04. 동적 크롤링

contents

- 동적 웹크롤링 개요
- Seleninum
- Selenium API 메서드
- 동적 크롤링

1. 동적 크롤링 개요

- **정적 웹 페이지 vs 동적 웹 페이지**

정적 웹 페이지

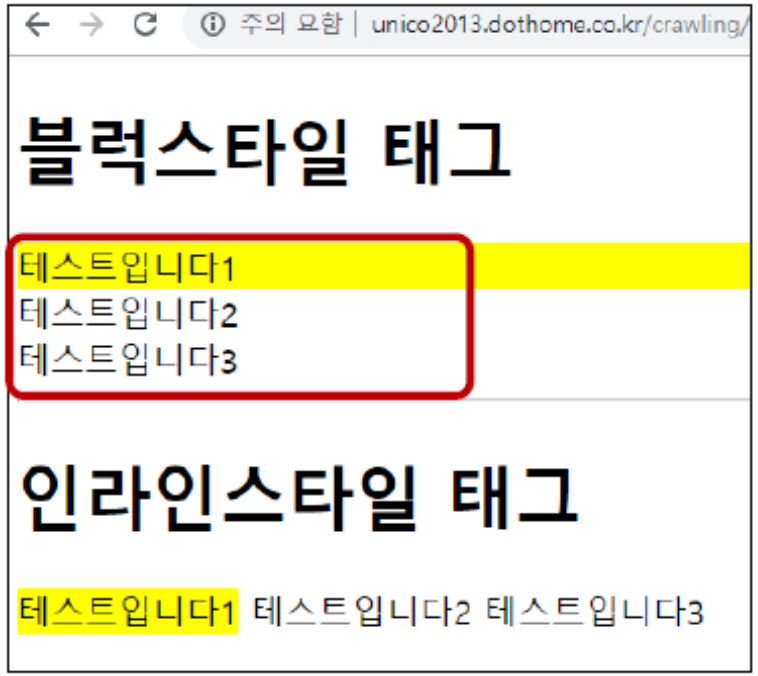
- 웹 서버에서 전송된 웹 페이지의 소스에서 화면에 렌더링된 내용을 모두 찾을 수 있는 경우
- HTML만으로 작성되거나 HTML과 CSS 기술 등으로 구현된 경우

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

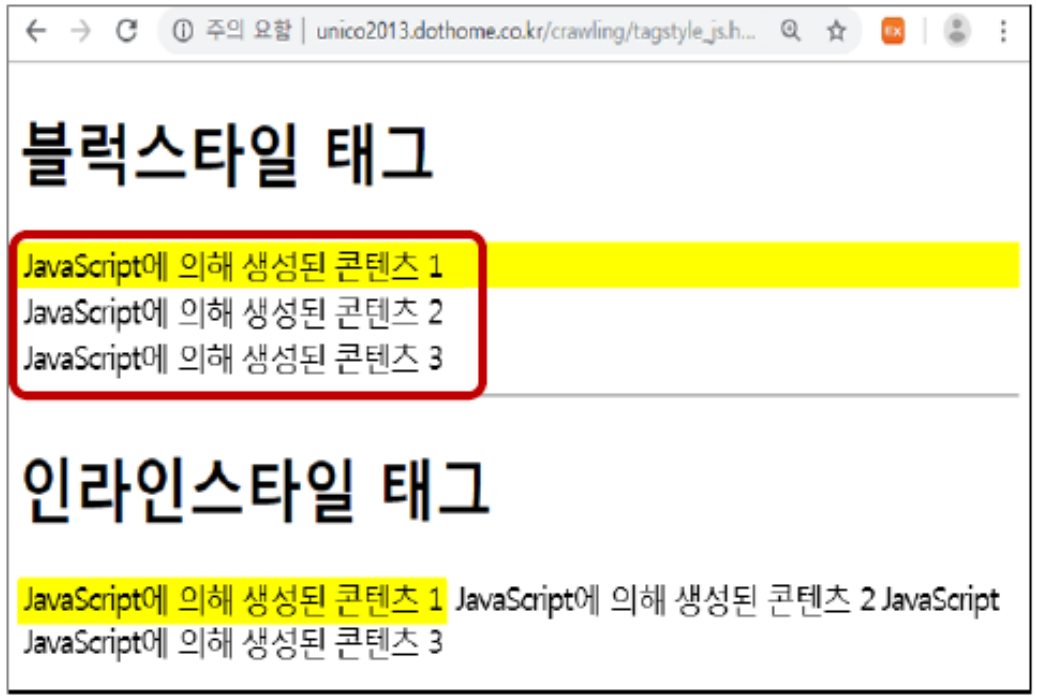
1. 동적 크롤링 개요

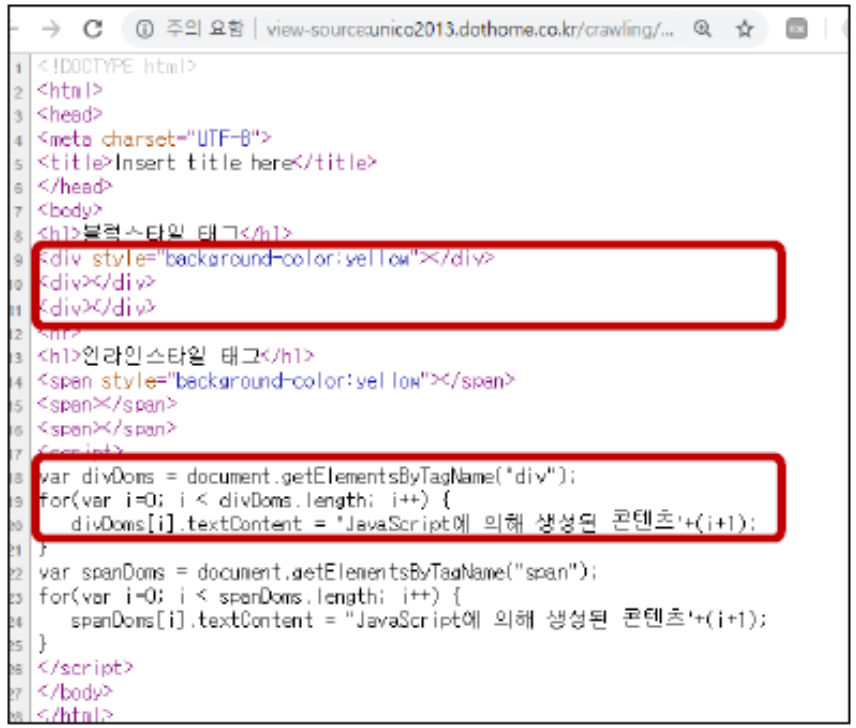
- **정적 웹 페이지 vs 동적 웹 페이지**
  - **정적 웹 페이지 화면**
    - 화면에 렌더링된 각 태그의 콘텐츠가 페이지의 소스에서도 모두 보여짐



1. 동적 크롤링 개요

- **정적 웹 페이지 vs 동적 웹 페이지**
  - **동적 웹 페이지 화면**
    - 화면에 렌더링된 일부 태그의 콘텐츠를 페이지의 소스에서 찾아볼 수 없음
    - 태그나 태그처럼 소스코드에서 그 내용을 찾아볼 수 없음





1. 동적 크롤링 개요

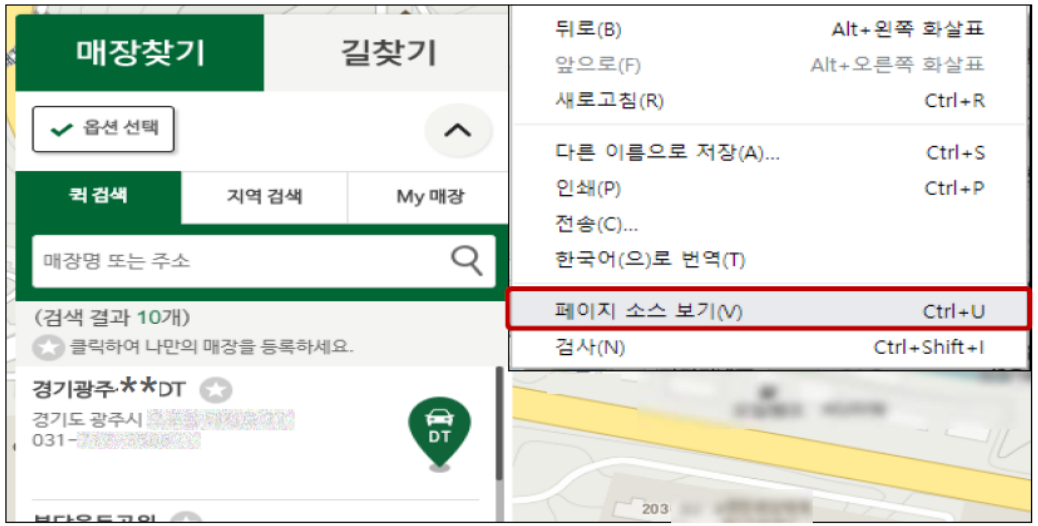
정적 웹 페이지와 동적 웹 페이지

정적 동적 콘텐츠 여부 체크

소스 코드 점검을 위해 페이지에서
마우스 오른쪽 버튼 클릭

팝업 메뉴 출력

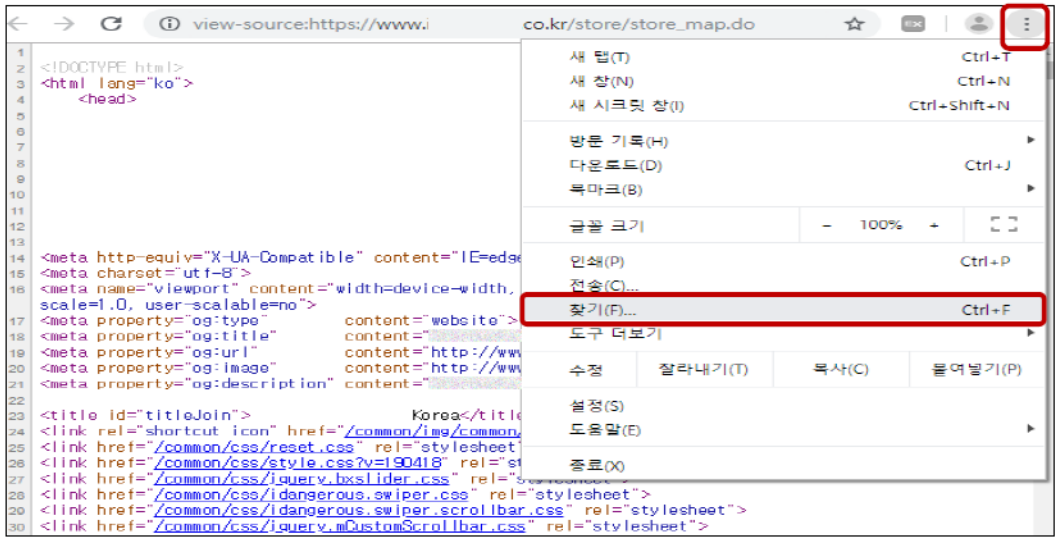
페이지 소스 보기 메뉴 클릭



오른쪽 상단의
'크롬 맞춤설정 및 제어' 메뉴 클릭

풀다운 메뉴 출력

찾기 메뉴 선택



1. 동적 크롤링 개요

정적 웹 페이지와 동적 웹 페이지

정적 동적 콘텐츠 여부 체크

광주를 입력하면 0/0출력

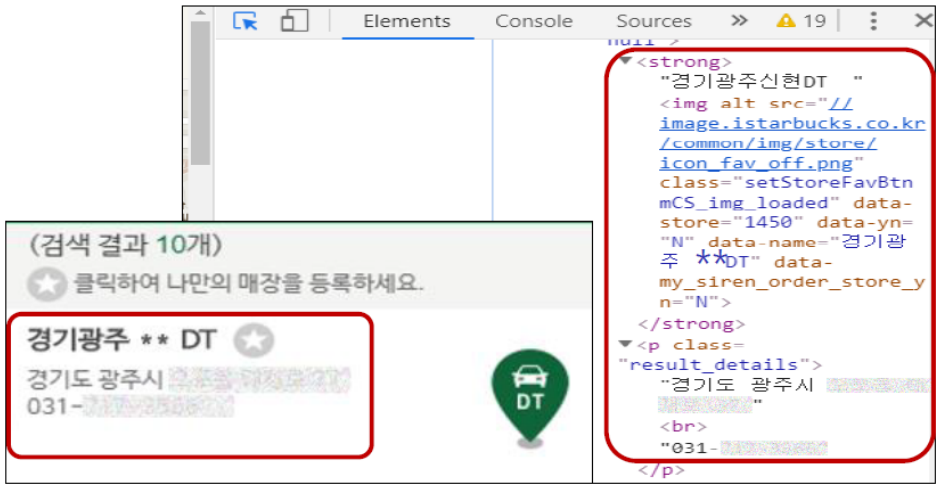
이 단어가 소스 코드에 존재하지 않음

JavaScript 코드에 의해 생성되는 동적 콘텐츠임

Elements 탭 선택

렌더링된 내용의 태그 영역을 찾음

화면과 같은 태그 정보 출력



1. 동적 크롤링 개요

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

동적 웹 페이지에 의해 렌더링된 동적 콘텐츠의 스크래핑

Selenium이라는 웹 브라우저를 자동화하는 도구 모음 사용

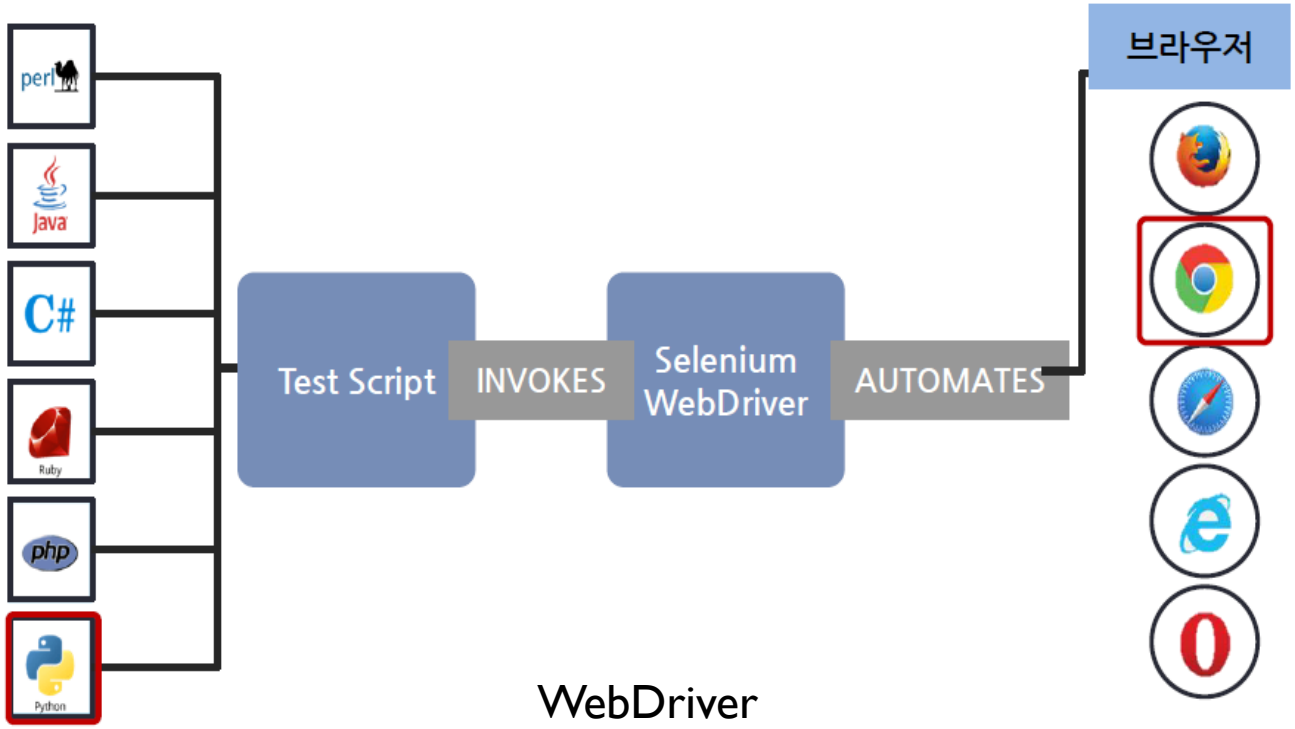
Selenium	
	▪ 다양한 플랫폼과 언어 지원 ▪ 이용하는 브라우저 자동화 도구 모음

2. Selenium

- 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

2. Selenium

- **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

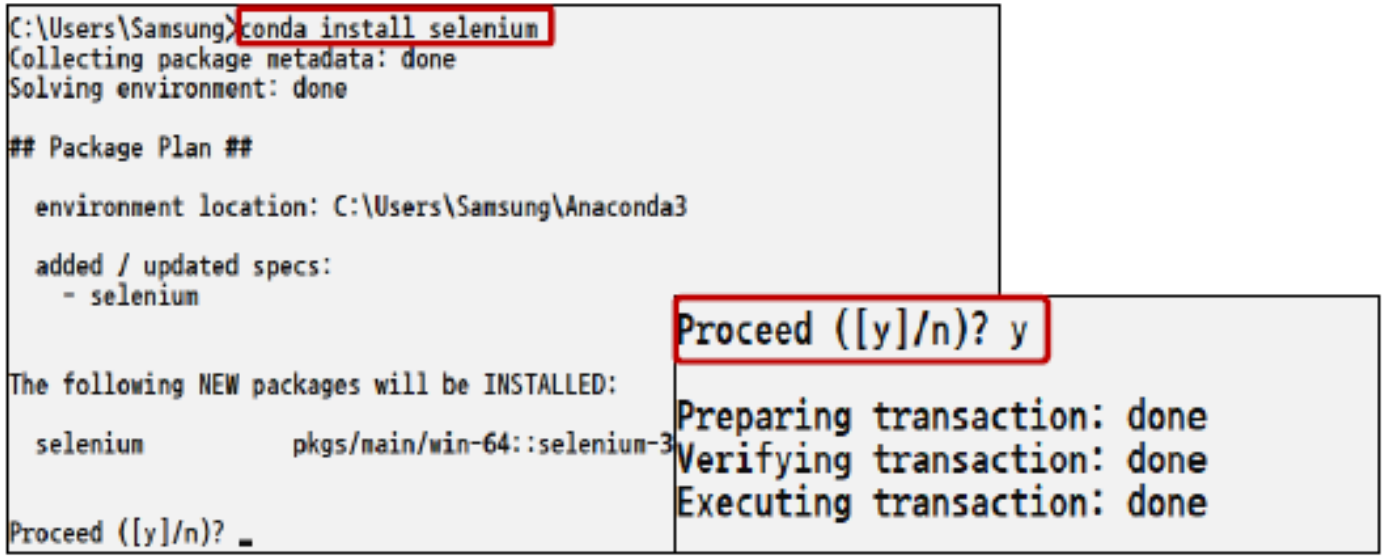


WebDriver

2. Selenium

- **Selenium 개발환경 구축**
  - Selenium 설치
    - **cmd 창에서 pip 명령 또는 conda 명령을 통해 설치 가능**

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



2. Selenium

- Selenium을 사용한 크롬 브라우저 제어 예제 테스트

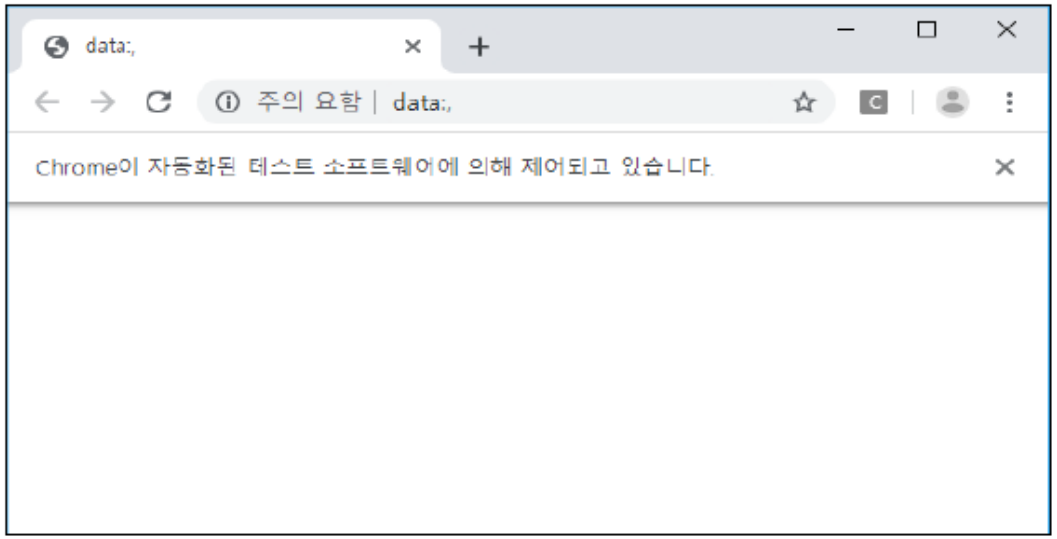
```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By

driver = webdriver.Chrome() # 드라이브 객체 생성
print("WebDriver 객체 : ", type(driver))

driver.get('http://www.google.com/ncr')   #page load
target=driver.find_element(By.CSS_SELECTOR, "[name='q']")
print("찾아온 태그 객체 : ", type(target))
target.send_keys('파이썬')   # 검색 폼에 '파이썬 검색어 보내기
target.send_keys(Keys.ENTER)  # Enter key 클릭
#driver.quit()
```

3. Selenium API 메서드

- **Selenium API 소개**
  - **WebDriver 객체 생성**
    - 다음 코드를 수행 시켜서 크롬드라이버를 기반으로 selenium.webdriver.chrome.webdriver.WebDriver 객체생성

```
driver = webdriver.Chrome()
```

    - Selenium에 의해 관리되는 크롬브라우저가 기동 됨



3. Selenium API 메서드

- **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. Selenium API 메서드

- 메서드를 사용한 웹 페이지 파싱
  - 요소 찾기

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

  - 태그의 id 속성값으로 요소 찾기

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

  - 태그의 class 속성값, 태그명으로 요소 찾기

```
target = driver.find_element(By.CLASS_NAME, "quickResultLstCon")
byTagName = driver.find_element(By.TAG_NAME, 'img')
```

3. Selenium API 메서드

- 메서드를 사용한 웹 페이지 파싱
  - 링크 텍스트 및 부분 링크 텍스트로 태그 요소 찾기

파이썬 학습 사이트

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

3. Selenium API 메서드

- 메서드를 사용한 웹 페이지 파싱
  - 조건에 맞는 요소 한 개 찾기:WebElement 객체 리턴

```
driver.find_element(By.xxx, xxx 조건에 맞는 식')
```

  - 조건에 맞는 모든 요소 찾기: list 객체 리턴

```
driver.find_elements(By.xxx, xxx 조건에 맞는 식')
```

3. Selenium API 메서드

- **메서드를 사용한 웹 페이지 파싱**
  - 요소의 정보 추출

```
element = driver.find_element(by=By.ID, value='element_id')
element.tag_name    # 태그명 추출
element.text        #텍스트 추출
element.get_attribute('속성명)    #속성값 추출
```

3. Selenium API 메서드

- **메서드를 사용한 웹 페이지 파싱(다양한 파싱 메소드 실습)**

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

driver = webdriver.Chrome()
print("webdriver 객체 : ", type(driver))
driver.get('http://www.naver.com/')
#target=driver.find_element(By.CSS_SELECTOR, "[name='query']")
#target=driver.find_element(By.NAME,"query")
#target=driver.find_element(By.ID,"query")
target=driver.find_element(By.CLASS_NAME,"search_input")
print("태그 객체 : ", type(target))
target.send_keys('파이썬')
target.send_keys(Keys.ENTER)
#driver.quit()
```

4. 동적 크롤링

- **동적 객에 대한 정적 스크래핑 실습**

```
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)
```

4. 동적 크롤링

- **실습(동적 스크리팽)**

```
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.implicitly_wait(3)
driver.get("https://www.istarbucks.co.kr/store/store_map.do")
target=driver.find_element(By.CLASS_NAME,"quickResultLstCon") # 객체 1개 추출
targets=driver.find_elements(By.CLASS_NAME,"quickResultLstCon") # 다수의 객체 추출
print(type(target))
print(type(target.text))
print(target.text)
print("*" *30)
for stor in targets:
    print(stor.text)
driver.quit()
```

5. 동적 크롤링

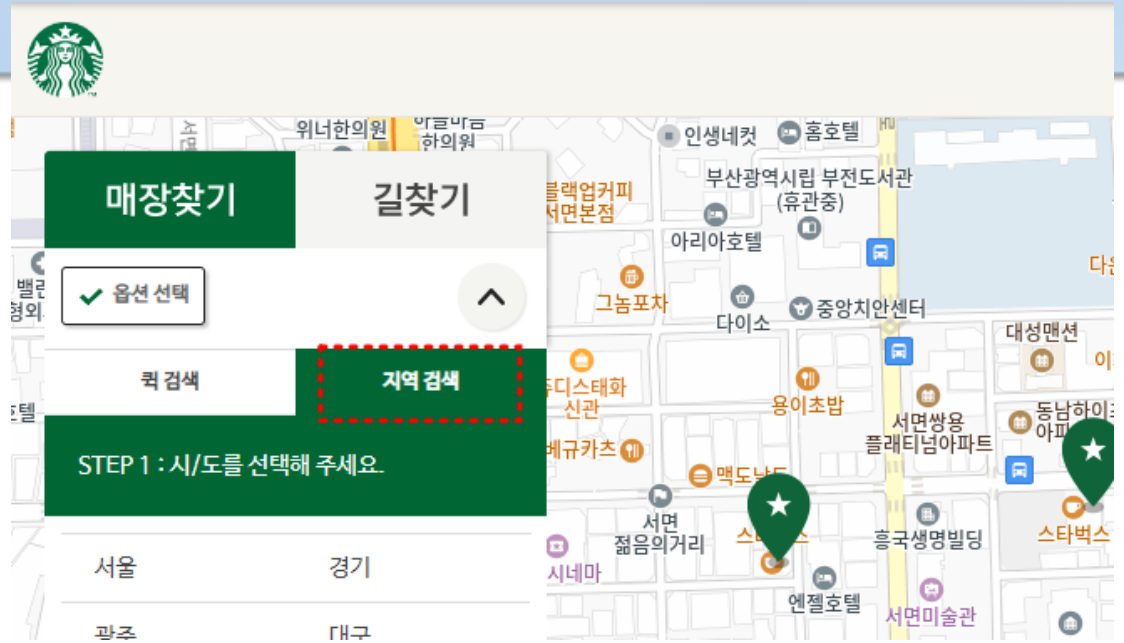
- **카페 및 서점 동적 웹 페이지 스크래핑 실습**
  - Selenium을 활용한 웹 크롤링과 스크래핑을 고려 해야하는 경우

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

5. 동적 크롤링

- **스크래핑 내용 : 관심 지역의 카페 매장 정보 읽어 오기**
  - https://www.starbucks.co.kr/store/store_map.do
  - **WebElement 객체를 추출하여 클릭 이벤트 발생(지역검색)**

```
s_link = driver.find_element(By.SELECTOR, #container > div > form > fieldset > div > section >
article.find_store_cont > article > header.loca_search > h3 > a")
s_link.click()
```



5. 동적 크롤링

- **스크래핑 내용 : 관심 지역의 카페 매장 정보 읽어 오기**
  - 서울지역 객체 추출 후 클릭
  - 전체 객체 추출 후 클릭
  - 출력된 CSS 선택자로 매장 리스트 추출

```
s_link2 = driver.find_element(By.CSS_SELECTOR, "div.loca_step1 > div.loca_step1_cont > ul > li:nth-child(1)")
s_link2.click()
```

```
s_link3 = driver.find_element(By.CSS_SELECTOR, "#mCSB_2_container > ul > li:nth-child(1) > a")
s_link3.click()
```

```
shop_list=driver.find_elements(By.CSS_SELECTOR,"#mCSB_3_container > ul > li")
for shop in shop_list:
    print(shop.text)
```

5. 동적 크롤링

- **스크래핑 내용 : 관심 지역의 카페 매장 정보 읽어 오기**
  - 매장리스트에서 매장명, 위도, 경도, 주소, 전화번호 데이터를 추출하여 리스트에 추가

```
temp_list = []
time.sleep(3)
count = 0
total = len(shop_list)
print(total)
for shop in shop_list :
    count += 1
    shoplat = shop.get_attribute("data-lat")
    shoplong = 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= (shopname.text, shoplat, shoplong, addr, phonenum)
    temp_list.append(temp))
print(temp_list)
```