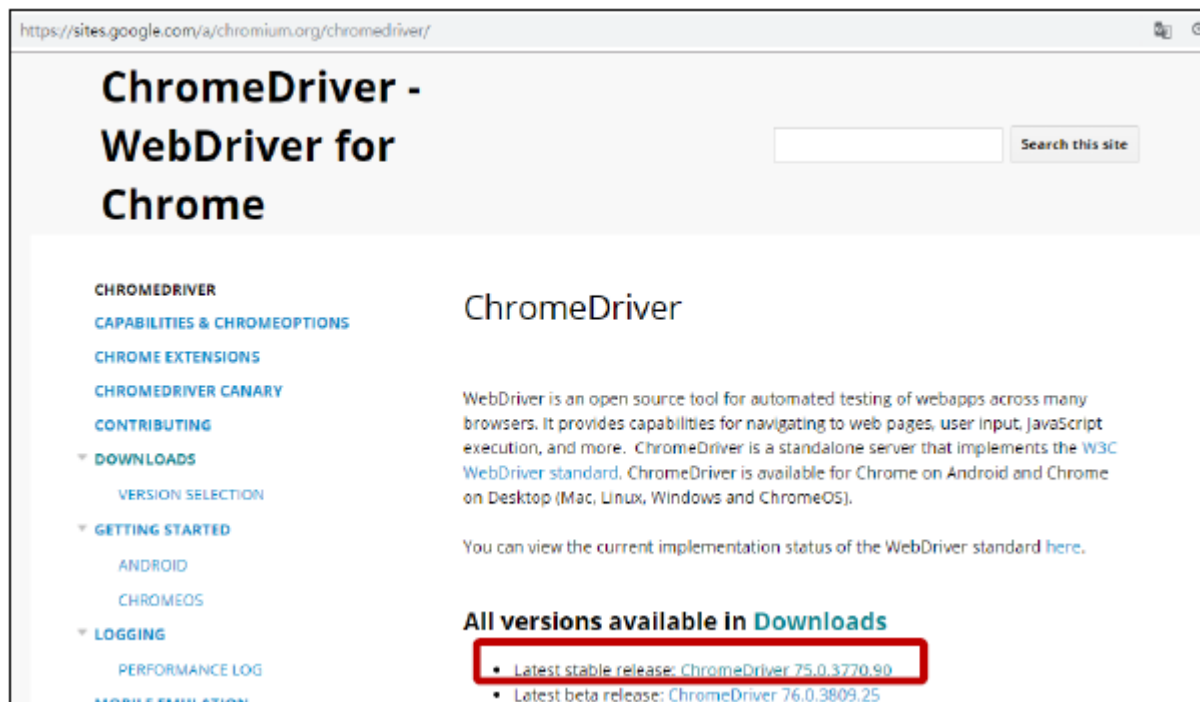
- Selenium 개발환경 구축
 - 크롬드라이버(Chrome Driver) 설치
 - Selenium의 Web Driver에 의해 제어되는 크롬 드라이버 설치를 위해 시스템에 설치된 크롬 브라우저의 버전 체크



2. Selenium 개요

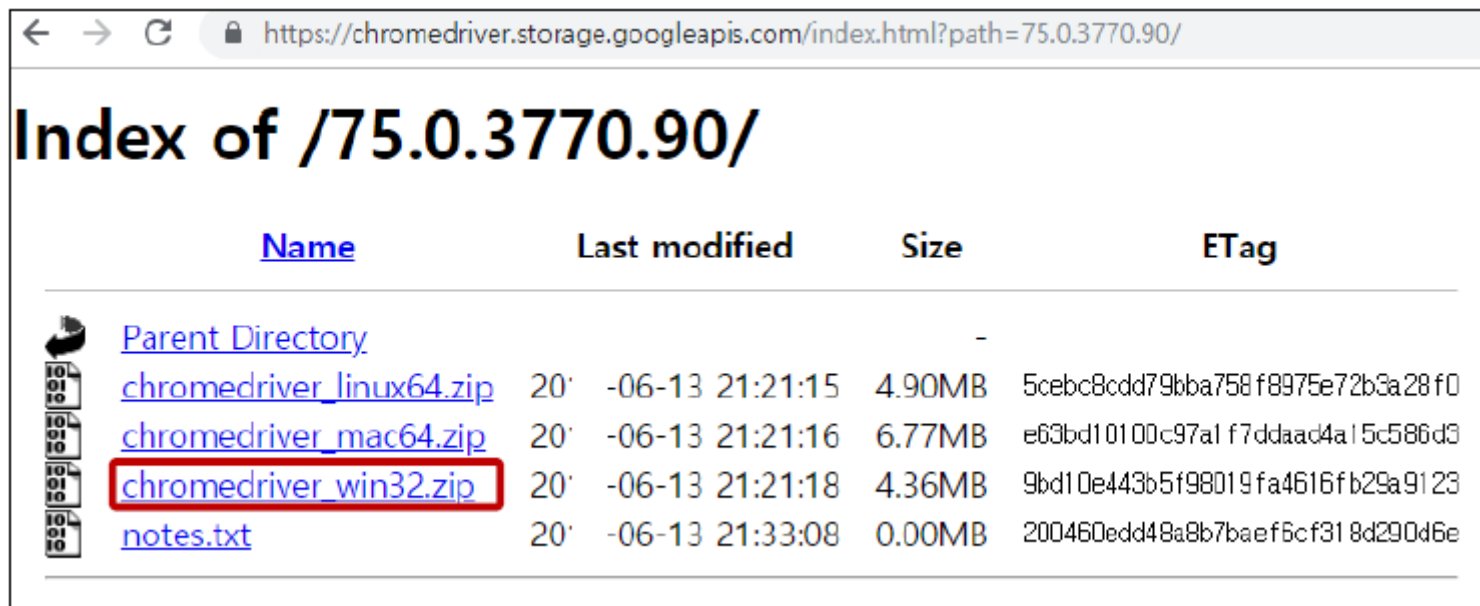
- Selenium 개발환경 구축
 - 크롬드라이버(Chrome Driver) 설치
 - 다음 사이트에서 시스템에 설치된 크롬 브라우저와 동일 버전의 링크 클릭






<https://sites.google.com/a/chromium.org/chromedriver/>



2. Selenium 개요

- Selenium 개발환경 구축
 - 크롬드라이버(Chrome Driver) 설치
 - chromedriver_win32.zip을 다운로드
 - 압축을 풀고 생성되는 chromedriver.exe를 적당한 디렉토리에 복사
 - 이 과정에서는 c:/Temp 폴더에 복사



Index of /75.0.3770.90/				
Name	Last modified	Size	ETag	
 Parent Directory		-		
 chromedriver_linux64.zip	20' -06-13 21:21:15	4.90MB	5ceb8cdd79bba758f8975e72b3a28f0	
 chromedriver_mac64.zip	20' -06-13 21:21:16	6.77MB	e63bd10100c97a1f7ddaad4a15c586d3	
 chromedriver_win32.zip	20' -06-13 21:21:18	4.36MB	9bd10e443b5f98019fa4616fb29a9123	
 notes.txt	20' -06-13 21:33:08	0.00MB	200460edd48a8b7baef6cf318d290d6e	

1. Selenium을 사용한 웹 페이지 스크래핑

- Selenium 개발환경 구축
 - Selenium을 사용한 크롬 브라우저 제어 예제 테스트

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome('C:/Temp/chromedriver')
print("WebDriver 객체 : ", type(driver))

driver.get('http://www.google.com/ncr')
target=driver.find_element_by_css_selector("[name = 'q']")
print("찾아온 태그 객체 : ", type(target))
target.send_keys('파이썬')
target.send_keys(Keys.ENTER)
#driver.quit()
```

3. Selenium을 사용한 웹 페이지 스크래핑

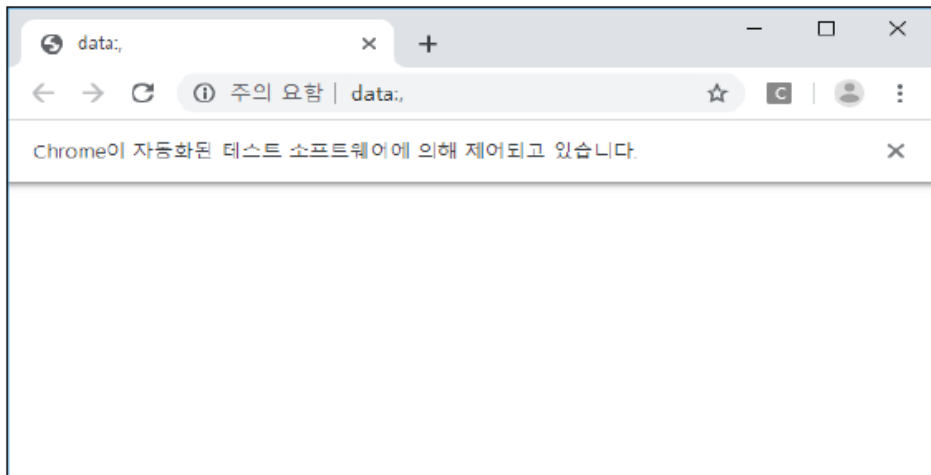
- Selenium API 소개

- WebDriver 객체 생성

- 다음 코드를 수행 시켜서 크롬드라이버를 기반으로 `selenium.webdriver.chrome.webdriver.WebDriver` 객체 생성

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

- 아규먼트로 `chromedriver.exe` 파일이 존재하는 디렉토리와 파일명에 대한 패스정보 지정 하면 Selenium에 의해 관리되는 크롬브라우저가 기동됨



3. Selenium을 사용한 웹 페이지 스크래핑

- 메서드를 사용한 웹페이지 파싱

- 페이지 가져오기

- selenium.webdriver.chrome.webdriver.WebDriver 객체의 get() 메서드사용
 - 크롤링하려는 웹페이지를 제어하는 크롬브라우저에 로드하고 렌더링

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

- WebDriver가 웹 페이지의 완전한 로드를 보장할 수 없음
 - 경우에 따라 페이지 로드 완료 또는 시작전에 WebDriver가 제어권을 반환할 수 있음
 - 견고성을 확보하려면 Explicit & Implicit Waits를 사용하여 요소가 페이지에 존재할 때까지 기다려야 함

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

3. Selenium을 사용한 웹 페이지 스크래핑

- 메서드를 사용한 웹 페이지 파싱
 - 요소 찾기

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

- 태그의 id 속성값으로 요소 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')
```

3. Selenium을 사용한 웹 페이지 스크래핑

- 메서드를 사용한 웹 페이지 파싱
 - 태그의 class 속성값으로 요소 찾기

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

- 태그명으로 요소 찾기

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

3. Selenium을 사용한 웹 페이지 스크래핑

- 메서드를 사용한 웹 페이지 파싱
 - 링크 텍스트로 태그 요소 찾기

```
byLinkText = driver.find_element_by_link_text('파이썬 학습사이트')
```

또는

```
byLinkText = driver.find_element(By.LINK_TEXT, '파이썬 학습사이트')
```

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

- 부분 링크 텍스트로 태그 요소 찾기

```
byLinkText = driver.find_elements_by_partial_link_text('사이트')
```

또는

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

3. Selenium을 사용한 웹 페이지 스크래핑

- 메서드를 사용한 웹 페이지 파싱
 - CSS 선택자로 태그 요소 찾기

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

- Xpath로 태그 요소 찾기

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

3. Selenium을 사용한 웹 페이지 스크래핑

- 메서드를 사용한 웹 페이지 파싱
 - 건에 맞는 요소 한개 찾기: WebElement 객체리턴

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

- 조건에 맞는 모든 요소 찾기: list 객체리턴

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


3. Selenium을 사용한 웹 페이지 스크래핑

- 메서드를 사용한 웹 페이지 파싱
 - 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')
```

3. Selenium을 사용한 웹 페이지 스크래핑

- URL에 접근하는 api

```
get('http://url.com')
```

- 페이지의 단일 element에 접근하는 api

```
find_element_by_name('HTML_name')  
find_element_by_id('HTML_id')  
find_element_by_xpath('/html/body/some/xpath')
```

- 페이지의 여러 elements에 접근하는 api

```
find_element_by_css_selector('#css > div.selector')  
find_element_by_class_name('some_class_name')  
find_element_by_tag_name('h1')
```

3. Selenium을 사용한 웹 페이지 스크래핑

- 실습(정적 스크래핑)

```
import urllib.request
from bs4 import BeautifulSoup
#서버 접속
server = urllib.request.urlopen("https://www.istarbucks.co.kr/store/store_map.do")

response =server.read().decode()
bs = BeautifulSoup(response, "html.parser")
li = bs.find_all('li', class_="quickResultLstCon")
print(li)
```

3. Selenium을 사용한 웹 페이지 스크래핑

- 실습(동적 스크리핑)

```
from selenium import webdriver

driver = webdriver.Chrome('C:/Temp/chromedriver')
driver.implicitly_wait(3)
driver.get("https://www.istarbucks.co.kr/store/store_map.do")
target=driver.find_element_by_class_name("quickResultLstCon")
print(type(target))
print(type(target.text))
print(target.text)
driver.quit()
```

3. Selenium을 사용한 웹 페이지 스크래핑

- 실습 : input '파이썬' 문자로 검색

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome('chromedriver')
print("webdriver 객체 : ", type(driver))

driver.get('http://www.naver.com/')
target=driver.find_element_by_css_selector("[name = 'query']") #name이 query
# target=driver.find_element_by_name( " query " ) # name가 query
# target=driver.find_element_by_id( " query " ) # id가 query
# target=driver.find_element_by_class_name("input_text") #class가 input_text

print("태그 객체 : ", type(target))
target.send_keys('파이썬')
target.send_keys(Keys.ENTER) # target.submit() 동일
driver.quit()
```

3. Selenium을 사용한 웹 페이지 스크래핑

- XPATH를 이용하여 크롤링하기

- 마크업에서 요소를 정의하기 위해 path 경로를 사용하는 방법
- find_element_by_xpath(), find_elements_by_xpath() 메서드로 검색 가능

/ : 절대경로를 나타냄

// : 문서내에서 검색

//@href : href 속성이 있는 모든 태그 선택

//a[@href='http://google.com'] : a 태그의 href 속성에 <http://google.com> 속성값을 가진 모든 태그 선택

(//a)[3] : 문서의 세 번째 링크 선택

(//table)[last()] : 문서의 마지막 테이블 선택

(//a)[position() < 3] : 문서의 처음 두 링크 선택

//table/tr/* 모든 테이블에서 모든 자식 tr 태그 선택

//div[@*] 속성이 하나라도 있는 div 태그 선택

3. Selenium을 사용한 웹 페이지 스크래핑

- 네이버 로그인하기

```
from selenium import webdriver
driver = webdriver.Chrome('chromedriver')
driver.implicitly_wait(3)
driver.get('https://nid.naver.com/nidlogin.login')
driver.find_element_by_name('id').send_keys('naver_id')
driver.find_element_by_name('pw').send_keys('password')
## 로그인 버튼을 눌러주자.
driver.find_element_by_xpath('//*[@id="log.login"]').click()
#driver.find_element_by_id('log.login').click()
```

3. Selenium을 사용한 웹 페이지 스크래핑

- 네이버 페이의 주문 내역 페이지 가져오기
 - 로그인 된 상태에서 진행

```
from bs4 import BeautifulSoup

driver.get('https://order.pay.naver.com/home')
html = driver.page_source
soup = BeautifulSoup(html, 'html.parser')
notices = soup.select('div.goods_item > div > a > p')

for n in notices:
    print(n.text.strip())
```


4. 카페 및 서점 동적 웹 페이지 스크래핑 실습

- Selenium을 활용한 웹 크롤링과 스크래핑을 고려 해야하는 경우

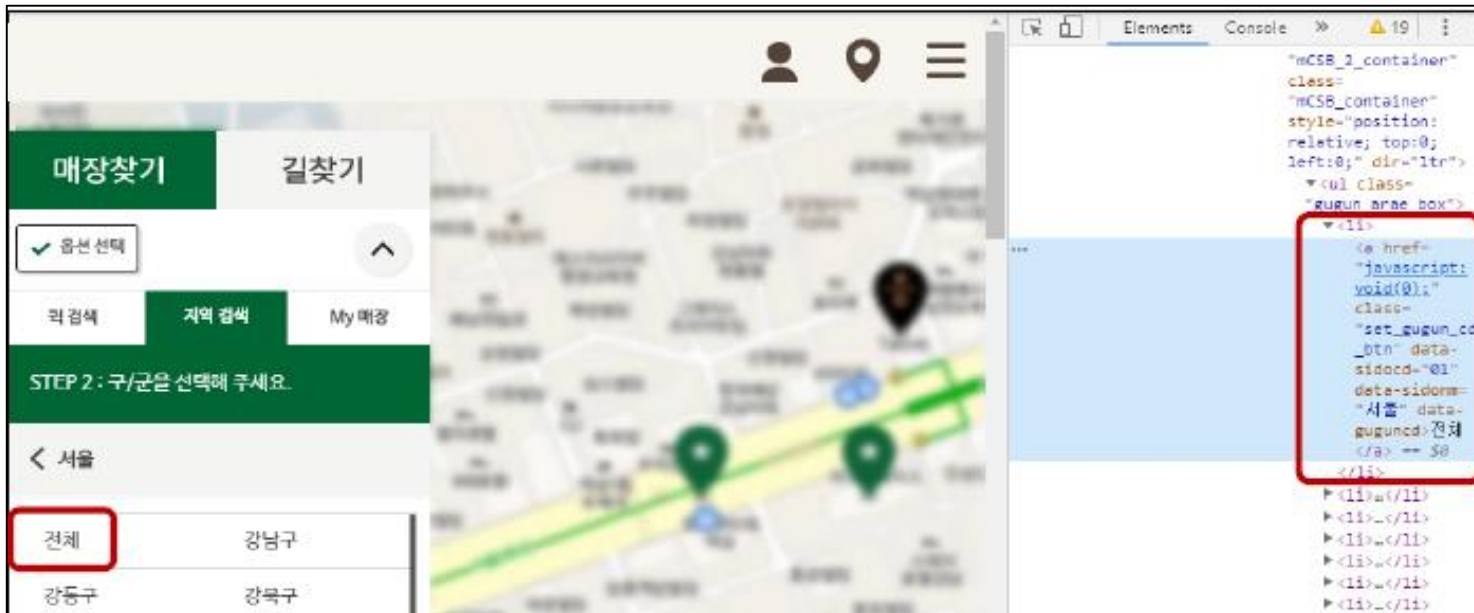
- 1 화면에 렌더링된 웹 페이지의 내용을 서버로부터 전송된 소스 코드에서 찾을 수 없는 경우
- 2 페이지 내의 링크를 클릭할 때 이동되는 페이지의 URL이 주소 필드에 출력되지 않는 경우
- 3 웹 페이지를 크롤링하기 전에 로그인 과정을 거쳐서 인증 처리를 해야 하는 경우
- 4 추출하려는 웹 페이지의 내용이 스크롤과 같은 이벤트가 발생해야 화면에 렌더링되는 경우
- 5 버튼을 클릭해야 웹 페이지의 콘텐츠가 출력되는 경우

4. 카페 및 서점 동적 웹 페이지 스크래핑 실습

- 스크래핑 내용 : 관심 지역의 카페 매장 정보 읽어 오기

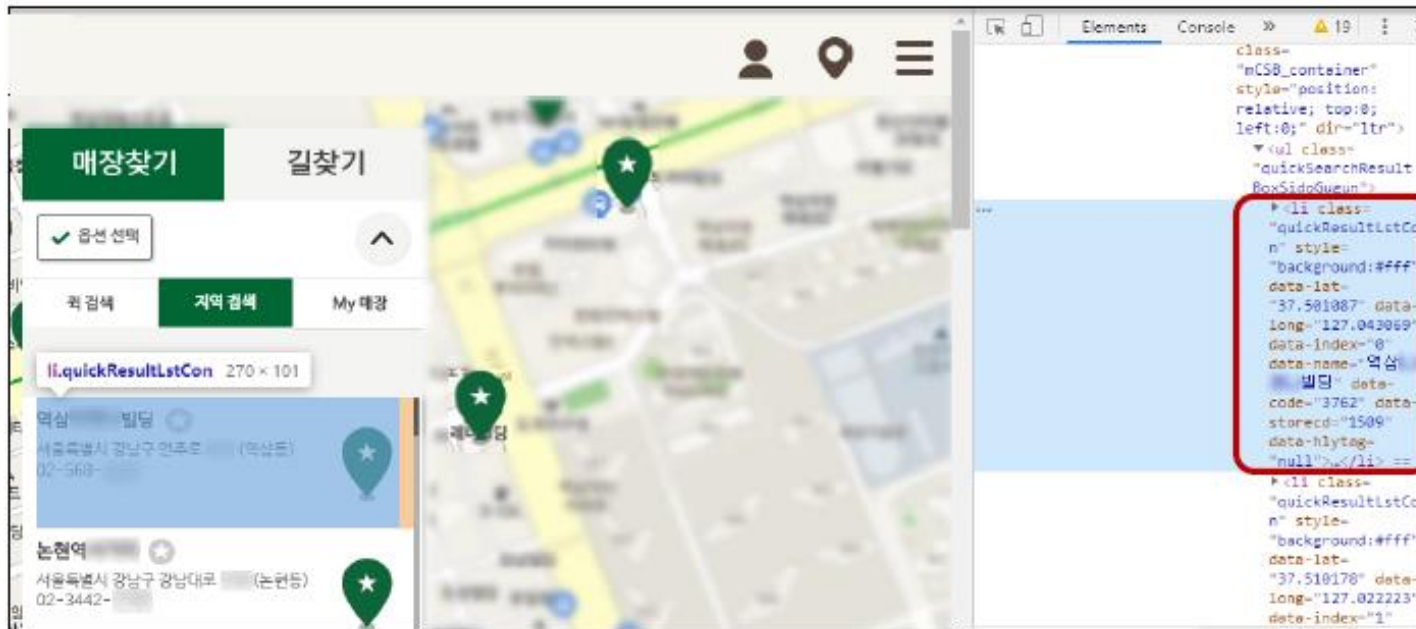
- https://www.starbucks.co.kr/store/store_map.do
- 전체의 WebElement 객체를 추출하여 클릭 이벤트 발생

```
s_link = driver.find_element_by_css_selector("#mCSB_2_container > ul > li:nth-child(1) > a")  
s_link.click()
```



- 스크래핑 내용 : 관심 지역의 카페 매장 정보 읽어 오기
 - 매장 리스트가 출력되면 다음과 같은 CSS 선택자로 매장 리스트의 WebElement 객체를 리스트 객체로 추출

```
shopList = driver.find_elements_by_css_selector("#mCSB_3_container > ul > li")
```



- 스크래핑 내용 : 관심 지역의 카페 매장 정보 읽어 오기

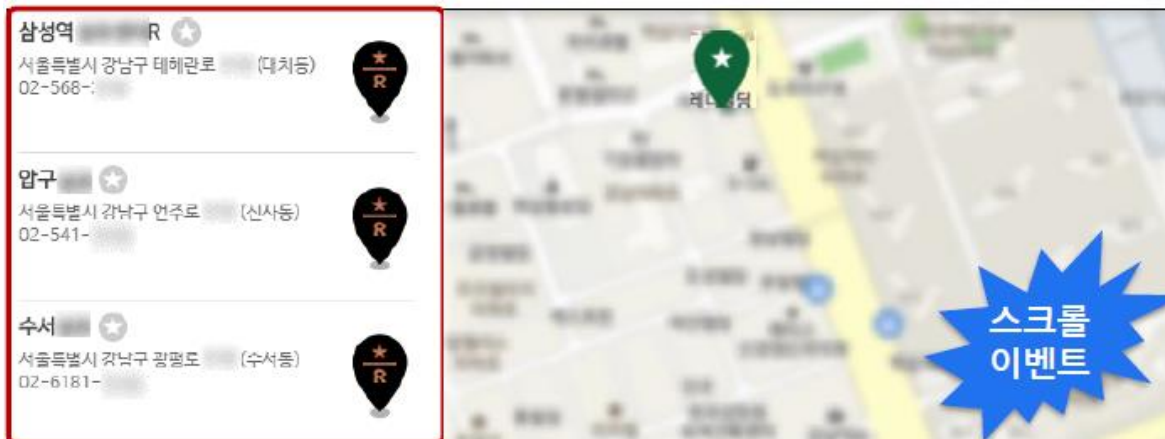
- 해당 뷰내에서 스크롤 이벤트 강제 발생

- 기본적으로 3개의 매장 정보 출력

➡ 해당 뷰의 스크롤을 아래로 내리면 그 다음 3개의 매장 정보 출력

- 화면에 렌더링된 매장 리스트의 세번째 항목에서 JavaScript 코드 실행

```
driver.execute_script("var su = arguments[0];  
var dom=document.querySelectorAll( '#mCSB_3_container > ul > li' ) [su];  
dom.scrollToView();", count)
```



```
from selenium import webdriver
import time
driver = webdriver.Chrome('chromedriver')
driver.implicitly_wait(3)
driver.get("https://www.istarbucks.co.kr/store/store_map.do")
time.sleep(2)
loca = driver.find_element_by_class_name('loca_search') # 지역 검색
loca.click()
time.sleep(2)
f_link = driver.find_element_by_css_selector("div.loca_step1_cont > ul > li:nth-child(1) > a") #서울, 부산 li:nth-child(6)
f_link.click()
time.sleep(2)
s_link = driver.find_element_by_css_selector("#mCSB_2_container > ul > li:nth-child(1) > a") # 서울지역 전체
s_link.click()
time.sleep(2)
shopList = driver.find_elements_by_css_selector("#mCSB_3_container > ul > li") # 서울전체 검색결과 매장 리스트

temp_list = []
time.sleep(3)
count = 0
total = len(shopList)
print(total)
```

```
for shop in shopList :
    count += 1
    print(count)
    shoplatt = shop.get_attribute("data-lat")
    shoplont = shop.get_attribute("data-long")
    shopname = shop.find_element_by_tag_name("strong")
    shopinfo = shop.find_element_by_tag_name("p")
    splitinfo = shopinfo.text.split('\n')
    if(len(splitinfo) == 2):
        addr = splitinfo[0]
        phonenum = splitinfo[1]
    temp_list.append([shopname.text, shoplatt, shoplont, addr, phonenum])
    if count != total and count % 3 == 0:
        driver.execute_script("var su = arguments[0]; var dom=document.querySelectorAll
            ('#mCSB_3_container > ul > li')[su]; dom.scrollToView();", count)
with open('C:/Temp/starbucks_shop.txt', "wt", encoding="utf-8") as f:
    for item in temp_list :
        f.write(str(item)+'\n')
```

- 스크래핑 내용 : 서점 사이트의 도서 댓글 읽어 오기
 - 요약 댓글과 전체 댓글의 두 가지 타입 댓글로 구성

전체 리뷰(97)

포토 리뷰(9)

스타블로거 리뷰(45)

《 < 1 2 3 4 5 6 7 8 9 10 > 》

구매리뷰

최근순

추천순

별점순

구매

지금부터 보는 눈

내용 ★★★★★ 편집/디자인 ★★★★★ | 2050-06-17

내가 살고 있는 지금 사회 문제는 무엇인지, 무엇을 고민하고 생각해봐야
많다. 바로 나와 내 가족, 내가 먹고 사는 일에 직결이 되는 일임에도 불구하고
다가오는 자동차와 같은 일들. 이것을 잡아 타면 남들보다 빠르고 편안하게 먼 길을
이러 미처 알지 못하고 있으면 놓치는 것은 물론...

이 리뷰가 도움이 되었나요?

♡0

댓글 0 >

펼쳐보기

링크를 클릭해야
전체 댓글
내용을 볼 수
있음

펼쳐보기

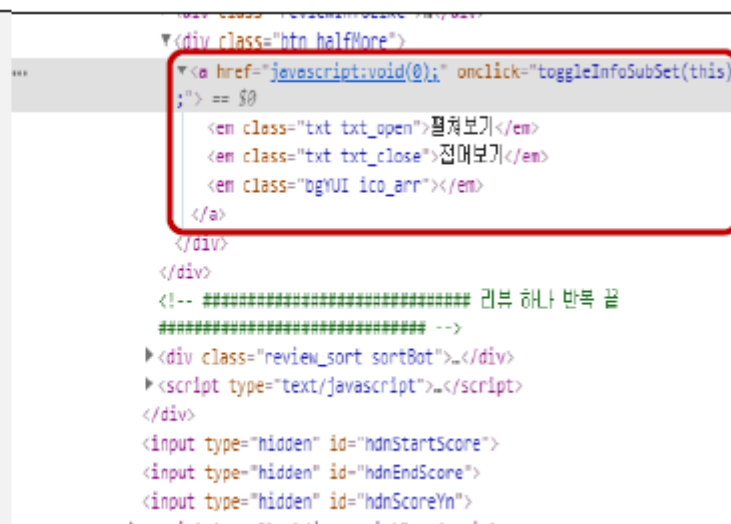
■ 스크래핑 내용 : 서점 사이트의 도서 댓글 읽어 오기

- ‘펼쳐보기’의 WebElement 객체들을 추출하여 각각 클릭 이벤트 발생

```
readLinks = driver.find_elements_by_css_selector('#infoSet_reviewContentList div.btn_halfMore > a') # 펼쳐보기
for readlink in readLinks :
```

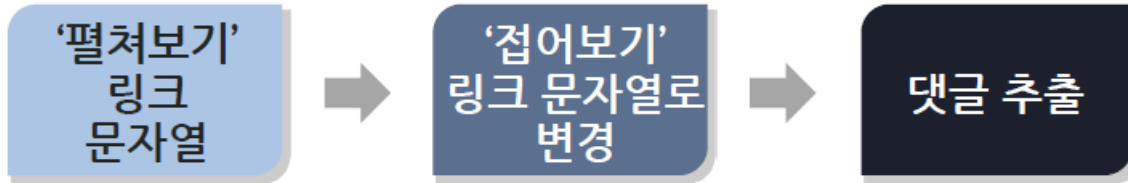
```
    readlink.click()
    time.sleep(1)
```

- 댓글 한 페이지당 5개의 댓글이 있음



- 스크래핑 내용 : 서점 사이트의 도서 댓글 읽어 오기
 - 전체 댓글이 보여지면 댓글 추출

```
reviewList = driver.find_elements_by_css_selector  
                ('#infoSet_reviewContentList div.reviewInfoBot.origin div.review_cont')
```



- 스크래핑 내용 : 서점 사이트의 도서 댓글 읽어 오기
 - 여러 페이지의 댓글 내용을 읽어오기 위해 다음 페이지 링크 클릭

전체 리뷰(97)	포토 리뷰(9)	스타블로거 리뷰(45)
<< 1 2 3 4 5 6 7 8 9 10 >> 구매리뷰 최근순 추천순 별점순		
<div>구매</div> <p>지금들 보는 눈</p> <p>내용 ★★★★★ 편집/디자인 ★★★★★ 2050-06-17</p> <p>내가 살고 있는 지금 사회 문제는 무엇인지, 무엇을 고민하고 생각해봐야 하는지 놓치는 경우가 많다. 바로 나와 내 가족, 내가 먹고 사는 일에 직결이 되는 일임에도 불구하고 전조등과 소리 없이 다가오는 자동차와 같은 일들. 이것을 잡아 타면 남들보다 빠르고 편안하게 먼 길을 갈 수 있지만, 이를 미처 알지 못하고 있으면 놓치는 것은 물론...</p> <p>이 리뷰가 도움이 되었나요? ♡0 댓글 0 펼쳐보기</p>		

- 스크래핑 내용 : 서점 사이트의 도서 댓글 읽어 오기
 - 페이지 이동을 위해 각 페이지 링크 숫자에 적용되는 URL 규칙파악

```
#infoset_reviewContentList > div.review_sort.sortBot > div.review_sortLft > div > a:nth-child(4)
#infoset_reviewContentList > div.review_sort.sortBot > div.review_sortLft > div > a:nth-child(5)
#infoset_reviewContentList > div.review_sort.sortBot > div.review_sortLft > div > a:nth-child(11)
#infoset_reviewContentList > div.review_sort.sortBot > div.review_sortLft > div > a:nth-child(12)
```



4. 카페 및 서점 동적 웹 페이지 스크래핑 실습

```
from selenium import webdriver
import time
driver = webdriver.Chrome('chromedriver')
driver.implicitly_wait(5)
driver.get("http://www.yes24.com/Product/goods/40936880")
time.sleep(2)

try :
    readLinks = driver.find_elements_by_css_selector('#infoSet_reviewContentList div.reviewInfoBot.crop > a')
    for readlink in readLinks :
        readlink.click()
        time.sleep(1)

    reviewList = driver.find_elements_by_css_selector('#infoSet_reviewContentList >
        div.reviewInfoGrp.lnkExtend.infoMoreSubOn > div.reviewInfoBot.origin > div.review_cont')
    temp_list = []
    time.sleep(3)

    for review in reviewList :
        temp_list.append(review.text)
    print(temp_list)
    stopFlag = False
```

```
while True :
    for n in range(4, 13) :
        linkurl = '#infofet_reviewContentList > div.review_sort.sortBot > div.review_sortLft > div > a:nth-child('+str(n)+' )'
        linkNum = driver.find_element_by_css_selector(linkurl)
        linkNum.click()
        time.sleep(3)
        readLinks = driver.find_elements_by_css_selector('#infofet_reviewContentList div.reviewInfoBot.crop > a')

        for readlink in readLinks :
            readlink.click()
            #driver.execute_script("arguments[0].click();", readlink)
            time.sleep(3)

        reviewList = driver.find_elements_by_css_selector('#infofet_reviewContentList > div.reviewInfoGrp.InkExtend.infoMoreSubOn >
            div.reviewInfoBot.origin > div.review_cont')
        time.sleep(2)

        for review in reviewList :
            temp_list.append(review.text)

        if len(reviewList) < 5 :
            stopFlag = True
            break

    if stopFlag == True :
        break
    nextPage = '#infofet_reviewContentList > div.review_sort.sortBot > div.review_sortLft > div > a.bgYUI.next'
    linkNum = driver.find_element_by_css_selector(nextPage)
    linkNum.click()
    time.sleep(1)
```

```
except Exception as ex:
    print("오류발생 : "+ex)
finally :
    driver.quit()

for item in temp_list :
    print(item)

wfile = open("d:/data/yes24file.txt","w")
wfile.writelines(temp_list)
wfile.close()
```

■ Selenium에서 캡처 파일 이미지로 저장

- Selenium에 의해 기동된 크롬 브라우저의 페이지 렌더링 화면을 캡처하여 이미지 파일로 저장

```
driver.get_screenshot_as_file('c:/Temp/python_main.png')
```

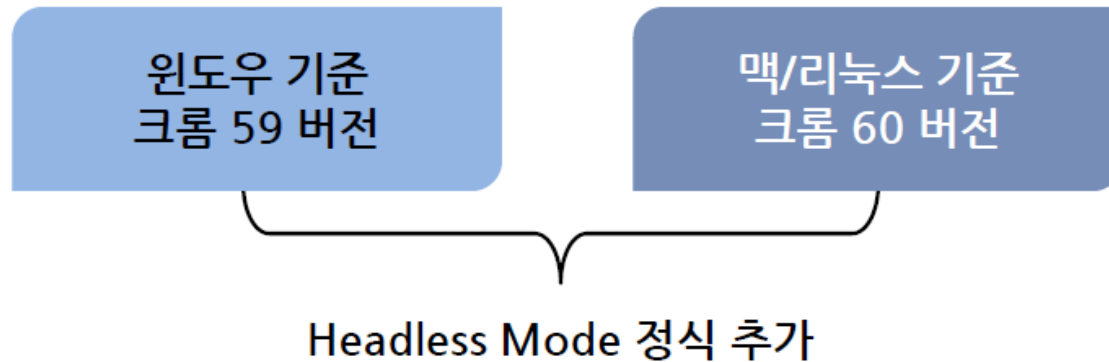
```
from selenium import webdriver

driver = webdriver.Chrome('chromedriver')
driver.get('http://www.python.org/')
driver.implicitly_wait(3)
driver.get_screenshot_as_file('c:/Temp/python_main.png')
print('캡처 저장 완료')
import time
time.sleep(2)
driver.quit()
```

■ Selenium에서 캡처 파일 이미지로 저장

■ Headless 모드

- 웹 브라우저를 기동 시키지 않고 웹 페이지 화면을 캡처하여 이미지 저장하기
- 브라우저창을 실제 운영체제의 '창'으로 띄우지 않고 웹 페이지의 화면을 그려주는 작업(렌더링)을 가상으로 진행 해주는 방법
- 실제브라우저와 동일하게 동작하지만 창은 띄지 않는방식으로 동작 가능



- 브라우저가 최신이라면 크롬의 Headless 모드를 쉽게 이용 가능

-
- Selenium에서 캡처 파일 이미지로 저장
 - Headless 모드

```
options = webdriver.ChromeOptions()  
options.add_argument('headless')  
options.add_argument('window-size=1920x1080')  
driver = webdriver.Chrome('C:/Temp/chromedriver', options=options)
```

```
from selenium import webdriver

options = webdriver.ChromeOptions()
options.add_argument('headless')
options.add_argument('window-size=1920x1080')

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

driver.get('http://www.python.org/')
driver.implicitly_wait(3)
driver.get_screenshot_as_file('c:/Temp/main_main_headless.png')
print('캡처 저장 완료')
import time
time.sleep(2)
driver.quit()
```

연습문제

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

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

driver.get('https://map.naver.com/v5/search#driver.get')#('https://map.naver.com/')
import time
time.sleep(2)
#target=driver.find_element_by_css_selector("#input_popup_selector")
#target.click()
#time.sleep(2)

target=driver.find_element_by_css_selector(".input_search")

target.send_keys('서울시어린이도서관')
target.send_keys(Keys.ENTER)

time.sleep(2)
page = 2
print("1 페이지")
```

```
while True :
    names = driver.find_elements_by_css_selector(".search_title_text")
    addrs = driver.find_elements_by_css_selector(".address")
    for i in range(len(names)) :
        print(names[i].text,addrs[i].text, sep=", ", end="\n")
    print("-----")
    page += 1
    if page >= 3 and page <= 7 :
        linkurl = '#panel div.search_result > div > div > a:nth-child('+str(page)+')'
        try :
            linkNum = driver.find_element_by_css_selector(linkurl)
            print(linkNum.text + " 페이지")
            linkNum.click()
            time.sleep(5)
        except :
            break
    if page == 7 :
        page = 2
print("스크래핑 종료")
```