requests 와 urllib. request의 차이점 알아보기

- **web scrapping**
  - **import requests**
    - requests.post(url)
    - response = requests.get(url)
      - url 반응 확인
      - .text
        - soup으로 불러들일 수 있음
    - url = 'link address'
      - html = requests.get(url).text
  - **import urllib**
    - 필요 모듈
      - req = Request(html.format(search_words = **urllib.parse.quote('어영의_논동자')**))
      - html = 'https://ko.wikipedia. org/wiki/**{search_words}**'
    - 한글주소를 UTF-8로 변환
    - from urllib.request import urlopen
      - url = 'link address'
      - page = urlopen(url).read()
        - **page.status**
          - http 상태코드
            - 100 : 조건부 응답
            - 200 : 성공
            - 300 : 리다이렉션 성공
            - 400 : 요청오류
            - 500 : 서버오류
      - soup = BeautifulSoup(page, ' html.parser')
        - 'lxml'
        - print(soup.prettify())
    - 로봇인식 감지
      - from urllib.request import Request
      - req = Request(url, **headers={' User-Agent' : 'Chrome'}**)
    - from urllib.parse import urljoin
      - 상대주소 / 절대주소 구분
      - **urljoin(url_base, item.find('a')['href'])**
        - 절대주소라면 url_base를 붙이지 않고, 상대주소라면 url_base를 붙여서 입력
  - **from bs4 import BeautifulSoup**
    - pip install bs4
    - soup = BeautifulSoup(**html, 'html.parser'**)
      - print(soup.**prettify()**)
      - tags = soup.select('css')
        - 다중 요소 선택
        - tag = tags[0]
          - print(tag.text)
        - tags = soup. select_one('#id > tag.class_name')
          - 단일 요소만 선택할 때
        - css = copy > copy selector 의 내용을 하나씩 확인해봐야함
        - <tag> : tag
        - # : ID
        - .class : class
          - class='c_name1 c_name2' 띄워쓰여진 클래스 이름은
            - 'c_name1.c_name2' 으로 찾으면 된다.
        - > : 바로 밑 하위를 검색
          - #article > div:nth-child(2) > div > div.btns_submit.search
            - article 이라는 **아이디** 안에
              - div라는 **태그 2번째 속성** 안에
              - div **태그** 안에
                - btns_submit.search 라는 **클래스 이름**을 가진 div태그
    - html = **open**('.. /data/data.html', **'r'**). read()
  - **URL 특수문자 의미**
  - **tqdm**
    - **크롤링할 때 진행도 상황을 보기 위해 쓰는 툴**
    - **conda install -c conda-forge tqdm**
      - from tqdm import tqdm
    - **pip install tqdm**
  - **import selenium + webdriver(chrome)**
    - from selenium import webdriver
      - driver = webdriver.Chrome('chromedriver')
        - error
          - mac 보안 설정에서 해당 파일이 제한되어있는지 확인할 것
        - driver.get('address')
          - driver.find_element(By.x, 'search thing')
            - By.XPATH
            - By.ID
            - By.LINK_TEXT
            - By.PARTIAL_LINK_TEXT
            - By.NAME
            - By.TAG_NAME
            - By.CLASS_NAME
            - By.CSS_SELECTOR
            - 동작
              - .click()
                - 해당 element 클릭
              - .send_keys('value', Keys.ENTER)
                - Value 입력
                - Keys.ENTER 엔터키 입력
            - **.get_attribute('value')**
              - **WebElement의 특정 값을 찾을 때**
    - from selenium.webdriver.common.keys import Keys
    - from selenium.webdriver.common.by import By
      - a1 = driver.find_element(By. XPATH, '~~')
        - **드랍다운 리스트 구하기**
          - from selenium.webdriver.common.by import By
      - a2 = a1.find_elements(By. TAG_NAME, '~~')
      - a3 = a2.get_attribute('~~')
    - **method**
      - **driver = webdriver.Chrome('chromedriver')**
      - 페이지 스크롤
        - driver.execute_script('**return document.body.scrollHeight**')
          - 스크롤 가능한 높이를 가져오기
        - driver.execute_script('**window.scrollTo(0, document.body.scrollHeight)**;')
          - 스크롤을 0 에서 끝단까지 스크롤
        - driver.execute_script('return ***arguments[0].scrollIntoView()***;', driver.find_element(By.XPATH, xpath)
          - driver.find_element(By.XPATH, xpath) 위치까지 스크롤을 이동
      - **ActionChains(driver)**
        - ActionChains(driver).**move_to_element(some_tag)**
          - 특정 태그(some_tag)의 위치로 이동을 준비
        - ActionChains(driver).**perform()**
          - 행동을 실행
        - 사용자 행동을 추가
  - **time wait**
    - from time import sleep
      - sleep(3)
        - 시스템 시간
    - from selenium import webdriver
      - driver = webdriver.Chrome(chromedriver)
        - driver.implicitly_wait(10)
          - 브라우저 자체 시간 지정
    - from selenium import webdriver
      from selenium.webdriver.support.ui import WebDriverWait
      from selenium.webdriver.support import expected_conditions as EC
      - browser = webdriver.Chrome('chromedriver')
        browser.get('https://www.naver.com/')
        - WebDriverWait(driver, 100).until(EC. presence_of_element_located((By. CSS_SELECTOR,'div.header')))
          - 조건이 True가 될 때까지 지정한 시간만큼 대기
  - **드랍다운 선택**
    - from selenium.webdriver. support.select import Select
      - select = Select(driver. find_element_by_id('#_id'))
        - select.select_by_visible_text('TEXT')
          # select visible text
        - select.select_by_value('value_number')
          # select option value
        - select.select_by_index(1)
          #select index value
      - select = Select(driver. find_element(By.x, 'search thing'))
    - 드랍다운 선택
      - from selenium.webdriver.support.ui import Select
        - select = Select(driver.find_element_by_css_selector('#sch_bub_nm'))
          - select.select_by_value('value_number')
            # select option value
          - select.select_by_index(1)
            # Choose the position you want
          - select.select_by_visible_text('TEXT')
            # select visible text
        - select = Select(driver. find_element(By.x, 'search thing'))
    - **🐶드롭다운 선택시**
      - **drive.find_element().send_keys('something')**
        - 를 활용하면 대체로 가능해 보이나, 해당 element의 값이 string 형식으로, 원하는 value를 보내면, 제대로 선택이 되지 않는 경우가 발생함을 확인
        - 이에 정확한 value를 지정하기 위해 Select 를 사용
          - select.select_by_value('something')
            - 원활하게 작동하는 것을 확인하였다.
            - 이후 조회버튼 클릭 등을 이용하기 위해
              - driver.find_element().send_keys(Keys.ENTER) 이용함

Beautiful Soup 과 연동

- **from bs4 import BeautifulSoup**
  - req = **driver.page_source**
    - driver는 현재 진행중인 드라이버의 위치를 기억하고 있음
  - soup = BeautifulSoup(**req**, ' html.parser')
    - titles = soup.find_all(class_='title')
      - for i in titles:
        print(i.get_text())

soup method

- soup.<tag_name>
  - 가장 첫 태그 내용을 반환
- soup.find('tag', 'class_name')
  - 가장 첫 p 태그 내용을 반환
  - soup.find('p', class_='inner-text')
    - p 태그 안에 inner-text 라는 클래스의 내용을 반환
  - soup.find('p', {'class' : ' inner-text', 'id' : 'first'})
- soup.find_all('p')
  - 모든 p 태그 내용을 반환
- soup.find_all(class_='outer-text')
  - 특정 클래스 내용을 반환
- soup.find_all('tag', ' class_name')[n].string
  - 특정 태그 > 특정 클래스의 string 가져오기
- soup.find(id='first')
  - ID 태그의 내용을 반환

tag method

- find 명령어를 사용할 수 있다.
- type(bs4.element.Tag)
- 내용 확인
  - tag.get_text()
    - tag의 text를 반환
  - tag.text
    - tag의 text를 반환
      - 가끔 공백이 들어가는 경우가 있는데, 이 경우 strip() 해주면 된다.
  - tag.string
    - tag의 text를 반환
      - 자식 태그의 text를 가져오지 못하는 경우가 발생
      - string : 문장 한개
        strings : 문장 여러개 묶음
  - tag['element']
    - tag의 element를 가져옴
  - tag.get('href')
    - tag의 href 속성 의 내용을 가져옴