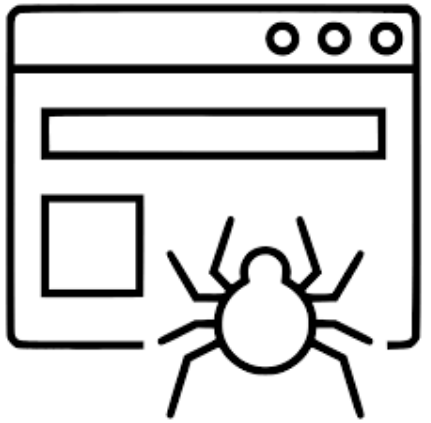
Web Crawling

2022.08

1. Web Crawling

■ 개요

- 사전적으로는 기어다닌다는 뜻
- Web상을 돌아다니면서 정보를 수집하는 행위
- 크롤링, 스크래핑(Scraping) 또는 데이터 긁기 등 다양한 단어로 불리움



■ 대상

- 웹 상의 자원
- 정적인 문서 또는 API와 같은 서비스
- 구글과 같은 검색엔진에서는 웹 사이트의 정적인 데이터를 긁어다가 필요한 정보를 추출해서 검색 인덱스를 생성
- 가격 정보 비교 사이트는 상품과 가격정보등을 긁어다가 서로 다른 쇼핑몰의 가격을 비교해줌

1. Web Crawling

■ 툴, 라이브러리

- 라이브러리
  - Beautiful Soup: 일반적, 파이썬
  - Jsoup: 자바 버전
  - Selenium: 브라우저를 이용
- 사전 지식: HTML, CSS, JavaScript 기초
- Chrome Web Browser 개발자 도구
- 설치: conda install beautifulsoup4

■ 과정

- 크롤링 대상 선정 (API 또는 웹 문서)
- 데이터 로드
- 데이터 분석
- 수집

- 예제 소스코드

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Web Crawling Example</title>
</head>

<body>
  <div>
    <p>a</p>
    <p>b</p>
    <p>c</p>
  </div>
  <div class=“ex_class">
    <p>1</p>
    <p>2</p>
    <p>3</p>
  </div>
  <div id=“ex_id">
    <p>X</p>
    <p>Y</p>
    <p>Z</p>
  </div>

  <h1>This is a heading.</h1>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
  <a href=“www.naver.com" class=“a”>
     Naver</a>
</body>

</html>
```

2. Beautiful Soup

- HTML 파일 열기

```
from bs4 import BeautifulSoup
with open("00_Example.html") as fp:
    soup = BeautifulSoup(fp, 'html.parser')
```

- urllib를 통해서 웹에 있는 소스 가져오기

```
import urllib.request
import urllib.parse

# web_url에 원하는 웹의 URL을 넣어주면 됨
with urllib.request.urlopen(web_url) as response:
    html = response.read()
    soup = BeautifulSoup(html, 'html.parser')

# 단순히 아래와 같이 해도 됨
from urllib.request import urlopen
html = urlopen(web_url)
soup = BeautifulSoup(html, 'html.parser')
```

- requests 이용

```
import requests
url = 'https://www.melon.com/chart/week/index.htm'
header = {'User-Agent':
          'Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko'}
req = requests.get(url, headers=header)
html = req.text



# 개발자 콘솔에서 navigator.userAgent로 확인
"Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/91.0.4472.77 Mobile Safari/537.36"
```

2. Beautiful Soup

- 태그를 이용해서 가져오기
- find - 해당 조건에 맞는 하나의 태그를 가져온다. 중복이면 가장 첫 번째 태그를 가져온다.

```
from bs4 import BeautifulSoup
fp = open("00_Example.html")
soup = BeautifulSoup(fp, 'html.parser')

first_div = soup.find("div")
print(first_div)

# 출력 결과
<div>
<p>a</p>
<p>b</p>
<p>c</p>
</div>
```

- find_all - 해당 조건에 맞는 모든 태그들을 가져온다.

```
all_divs = soup.find_all("div")
print(all_divs)

# 출력 결과
[<div>
<p>a</p>
<p>b</p>
<p>c</p>
</div>, <div class="ex_class">
<p>1</p>
<p>2</p>
<p>3</p>
</div>, <div id="ex_id">
<p>X</p>
<p>Y</p>
<p>Z</p>
</div>]
```

- find_all - 해당 조건에 맞는 모든 태그들을 가져온다.

```
all_ps = soup.find_all("p")
print(all_ps)

# 출력 결과
[<p>a</p>, <p>b</p>, <p>c</p>, <p>1</p>, <p>2</p>, <p>3</p>, <p>X</p>, <p>Y</p>,
<p>Z</p>, <p>This is a paragraph.</p>, <p>This is another paragraph.</p>]
```

- 태그와 속성을 이용해서 가져오기

```
# 함수의 인자로 원하는 태그를 첫번째 인자로
# 그 다음에 속성:값의 형태(dictionary)로 만들어서 넣어주면 됨
# - find_all('태그명', {'속성명' : '값' ...})
# - find('태그명', {'속성명' : '값' ...})

ex_id_divs = soup.find('div', {'id' : 'ex_id'})
print(ex_id_divs)

# 출력 결과
<div id="ex_id">
<p>X</p>
<p>Y</p>
<p>Z</p>
</div>

# 동일한 결과
딕셔너리 대체
    result = soup.find('div', id = 'ex_id')
```

- class 이름으로 찾기

```
result = soup.find('div', {'class' : 'ex_class'})
print(result)

# 출력 결과
<div class="ex_class">
<p>1</p>
<p>2</p>
<p>3</p>
</div>


# 동일한 결과를 얻는 방법
1. class 생략
        result = soup.find('div', 'ex_class’)

2. class_ (under bar가 있음) 로 검색
        result = soup.find(class_ = 'ex_class')
```

- CSS Selector로 찾기

1. id로 찾기

```
result = soup.select_one('#ex_id')
result_list = soup.select('#ex_id')
```

2. class로 찾기

```
result = soup.select_one('.ex_class')
result_list = soup.select('.ex_class')
```

- 결과 가져오기

```
ex_id_divs = soup.find("div", {"id":"ex_id"})
# 찾은 태그들 안에서 p 태그를 가져온다.
all_ps_in_ex_id_divs = ex_id_divs.find_all("p")
print(all_ps_in_ex_id_divs)

# 출력 결과
[<p>X</p>, <p>Y</p>, <p>Z</p>]


all_ps_in_ex_id_divs[0].get_text()
X

all_ps_in_ex_id_divs[1].string
Y
```

- attribute 결과 가져오기

```
<a href="www.naver.com" class="a">Naver</a>

a_tag = soup.find('a')

a_tag.string
Naver

a_tag.attrs['href']
www.naver.com

a_tag[‘href’]
www.naver.com
```

■ 설치

- conda 설치 : conda install selenium
- webdriver 설치(본인 웹 브라우저 버전에 맞춰 설치)
  - Chrome :

    `https://sites.google.com/a/chromium.org/chromedriver/downloads`
  - Safari :

    `https://webkit.org/blog/6900/webdriver-support-in-safari-10`

■ 특징

- 브라우저를 통해서 할 수 있는 모든 일을 자동화하여 처리할 수 있음
- 입력 가능(마우스 이동/클릭, 키보드 입력 등)
- 브라우저를 띄우지 않고도 실행 가능
- 웹 로딩시 시간이 걸리므로 프로그래밍시 주의가 필요
- 끝내기 전에 브라우저를 닫아야 함(driver.close())

- 구글 검색결과 샘플 코드

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

driver = webdriver.Chrome('./chromedriver')
# driver.maximize_window()
driver.get('http://www.google.com/')
time.sleep(1)

search_box = driver.find_element_by_name('q')   # class name이 q인 곳을 찾아
search_box.send_keys('ChromeDriver')            # 키워드를 입력하고
search_box.send_keys(Keys.RETURN)               # submit()
time.sleep(2)                                   # 2초간 기다림

divs = driver.find_elements_by_css_selector('.g')
title_list, content_list = [], []
for div in divs:
    title = div.find_element_by_css_selector('.LC20lb.DKV0Md').text
    content = div.find_element_by_css_selector('.VwiC3b.yXK7lf.MUxGbd.yDYNvb.
lyLwlc').text
    print(title)
    print(content)
```

- 구글 검색결과 샘플 코드 – BeautifulSoup version

```
import pandas as pd
from bs4 import BeautifulSoup

html = driver.page_source
soup = BeautifulSoup(html, 'html.parser')
divs = soup.select('.g')
title_list, content_list = [], []
for div in divs:
    title = div.select_one('.LC20lb.DKV0Md').get_text()
    content = div.select_one('.VwiC3b.yXK7lf.MUxGbd.yDYNvb.lyLwlc').get_text()
    title_list.append(title)
    content_list.append(content)

df = pd.DataFrame({
    'title': title_list, 'content': content_list
})
df.to_csv('google.tsv', sep='\t')

driver.close()
```

■ 요소 찾는 방법

- find_element_by_OOO – 하나의 요소를 찾음
- find_elements_by_OOO – 복수개의 요소를 찾음
- OOO 에 들어갈 수 있는 방법
  - name
  - tag_name
  - id
  - class_name
  - css_selector
  - link_text
  - partial_link_text
  - xpath

```
driver.find
    find_element(self, by, value)
    find_element_by_class_name(self, name)
    find_element_by_css_selector(self, css_selector)
    find_element_by_id(self, id_)
    find_element_by_link_text(self, link_text)
    find_element_by_name(self, name)
    find_element_by_partial_link_text(self, link_text)
    find_element_by_tag_name(self, name)
    find_element_by_xpath(self, xpath)
    find_elements(self, by, value)
    find_elements_by_class_name(self, name)
    find_elements_by_css_selector(self, css_selector)
    find_elements_by_id(self, id_)
    find_elements_by_link_text(self, text)
    find_elements_by_name(self, name)
    find_elements_by_partial_link_text(self, link_text)
    find_elements_by_tag_name(self, name)
    find_elements_by_xpath(self, xpath)
```

■ 필요한 데이터 가져오기

- 속성 가져오기
  - `get_attribute('attribute명')`

- 텍스트 가져오기
  - `text`