Ask Company

크롤링 차근차근 시작하기 (2/E) - 중급편

Selenium 환경 구축 및 WebDriver

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

Selenium

브라우저 자동화 라이브러리

주로 브라우저를 통한 UI 테스트에서 사용

다양한 브라우저/OS/언어 지원

Browsers

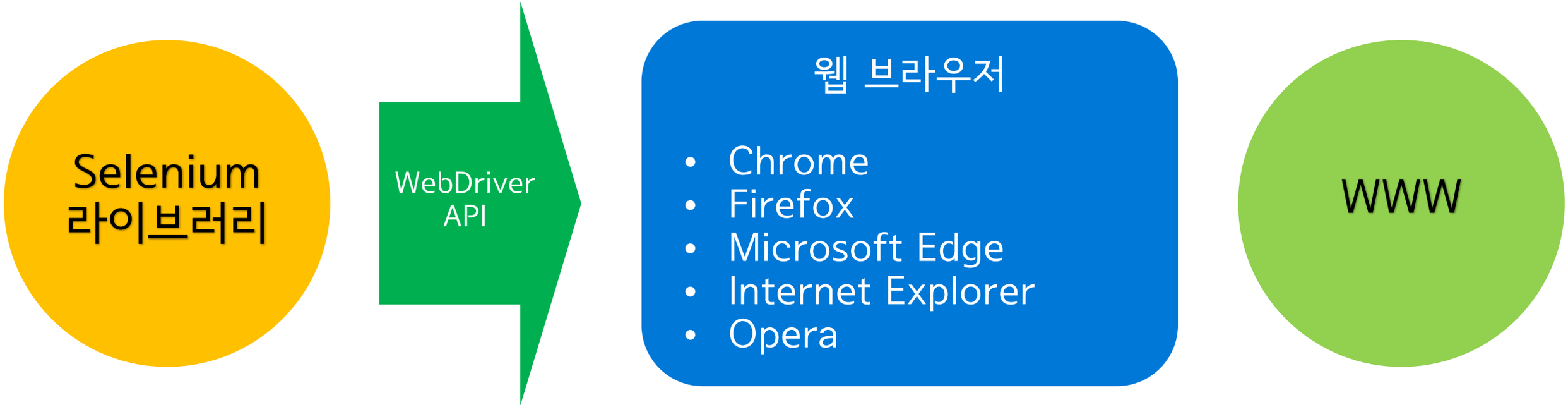
Firefox, Internet Explorer, Safari, Opera, Chrome, etc.

OS : Windows, Mac OSX, Linux, etc.

Language : Python, Java, JavaScript, R, Ruby, etc.

https://docs.seleniumhq.org/about/platforms.jsp

Selenium은 웹브라우저가 아닙니다.



Selenium의 WebDriver API

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

다양한 브라우저를 지원하는 API

즉, 같은 API로 다양한 브라우저로 접근 가능

하지만, Driver에 따라 다른 동작을 보일 수도 있습니다.

Driver 종류

Ie : Internet Explorer

Remote : 다른 머신에 Selenium Standalone Server를 설치하여 접속

Third Parties

Firefox : 버전 56(2017년)부터 Headless 모드 지원

Chrome : 버전 59(2017년)부터 Headless 모드 지원

Chrome 버전 59 이상을 써주세요. chrome://version 페이지에서 확인 가능

Opera

PhantomJS : nodejs로 구현되고 Headless 모드 구동이 장점 ➔ 개발 중단

Chrome 브라우저에서 2017년에 Headless 기능을 지원함에 따라, 의미가 없어짐.

WebDriver 기본 API

driver.get("...") : 지정 URL로 이동하기

driver.save_screenshot("저장경로.png") : 스크린샷 찍기

driver.title : 현재 페이지 제목

driver.page_source : 현재 HTML 내역

driver.current_url : 현재 URL

driver.close() : Focus된 윈도우만 닫기

driver.quit() : 현재 세션의 모든 윈도우 닫기

driver.find_element(s)_* : Element 찾기

Selenium을 활용한 네이버 검색 예시 (1)

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from bs4 import BeautifulSoup

# Selenium으로 브라우저 구동
driver = webdriver.Chrome('drivers/chromedriver_win32/chromedriver.exe')
try:
    driver.get('http://naver.com') # 지정 주소의 웹페이지 방문

    # 현재 페이지 내에서 name=query 요소 찾고, 검색어 입력 후에, form submit
    field = driver.find_element_by_name('query')
    field.send_keys('파이썬')
    field.submit()
    print('wait 5 seconds ...')

    # 다음 페이지에 계속 ...
```

Selenium을 활용한 네이버 검색 예시 (2)

```
    # 이전 페이지에 이어 ...

    # 검색결과를 보여주기 위해, 페이지 전환이 발생할 텐데
    # 찾고자 하는 요소가 렌더링이 될 때까지, 최대 5초 대기
    condition = EC.presence_of_all_elements_located((By.CLASS_NAME, 'sh_blog_title'))
    WebDriverWait(driver, 5).until(condition)
    print('loaded.')

    # 방법1) 지정 태그 요소 찾기
    # for tag in driver.find_elements_by_css_selector('.sh_blog_title'):
        # print(tag.text, tag.get_attribute('href'))

    # 방법2)
    html = driver.page_source
    soup = BeautifulSoup(html, 'html.parser')
    tag_list = soup.select('.sh_blog_title')
    for tag in tag_list:
        print(tag['href'])
finally:
    driver.quit()
```

흔한 Selenium 사용 패턴

1) 브라우저 구동

2) 원하는 페이지/화면 이동

방법1) 주소창에 주소를 직접 넣고, 페이지이동

방법2) 링크를 클릭해서, 페이지 이동

방법3) 화면 스크롤을 하거나, JavaScript 코드를 넘겨 실행

3) 페이지 현재 HTML 현황을 읽거나, 특정 태그를 선택하여 읽기

3) Form Fields에 값 채워넣고, Submit

ex) 검색폼, 로그인폼, 글쓰기폼 등

각 Field를 지정하여 값을 채워넣고, Submit

4) 페이지 전환이 완료되기를 대기하거나, 원하는 컨텐츠가 렌더링될 때까지 대기

5) 스크린샷 찍기

설치 (1)

python3

공식 배포판 : https://www.python.org

Anaconda 배포판 : https://www.anaconda.com/distribution/ (추천)

주의) 설치 시에 환경변수 PATH에 꼭 체크해주세요.

라이브러리

pip3 install selenium beautifulsoup4 requests jupyter

설치 (2)

사용할 Driver 다운로드 (원하는 경로의 drivers 폴더에 압축풀어서 모아두기)

Internet Explorer Driver Server : 32비트용 권장 (64비트 머신이라도 32비트 권장)

윈도우만 지원

Chrome Driver

Chrome 설치 필요 : https://www.google.com/chrome/

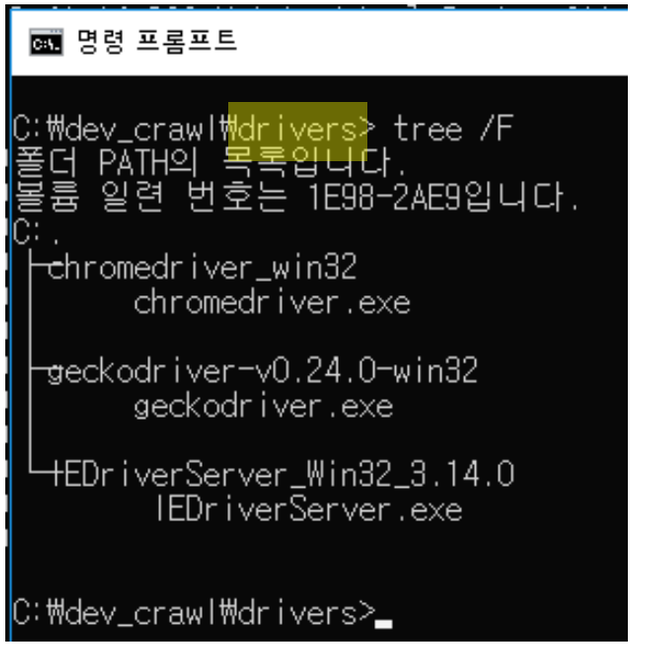
Firefox Gecko Driver

Firefox 설치 필요 : https://www.mozilla.org/firefox/

Edge

윈도우만 지원

정기적으로 Driver 버전을 체크하여, 최신버전으로 업데이트해주세요.
(요즘 브라우저는 자동 업데이트)



샘플 코드

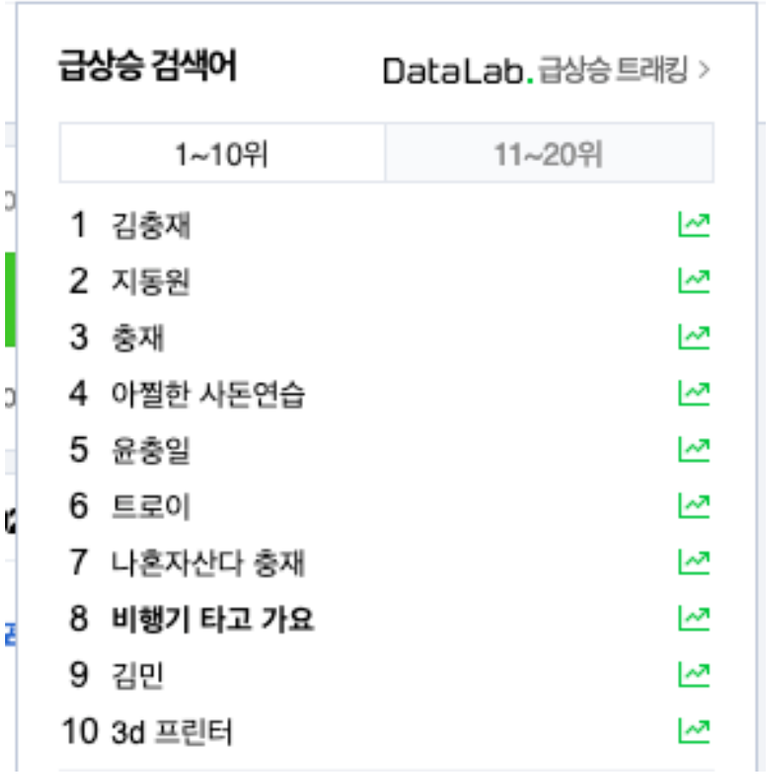
1) 브라우저 구동
2) 웹페이지 방문
3) 전체 HTML을 읽고 파싱하여 원하는 데이터 획득
4) 브라우저 종료

requests를 활용한 HTTP 요청

```
import requests
from bs4 import BeautifulSoup

res = requests.get('https://www.naver.com')
html = res.text # 초기 응답 HTML 내역

soup = BeautifulSoup(html, 'html.parser')
tag_list = soup.select('.PM_CL_realtimeKeyword_rolling_base .ah_k')
keyword_list = [tag.text for tag in tag_list]
print(len(keyword_list))
print(keyword_list)
```



JavaScript 위주의 사이트에 대해서는 크롤링이 불가할 확률이 높음.
하지만 빠르고 효율적인 처리.

Firefox 브라우저를 활용한 HTTP 요청

```
from selenium import webdriver
from bs4 import BeautifulSoup

executable_path = 'drivers/geckodriver.exe'

with webdriver.Firefox(executable_path=executable_path) as driver:  # 브라우저 구동
    driver.get("https://www.naver.com")  # 주소를 직접 지정하여, 웹페이지 방문
    html = driver.page_source            # 웹페이지 현재의 전체 HTML 읽기

    soup = BeautifulSoup(html, 'html.parser')
    tag_list = soup.select('.PM_CL_realtimeKeyword_rolling_base .ah_k')
    keyword_list = [tag.text for tag in tag_list]
    print(len(keyword_list))
    print(keyword_list)
```

각 머신 상의 경로에 맞게 수정.
혹은 환경변수 PATH상의 디렉토리에
geckodriver.exe 를 복사하고 생략 가능

WebDriver의 __exit__ 에서 driver.quit() 가 호출됩니다.
https://github.com/SeleniumHQ/selenium/blob/selenium-3.14.0/py/selenium/webdriver/remote/webdriver.py#L168

JavaScript 위주의 사이트도 가능.
하지만 리소스를 많이 먹고,
requests에 비해서는 느린 처리
단, IE 위주의 사이트는 IE를 써야함.

Chrome 브라우저를 활용한 HTTP 요청

```
from selenium import webdriver
from bs4 import BeautifulSoup

executable_path = 'drivers/chromedriver.exe'

with webdriver.Chrome(executable_path=executable_path) as driver:  # 브라우저 구동
    driver.get("https://www.naver.com")  # 주소를 직접 지정하여, 웹페이지 방문
    html = driver.page_source            # 웹페이지 현재의 전체 HTML 읽기

    soup = BeautifulSoup(html, 'html.parser')
    tag_list = soup.select('.PM_CL_realtimeKeyword_rolling_base .ah_k')
    keyword_list = [tag.text for tag in tag_list]
    print(len(keyword_list))
    print(keyword_list)
```

각 머신 상의 경로에 맞게 수정.
혹은 환경변수 PATH상의 디렉토리에
chromedriver.exe 를 복사하고 생략 가능

각 머신 상의 경로에 맞게 수정.
혹은 환경변수 PATH상의 디렉토리에
geckodriver.exe 를 복사하고 생략 가능

JavaScript 위주의 사이트도 가능.
하지만 리소스를 많이 먹고,
requests에 비해서는 느린 처리
단, IE 위주의 사이트는 IE를 써야함.

요약

selenium을 통해, 다양한 브라우저에 대한 자동화

실제 브라우저를 사용하기에 JavaScript 지원 가능

- 사이트에 따라, 다른 브라우저를 써야할 수도.
- JavaScript 동작이 필요하지 않다면, requests를 사용하는 것이 빠르고 효율적인 처리.

하지만, 실제로 처리하는 요청이 많지 않고, 고민/검토할 시간이 부족할 때에는

→ Selenium을 통한 처리를 먼저 시도해볼 수 있겠습니다. ☺

인생은 짧습니다.
파이썬/장고를 쓰세요.

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

- Ask Company