고급? 하지만 간단할 수 있는 크롤링 기술: Scrapy

Scrapy

> 처음에는 난해하지만, 사용법을 알면 BeautifulSoup 라이브러리보다 편함
> 대표적인 크롤링 프레임워크로 상세 사용법은 책 한 권
> 하지만, 간단하게는 바로 사용할만 함

- 프레임워크
  - 함수와 코드를 미리 작성해놨다.
  - 특정 함수를 특정 위치에 어떻게 사용해야 하는지, 작성해야 하는지를 정해놓은 프로그램
  - python, C, C++, JAVA
  - learning curve

Scrapy

- 크롤링을 좀더 안정적으로 할 수 있음
  - Scrapy 내부에서 다양한 안정장치가 있음
- 크롤링을 좀더 빠르게 할 수 있음
  - 크롤링 프로그램을 동시에 여러개 실행시켜서 많은 량의 데이터 크롤링시, 시간 단축
- 다양한 크롤링 관련 기능
  - 크롤링한 데이터를 다양한 포멧으로 저장도 가능

사용 방법

1. 실제 크롤링할 스파이더(spider, scrapy 기반 크롤링 프로그램) 생성
2. 크롤링할 사이트(시작점)와 크롤링할 아이템(item)에 대한 selector 설정
3. 크롤러 실행

설치

- 윈도우/맥 공통

```
pip install scrapy
```

- 윈도우에서 정상 설치가 안될 시, 다음 항목 시도

```
pip install --upgrade setuptools
pip install pypiwin32
pip install twisted[tls]
```

- 그래도 안된다면,
    - https://visualstudio.microsoft.com/ko/downloads/

> 이럴때, 윈도우가 불편하다고 느끼지만, 맥이 그렇다고 꼭 좋은 것은 아닙니다.

크롤링 프로젝트 생성

1. 터미널 오픈

> 터미널이라는 용어가 낯설 수 있습니다. IT기술 중 하나입니다. 키보드로 명령을 내릴 수 있는 프로그램으로 이해합니다.

2. scrapy startproject <프로젝트이름> 으로 생성

```
scrapy startproject ecommerce
```

3. 생성된 프로젝트 확인

> 에디터로 해당 폴더(ecommerce) 를 오픈합니다. HTML 익히기에서 설치한 sublime text3를 사용하기로 합니다.

Scrapy로 작성된 프로젝트

> ecommerce 폴더 내의 파일 구조 확인

```
    scrapy.cfg              # deploy configuration file
    ecommerce/               # project's Python module, you'll import your code fr
        __init__.py
        items.py            # project items definition file
        pipelines.py        # project pipelines file
        settings.py         # project settings file
        spiders/            # a directory where you'll later put your spiders
            __init__.py
```

크롤러(spider) 작성

> 터미널에서 ecommerce/ecommerce 폴더에서 다음 명령으로 작성 가능

- 크롤러이름: 크롤링 프로젝트 내에, 여러 가지 크롤러(scrapy에서는 spider라고 함) 있을 수 있으므로, 각 크롤러의 이름을 지정
- 크롤링페이지주소: 각 크롤러가 크롤링을 시작할 페이지 주소를 지정

```
scrapy genspider <크롤러이름> <크롤링페이지주소>

scrapy genspider gmarket www.gmarket.co.kr
```

크롤러(spider) 작성

> ecommerce/ecommerce/spiders 디렉토리에 gmarket.py 파일(템플릿)이 생김
> 직접 scrapy genspider 명령을 사용하지 않고, 만들어도 됨

크롤러(spider) 실행

- 터미널 환경에서, ecommerce 디렉토리에서 scrapy crawl gmarket 명령

쉬어가기

```
scrapy startproject ecommerce
scrapy genspider gmarket www.gmarket.co.kr
scrapy crawl gmarket
```

크롤러(spider) 작성 - gmarket.py 로 보는 기본 템플릿 구조

> 파이썬 객체지향 프로그래밍을 이해하시면, 조금더 코드 이해 및 작성이 빠를 수 있음

- 클래스 이름은 마음대로 정하면 됨, 단 scrapy.Spider 를 상속받아야 함
- name이 크롤러(spider)의 이름
- allowed_domains는 옵션 (삭제해도 무방함)
    - 별도 상세 설정으로 허용된 주소 이외의 주소는 크롤링 못하게끔 하는 기능을 위한 변수

```
# -*- coding: utf-8 -*-
import scrapy


class GmarketSpider(scrapy.Spider):
    name = 'gmarket'
    allowed_domains = ['www.gmarket.co.kr']
    start_urls = ['http://www.gmarket.co.kr/']

    def parse(self, response):
        pass
```

크롤러(spider) 작성 - gmarket.py 로 보는 기본 템플릿 구조

- start_urls 가 중요함. 크롤링할 페이지 주소를 나타냄.
- parse 함수는 클래스의 메서드로 response를 반드시 인자로 받아야 함
    - response에 start_urls 에 기록된 주소의 크롤링 결과가 담아져오기 때문임

```
# -*- coding: utf-8 -*-
import scrapy


class GmarketSpider(scrapy.Spider):
    name = 'gmarket'
    allowed_domains = ['www.gmarket.co.kr']
    start_urls = ['http://www.gmarket.co.kr/']

    def parse(self, response):
        pass
```

크롤러(spider) 작성 - gmarket.py 로 보는 기본 템플릿 구조

- response 확인하기
    - response.text 에 크롤링된 데이터가 담겨져 있음

```
# -*- coding: utf-8 -*-
import scrapy


class GmarketSpider(scrapy.Spider):
    name = 'gmarket'
    allowed_domains = ['www.gmarket.co.kr']
    start_urls = ['http://www.gmarket.co.kr/']

    def parse(self, response):
        print(response.text)
```

크롤러(spider) 작성 - gmarket.py 로 보는 기본 템플릿 구조

- start_urls는 리스트로 크롤링할 주소를 여러개 써도 됨
- 동작 방식
    i. start_urls에서 주소를 하나씩 가져와서 크롤링한 후,
    ii. response 에 넣고, parse 함수를 호출함
    iii. parse 함수에 response에 담아져있는 크롤링 결과를 원하는 대로 처리하면 됨

```
# 예
start_urls = ['https://corners.gmarket.co.kr/Bestsellers/', 'https://sports.v.da
```

scrapy 연습 - response 사용법 이해

> scrapy shell 도 익힐겸, response 사용법도 쥬피터 노트북 스타일로 빠르게!
> gmarket 베스트 상품 페이지

```
scrapy shell 'http://corners.gmarket.co.kr/Bestsellers'
```

> exit 로 scrapy shell 을 종료할 수 있음

scrapy 연습 - response 사용법 이해

- view(response): 크롤링한 페이지를 웹브라우저를 통해 확인하기

```
view(response)
```

- response.url: 크롤링한 페이지 주소 확인

```
view(response)
```

scrapy 연습 - response 사용법 이해

- response.css(): css selector 로 데이터 가져오기

```
response.css('head > title').get()
response.css('head > title').getall()
response.css('head > title::text').get()
```

> gmarket 베스트 상품의 타이틀 가져오기

```
response.css('div.best-list li > a::text').getall()
```

> gmarket 베스트 상품의 특정 순번째의 타이틀 가져오기

```
response.css('div.best-list li > a::text')[1].get()
```

scrapy 연습 - response 사용법 이해

- response.xpath(): xpath 로 데이터 가져오기

```
response.xpath('//div[@class="best-list"]/ul/li/a').getall()

response.xpath('//div[@class="best-list"]/ul/li/a/text()').getall()
```

scrapy 연습 - response 사용법 이해

- re(): 정규표현식 쓰기

```
response.css('div.best-list li > a::text')[1].re('(\w+)')
```

```
response.xpath('//div[@class="best-list"]/ul/li/a/text()')[1].re('(\w+)')
```

크롤러(spider)로 작성해보기

- scrapy genspider 로 spider 작성하기

```
scrapy genspider gmarket http://corners.gmarket.co.kr/Bestsellers
```

- ecommerce/ecommerce/spiders/gmarket.py 파일

```
 -*- coding: utf-8 -*-
import scrapy


class GmarketSpider(scrapy.Spider):
    name = 'gmarket'
    allowed_domains = ['http://corners.gmarket.co.kr/Bestsellers']
    start_urls = ['http://corners.gmarket.co.kr/Bestsellers/']

    def parse(self, response):
        pass
```

크롤러(spider)로 작성해보기

- 크롤링 데이터 출력하기
  - 터미널 환경에서, ecommerce 디렉토리에서 scrapy crawl gmarket 명령

```
# -*- coding: utf-8 -*-
import scrapy


class GmarketSpider(scrapy.Spider):
    name = 'gmarket'
    allowed_domains = ['http://corners.gmarket.co.kr/Bestsellers']
    start_urls = ['http://corners.gmarket.co.kr/Bestsellers/']

    def parse(self, response):
        titles = response.css('div.best-list li > a::text').getall()
        for title in titles:
            print (title)
```

크롤러(spider)로 작성해보기

- 크롤링 데이터 다루기: 저장하기

> 이제부터 시작입니다.

- items.py 를 작성해야 합니다.

```
    scrapy.cfg              # deploy configuration file
    ecommerce/               # project's Python module, you'll import your code fr
        __init__.py
        items.py            # project items definition file
        pipelines.py        # project pipelines file
        settings.py         # project settings file
        spiders/            # a directory where you'll later put your spiders
            __init__.py
            gmarket.py
```

크롤러(spider)로 작성해보기

- items.py 작성
  - 크롤링하고자 하는 데이터를 아이템(item)으로 선언해줘야 함
  - 클래스를 만들고, scrapy.Item을 상속받고, 아이템의 이름을 만들고, scrapy.Field()를 넣어줘야 함

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# https://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class EcommerceItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    title = scrapy.Field()
```

크롤러(spider)로 작성해보기

- gmarket.py 수정
  - yield 명령어로 아이템(item)에 저장할 수 있다!
  - 아이템 클래스 생성 및 아이템 저장
    - 선언: from 프로젝트이름.items import 아이템클래스명
    - 클래스 생성: item = 아이템클래스명()
    - 아이템 저장
      - item['아이템명'] = 아이템데이터
      - yield item

크롤러(spider)로 작성해보기

- gmarket.py 수정

```
import scrapy
from ecommerce.items import EcommerceItem

class GmarketSpider(scrapy.Spider):
    name = 'gmarket'
    allowed_domains = ['http://corners.gmarket.co.kr/Bestsellers']
    start_urls = ['http://corners.gmarket.co.kr/Bestsellers/']

    def parse(self, response):
        titles = response.css('div.best-list li > a::text').getall()
        for title in titles:
            item = EcommerceItem()
            item['title'] = title
            yield item
```

크롤러(spider)로 작성해보기

- 아이템 데이터 처리하기

> 지금부터 scrapy의 강점이 나타난다.

- 다양한 데이터 포멧으로 아이템 저장하기
  - csv, xml, json 포멧
  - 터미널 환경에서, ecommerce 폴더에서 다음 명령

```
scrapy crawl 크롤러명 -o 저장할파일명 -t 저장포멧
# 예
scrapy crawl gmarket -o gmarket.csv -t csv
scrapy crawl gmarket -o gmarket.xml -t xml
```

> json 파일을 확인하면, 한글문자가 깨져나옴

```
scrapy crawl gmarket -o gmarket.json -t json
```

크롤러(spider)로 작성해보기

> 필요할 때마다, 설정을 해보면서, 설정에도 익숙해집니다.
> 문자의 유니코드 문제는 항상 존재! (utf-8 만 기억해놓으시면 됩니다.!)

- settings.py 수정

```
FEED_EXPORT_ENCODING = 'utf-8'
```

> 기존 json 파일을 지우고! 다시 다음 명령을 실행하면, 한글문자가 정상적으로 저장되는 것을 확인할 수 있음

```
scrapy crawl gmarket -o gmarket.json -t json
```

크롤러(spider)로 작성해보기

- 아이템 데이터 후처리하기
  - 일부 아이템은 저장하지 않거나,
  - 중복되는 아이템을 저장하지 않거나,
  - 데이터베이스등에 저장하거나,
  - 특별한 포멧으로 아이템을 저장하고 싶거나

> 프레임워크 스타일로 구조화시키기 위해 별도 파일에 작성할 수 있도록 하였음 (parse 함수에서 작성해도 됨)

- navernews/navernews/pipelines.py
  - 아이템이 저장되려 할 때마다, pipelines.py의 process_item 함수를 호출한다.

크롤러(spider)로 작성해보기

> 우선 gmarket.py 에서 상품과 가격을 모두 가져와보도록 코드를 수정합니다.

```
class GmarketSpider(scrapy.Spider):
    name = 'gmarket'
    allowed_domains = ['http://corners.gmarket.co.kr/Bestsellers']
    start_urls = ['http://corners.gmarket.co.kr/Bestsellers/']

    def parse(self, response):
        titles = response.css('div.best-list li[id] a::text').getall()
        prices = response.css('div.best-list li[id] div.s-price > strong > span >
        for i in range(len(titles)):
            item = NavernewsItem()
            item['title'] = titles[i]
            item['price'] = int(prices[i].replace("원", "").replace(",", ""))
            yield item
```

크롤러(spider)로 작성해보기

> pipelines.py 사용을 위해 별도 설정 필요

- settings.py 수정

```
ITEM_PIPELINES = {
    'navernews.pipelines.NavernewsPipeline': 300,
}
```

> 코드가 복잡해보여도 걱정할 필요는 없음, 프로젝트 생성시 작성이 되어 있으므로, 주석처리(#)만 삭제 하면 됨

크롤러(spider)로 작성해보기

> 각 아이템 생성시, pipeline.py 에 있는 process_item 함수를 거쳐가게 되어 있음
> 필요한 아이템만 return 해주고, 필터링할 아이템은 DropItem 을 통해, 더이상의 아이템처리를 멈추게 해줘야 함

```
from scrapy.exceptions import DropItem

class NavernewsPipeline(object):
    def process_item(self, item, spider):
        print (item)
        if item['price'] > 10000:
            return item
        else:
            raise DropItem("drop item having lower price than 10000")
```

여러 페이지 한번에 크롤링하는 spider 만들기

> 카테고리별 베스트상품중 1만원 이상의 상품 타이틀 가져오기

```
scrapy genspider gmarketcategory http://corners.gmarket.co.kr/Bestsellers\?viewTy
```

- gmarketcategory.py 파일

```
 -*- coding: utf-8 -*-
import scrapy


class GmarketcategorySpider(scrapy.Spider):
    name = 'gmarketcategory'
    allowed_domains = ['http://corners.gmarket.co.kr/Bestsellers?viewType=G&grou
    start_urls = ['http://http://corners.gmarket.co.kr/Bestsellers?viewType=G&gr

    def parse(self, response):
        pass
```

여러 페이지 한번에 크롤링하는 spider 만들기

> gmarketcategory.py 만 수정하면 됨

- start_urls 이외에도, start_requests 함수를 사용하는 방법이 있음
  - 각 페이지마다, parse 함수를 별도로 정의해서, 호출할 수도 있음

```
import scrapy
from navernews.items import NavernewsItem

class GmarketcategorySpider(scrapy.Spider):
    name = 'gmarketcategory'
    allowed_domains = ['http://corners.gmarket.co.kr/Bestsellers']

    # 1번만 호출
    def start_requests(self):
        for i in range(1, 13):
            yield scrapy.Request("http://corners.gmarket.co.kr/Bestsellers?viewTy
```

여러 페이지 한번에 크롤링하는 spider 만들기

> parse 함수는 기존과 동일하게 작성
>
> item, pipeline도 gmarket 과 동일하게 사용

```
    def parse(self, response):
        titles = response.css('div.best-list li[id] a::text').getall()
        prices = response.css('div.best-list li[id] div.s-price > strong > span >
        for i in range(len(titles)):
            item = NavernewsItem()
            item['title'] = titles[i]
            item['price'] = int(prices[i].replace("원", "").replace(",", ""))
            yield item
```

- scrapy 장점: 빠르다! 구조화되어 있어서 여러 데이터를 빠르게 정리할 수 있다.
- scrapy 단점: 프레임워크 사용법에 익숙해야 한다. (객체지향도 기본적인 이해 필요)

가장 빠른 scrapy 활용법

> 고급과정입니다. 현실적인 방법으로 강의에서 작성된 코드를 기반으로 필요한 부분만 수정하세요!

- 강의에서 작성된 gmarket, gmarketcategory 코드를 기반으로
    i. 크롤링하고자 하는 주소와 css selector 수정
    ii. 크롤링하고자 하는 데이터 구조 수정 (item)
    iii. 필터링하고자 하는 데이터 수정 (pipeline)