# Scrapy

스크래피는 웹사이트에서 데이터 추출을 위한 오픈소스 프레임워크 입니다.

- https://scrapy.org/

**index**



## 1. install scrapy

**windows**

$ conda install -c conda-forge scrapy

**mac**

$ pip3 install scrapy

## 2. xpath

css selector가 아닌 xpath를 이용하여 웹 페이지의 html 엘리먼트를 선택한 객체를 생성할수 있습니다. shell 환경과 jupyter notebook 환경에서 xpath를 연습 하겠습니다.

**2.1 xpath 기본 문법**

// : 가장 상위 Element

. : 현재 Element

* : 조건에 맞는 앞부분의 하위 Element를 모두 살펴봄 ( css selector에서 한칸 띄기와 같음 )

/ : 바로 아래 요소 ( css selector에서 > 와 같음 )

element[조건]

- p[2] : p element의 두번째 element : index가 0이 아닌 1부터 시작
- [@(attribute key)="(attribute value)"]
- [@id="email"] : 아이디 값이 email인 element
- [@class="pw"] : 클래스 값이 pw인 element

not

- not(조건) : 조건이 아닌 요소를 찾음

**2.2 scrapy shell**

2.2.1 네이버 실시간 검색어

---

# 스크래피 쉘을 이용하여 네이버 페이지 연결하기

$ scrapy shell "http://naver.com"

# 네이버 실시간 검색 1위 xpath 객체 가져오기

In [1] : response.xpath('//*[@id="PM_ID_ct"]/div[1]/div[2]/div[2]/div[1]/div/ul/li[1]/a/span')

# 네이버 실시간 검색 20개 xpath 객체 가져오기

In [2] : response.xpath('//*[@id="PM_ID_ct"]/div[1]/div[2]/div[2]/div[1]/div/ul/**li**/a/span')

# xpath의 text() 를 이용하여 xpath 객체의 text 데이터 추출하기

In [3] : response.xpath('//*[@id="PM_ID_ct"]/div[1]/div[2]/div[2]/div[1]/div/ul/li/a/span/**text()**')

# .extract() 함수를 이용하여 xpath에서 추출된 text 데이터 리스트로 추출하기

In [4] : response.xpath('//*[@id="PM_ID_ct"]/div[1]/div[2]/div[2]/div[1]/div/ul/li/a/span/text()')**.extract()**

# 검색어 데이터만 가져오기

In [5] : response.xpath('//*[@id="PM_ID_ct"]/div[1]/div[2]/div[2]/div[1]/div/ul/li/a/span[2]/text()').extract()

# xpath의 .을 이용해서 가져오기

In [6] : keywords = response.xpath('//*[@id="PM_ID_ct"]/div[1]/div[2]/div[2]/div[1]/div/ul/li')

In [7] : for keyword in keywords:

... :  print(keyword.xpath('./a/span[2]/text()')[0].extract())

---

2.2.2 다음 실시간 검색어

---

# 스크래피 쉘을 이용하여 다음 페이지 연결하기

$ scrapy shell "http://daum.net"

# 다음의 실시간 검색어 데이터 가져오기

In [1] : response.xpath('//*[@id="mArticle"]/div[2]/div[2]/div[2]/div[1]/ol/li/div/div[1]/span/a/text()').extract()

---

2.2.3 gmarket 베스트 아이템

---

# 스크래피 쉘을 이용하여 gmarket 베스트 페이지 연결하기

$ scrapy shell "http://corners.gmarket.co.kr/Bestsellers"

# 베스트 200 아이템 제목 데이터 가져오기

In [1] : response.xpath('//*[@id="gBestWrap"]/div/div[3]/div[2]/ul/li/a/text()').extract()

# li 엘리먼트에서 class가 first가 데이터만 가져오기

In [2] : response.xpath('//*[@id="gBestWrap"]/div/div[3]/div[2]/ul/li**[@class="first"]**/a/text()').extract()

# li 엘리먼트에서 class가 first가 아닌 데이터만 가져오기

In [3] : response.xpath('//*[@id="gBestWrap"]/div/div[3]/div[2]/ul/li**[not(@class="first")]**/a/text()').extract()

# gmartket 베스트 아이템 링크 가져오기

In [4] : response.xpath('//*[@id="gBestWrap"]/div/div[3]/div[2]/ul/li/a/**@href**').extract()

---

**2.3 jupyter notebook**

---

import requests

from scrapy.http import TextResponse

```
# 웹 페이지에 연결

req = requests.get('https://www.naver.com/')


# response 객체 생성

response = TextResponse(req.url, body=req.text, encoding='utf-8')


# 네이버 실시간 검색어

response.xpath('//*[@id="PM_ID_ct"]/div[1]/div[2]/div[2]/div[1]/div/ul/li/a/span[2]/
text()').extract()


# 다음 실시간 검색어

req = requests.get('https://daum.net/')

response = TextResponse(req.url, body=req.text, encoding='utf-8')

response.xpath('//*[@id="mArticle"]/div[2]/div[2]/div[2]/div[1]/ol/li/div/div[1]/span/a/
text()').extract()
```

---

## 3. scrapy project

### 3.1 scrapy 프로젝트 생성

```
$ scrapy startproject crawler
```

아래 트리 구조의 디렉토리와 파일들이 생성 됩니다.

```
.
├── gb_crawler
|   ├── __init__.py
```

```
|   ├── items.py
|   ├── middlewares.py
|   ├── pipelines.py
|   ├── settings.py
|   └── spiders
|       ├── __init__.py
└── scrapy.cfg
```

### 3.2 scrapy 프로젝트 파일 설명

spiders directory

- 어떤 웹사이트들을 어떻게 크롤링할 것인지 명시하고, 각각의 웹 페이지의 어떤 부분을 스크래핑할 것인지 명시하는 클래스가 모여있는 디렉토리

items.py

- 웹페이지에서 원하는 부분을 스크랩하여 저장할 때 사용하는 사용자 정의 자료구조 클래스

pipeline.py

- 스크래핑 결과물을 Item 형태로 구성하였을 때, 이를 자유롭게 가공하거나 다양한 파일 형태로 저장할 수 있도록 하는 클래스

settings.py

- Spider나 Item Pipeline 등이 어떻게 동작하도록 할 지에 대한 세부적인 설정 사항을 기재하는 파일 크롤링의 빈도등을 설정

- cf. robots.txt (settings.py - ROBOTSTXT_OBEY = True)

### 3.3 크롤링 코드 작성

네이버 영화의 현재 상영작의 관객수 크롤링

### 3.3.1 item.py

---

```
import scrapy


class CrawlerItem(scrapy.Item):
    title = scrapy.Field()
    count = scrapy.Field()
```

---

### 3.3.2 spiders/gb.py

link 리스트를 크롤링하고 link의 상세 페이지에 들어가서 영화 제목과 관객수 데이터를 가져옵니다.

yield - return과 비슷, 함수가 제너레이터를 반환

---

```
import scrapy
from crawler.items import CrawlerItem


# start_urls -> parse -> parse_page_contents 순으로 호출
class MovieSpider(scrapy.Spider):
    name = "NaverMovie"
    allow_domain = ["https://movie.naver.com"]
    start_urls = [
        "https://movie.naver.com/movie/running/current.nhn"
    ]


    # link 리스트를 가져옴
    def parse(self, response):
        links = response.xpath('//*[@id="content"]/div[1]/div[1]/div[3]/ul/li/dl/dt/a/@href')[:10].extract()
        for link in links:
```

```
            link = response.urljoin(link)
            yield scrapy.Request(link, callback=self.parse_page_contents)

    # 각페이지의 link로 접속하여 데이터를 가져옴
    def parse_page_contents(self, response):
        item = CrawlerItem()
        item["title"] = response.xpath('//*[@id="content"]/div[1]/div[2]/div[1]/h3/a[1]/text()').extract()[0]
        try:
            item["count"] = response.xpath('//*[@id="content"]/div[1]/div[2]/div[1]/dl/dd[5]/div/p[2]/text()').extract()[0]
        except:
            item["count"] = "0명"
        yield item
```

---

### 3.3.3 settings.py

---

ROBOTSTXT_OBEY = False

---

### 3.3.4 실행

$ scrapy crawl NaverMovie

# csv 파일로 저장

$ scrapy crawl NaverMovie -o movie.csv

### 3.3.5 pipeline 사용

컬럼의 순서를 지정할수 있습니다. 크롤러를 실행할때 CrawlerPipeline 객체를 생성하고 아이템을 하나씩 크롤링할때마다 process_item 함수를 실행합니다.

pipelines.py

---

```
import csv

class CrawlerPipeline(object):

    def __init__(self):
        self.csvwriter = csv.writer(open("NaverMovie.csv", "w"))
        self.csvwriter.writerow(["title","count"])

    def process_item(self, item, spider):
        row = []
        row.append(item["title"])
        row.append(item["count"])
        self.csvwriter.writerow(row)
        return item
```

---

settings.py

---

```
ITEM_PIPELINES = {
   'crawler.pipelines.CrawlerPipeline': 300,
}
```

---