Урок 6. Scrapy.

Парсинг фото и файлов

Домашнее задание

1. Создайте новый проект Scrapy. Дайте ему подходящее имя и убедитесь, что ваше окружение правильно настроено для работы с проектом.
2. Создайте нового паука, способного перемещаться по сайту www.unsplash.com. Ваш паук должен уметь перемещаться по категориям фотографий и получать доступ к страницам отдельных фотографий.

3. Определите элемент (Item) в Scrapy, который будет представлять изображение. Ваш элемент должен включать такие детали, как URL изображения, название изображения и категорию, к которой оно принадлежит.
4. Используйте Scrapy ImagesPipeline для загрузки изображений. Обязательно установите параметр IMAGES_STORE в файле settings.py. Убедитесь, что ваш паук правильно выдает элементы изображений, которые может обработать ImagesPipeline.
5. Сохраните дополнительные сведения об изображениях (название, категория) в CSV-файле. Каждая строка должна соответствовать одному изображению и содержать URL изображения, локальный путь к файлу (после загрузки), название и категорию.

Решение

Краткое описание

Предлагаю "готовое" решение для дизайнеров, вебмастеров, блогеров, контент-менеджеров, UI-разработчиков и всех тех, кому важно быстро и бесплатно получить тематический набор (в среднем 20 шт.) высококачественных изображений с большим разрешением. И самое важное, без нарушения авторских прав.

Бесплатные фотографии с сайта unsplash.com распространяются под лицензией Creative Commons Zero (CC0), что означает, что их можно использовать бесплатно в любых целях без необходимости указания авторства или ссылки на источник.

Быстрый старт

Запуск решения осуществляется из корня директории 06.Tasks:

```
python runner.py
```

Если же выставить бит исполнения на стартовом скрипте `chmod +x runner.py`, то запуск можно выполнить так:

```
./runner.py
```

Решение запросит ввод темы для поиска, после чего осуществит поиск по сайту www.unsplash.com фотографий и сохранит их в папке с именем темы, а также создаст CSV-файл с информацией о скачанных изображениях.

Подробное описание

Развертывание решения на свой ПК

Для работы решения требуется python3 и дополнительные пакеты, которые можно установить с помощью pip:

```
pip install scrapy
pip install pillow
pip install scrapy-rotating-proxies # Нужен в случае использования прокси серверов и их карусели
```

Для развертывания самого решения на своем компьютере скачайте сжатый файл, содержащий корневую директорию 06.Tasks, распакуйте архив на вашем компьютере.

Корневую директорию 06.Tasks можно переименовать в любое другое имя, главное не менять внутреннюю структуру основного каталога.

Запуск решения

Windows

```
python runner.py
```

Linux

```
chmod +x runner.py
./runner.py
```

MacOs

```
python3 runner.py
```

После запуска появится запрос на ввод темы для поиска. Название темы можно вводить любую с кириллицей и из нескольких слов.

На выходе, в зависимости от популярности поиска, будет скачано в среднем 20 фотографий с большим разрешением и сохранено в папке с именем темы в корне проекта.

Количество фотографий может зависеть от темы и от формулировки запроса, в некоторых случаях фотографий может не найтись и вовсе.

В решении не используется пагинация, т.к. на сайте реализована "бесконечная" прокрутка на базе JavaScript. JS Scrapy не поддерживает но, возможна связка его с Selenium, который способен эмулировать работу пользователя за браузером.

Помимо загрузки фото, в корне решения создается табулированный CSV-файл с информацией о скачанных изображениях:

- Название фото
- Ссылка на картинку на сайте www.unsplash.com
- Локальный путь до картинки
- Котрольная сумма
- Статус загрузки
- Описание картинки сделанное ее автором
- Дата и время загрузки картинки на сайт www.unsplash.com

Параметры запуска

При работе с unsplash.com, блокировок из-за частых скачиваний фотографий, мною не замечено. Но, предусмотрел работу через прокси-серверы.

Для использования решения вместе с прокси-серверами и их каруселью, необходимо передать параметр при запуске решения:

```
runner.py -p ../path/proxy_list.txt
```

где:

- p - параметр активирующий поддержку прокси
- ../path/ относительный или абсалютный путь до фала со списком прокси
- proxy_list.txt - имя файла списка прокси

Формат файла со списком прокси, обычный текстовый, где каждая линия в виде IP:PORT:

```
38.0.105.76:3128
38.0.106.16:3128
118.0.10.78:3128
```

Тех.описание решения

1. Создание проекта Scrapy

Создаем директорию для выполнения 6-ой домашней работы:

```
mkdir 06.Tasks
```

В этой же директории создаем проект Scrapy по имени splash:

```
scrapy startproject splash . # вместо папки указываем . (текущая
директория), чтобы не плодить папки
```

В результате в папке проекта будут следующие файлы:

- scrapy.cfg - корневой файл настроек, ссылающийся на основной файл настроек settings.py
- splash/items.py - файл определяющий основные сущности (инфоблоки) с которыми по отдельности можно производить манипуляции
- splash/spiders/middleware.py - в этом файле прописываются модули, методы, библиотеки и даже программы на других языках программирования
- splash/pipelines.py - файл пайплайнов выполняемых после обработки items.py
- splash/settings.py - основные настройки, загружаемые при старте паука

- splash/spiders/ - пустая папка пауков

Для удобства работы, запуска и отладки проекта создадим в его корне стартовый скрипт runner.py:

```
touch runner.py
```

Стартовый скрипт runner.py

Импортируем класс управления сборщиком:

```
from scrapy.crawler import CrawlerProcess
```

Импортируем класс реактор сборщика:

```
from scrapy.utils.reactor import install_reactor
```

Импортируем класс для логирования

```
from scrapy.utils.log import configure_logging
```

Класс загрузки настроек из файла settings.py

```
from scrapy.utils.project import get_project_settings
from splash.spiders.unsplash import UnsplashSpider
```

Устанавливаем ту корневую директорию из которой запущен runner.py

```
import os
os.chdir(os.path.dirname(os.path.abspath(__file__)))
```

Стандартное условие выполнения следующего кода, если запущен сам файл runner.py

```
if __name__ == "__main__":
    # запрос к пользователя для ввода желаемой темы фоток для поиска
    #query = input("Введите тему для поиска: ")
    # настройки логирования
    configure_logging()
    # включение асинхронного реактора для реализации многопоточности

install_reactor("twisted.internet.asyncioreactor.AsyncioSelectorReactor")
    # настройка логирования
    configure_logging
    # загрузка настроек из файла settings.py
    settings = get_project_settings()
    # Установка новых, либо изменение старых параметров настроек
    settings.set('IMAGES_STORE', query)
    # создание процесса сборщика с указанными настройками
    process = CrawlerProcess(get_project_settings())
    включение в процесс сборщика конкретного паука и конкретной темы для
поиска
    process.crawl(UnsplashSpider, query='Python3')
    запуск на исполение процесса
    process.start()
```

2. Создание паука

Сбором информации в интернете занимаются "пауки", пауков может быть несколько, причем каждый паук способен обрабатывать по несколько сайтов.

Создадим одного паука по имен unsplash для домена www.unsplash.com:

```
scrapy genspider unsplash www.unsplash.com
```

В результате в папке splash/spiders/ будет создан файл по заданному имени unsplash.py

Файл паука unsplash.py

Вместо загрузки всей библиотеки scrapy, можно загрузить только нужные классы:

```
from scrapy.http import HtmlResponse
from splash.items import SplashItem
from scrapy.loader import ItemLoader
```

Чтобы передавать параметры в паука из вне, переопределим конструктор, автоматически созданного класса UnsplashSpider:

```
 def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.start_urls = [f"{self.protocol}
{self.allowed_domains[0]}/s{self.folder}/{kwargs.get('query')}"]
```

В этом классе изменим метод parse, так чтоб он начал "выковыривать" ссылку по тем html-тегам и артибутам, которые их содержат:

```
        links = response.xpath('//a[@itemprop="contentUrl"]/@href').getall()
        for link in links:
            link = self.protocol + self.allowed_domains[0] + link
            yield response.follow(link, callback=self.parse_photo)
```

Добавим метод parse_photo, который при переходе по ссылкам, в методе parse, будет, на вновь открытых страницах, искать информацию,которую мы хотим вытащить для дальнейшей обработки и сохранения:

```
 def parse_photo(self, response: HtmlResponse):
        loader = ItemLoader(item=SplashItem(), response=response)
        loader.add_value('url', response.url)
        loader.add_xpath('photos', '//button//img/@srcset')
        loader.add_xpath('name', '//h1/text()')
        loader.add_xpath('description', '//p[@style]/text()')
        loader.add_xpath('date', '//time/@datetime')
        yield loader.load_item()
```

В результате разбитая на блоки информация будет передана в файл splash/items.py

3. Инфоблоки (элементы информации)

Файл обработки инфоблоков items.py

Обработку блоков информации в данном файле можно осуществлять с помощью 7 методов, мы импортируем 3 самых распространенных из них:

```
from itemloaders.processors import TakeFirst, MapCompose, Compose
```

```
from datetime import datetime # для преобразования в формат datatime
```

Для начала изменить основной класс в данном файле SplashItem и зададим точно те блоки, которые мы передали из модуля unsplash.py

```
class SplashItem(scrapy.Item):
    url = scrapy.Field(output_processor=TakeFirst())
    photos = scrapy.Field(input_processor=MapCompose(proccess_photos))
    name = scrapy.Field(input_processor=Compose(proccess_name),
output_processor=TakeFirst())
    description = scrapy.Field(output_processor=TakeFirst())
    date = scrapy.Field(input_processor=Compose(proccess_date),
output_processor=TakeFirst())
    _id = scrapy.Field()
```

Если блоки приходят в том формате в котором мы хотим, с помощью TakeFirst() берем первый элемент списка, если нужен только один. Если информацию нужно обработать и очистить, то для этого блока определим методы, которые передадим в Compose() в случае одно элемента и MapCompose() в случае нескольких.

Нам нужно очистить и обработать 3 блока - ссылки на фото, название фото, и конвертировать формат даты:

```
def proccess_photos(photos):
    photos = photos.split(",")[-1].split()[0]
    return photos


def proccess_name(name):
    name = name[0].strip()
    return name

def proccess_date(date):
    date = datetime.strptime(date[0].strip(), '%Y-%m-%dT%H:%M:%S.%fZ')
    return date
```

4. Загрузка изображений

После обработки блоков данных они передаются в piplenine.py для работы с этими данными, если это фото или видео, то в пайплайне они скачиваются, если это название и дата, то в зависимости от настроек эти данные записываются в базу данных или выходной файл

Постобработка очередей в pipeline.py

Импортируем библиотеки, необходимые для работы с изображениями:

```
from scrapy.pipelines.images import ImagesPipeline
from scrapy.http import Request
```

Создадим класс насследник PhotoPipeline от родителя ImagesPipeline:

```
# Создадим дочерний класс от SplashPipeline
class PhotoPipeline(ImagesPipeline):
    def get_media_requests(self, item, info):
        if item['photos']:
            for img in item['photos']:
                try:
                    yield Request(img)
                except Exception as e:
                    print(e)

    def item_completed(self, results, item, info):
        if results:
            item['photos'] = [itm[1] for itm in results if itm[0]]
        print()
        return item
```

В данном классе определена логика работы с блоком информации photos полученного из модуля items.py.

Т.к. в блоке пришли ссылки до по ним в асинхронном режиме будет производиться скачивание файлов, и размещение их в директории указанной в файле основных настроек settings.py

5. Дополнительные сведения (настройки)

Scrapy организован таким образом, чтобы при создании и настройке проекта, при последующей эксплуатации не править основной код. Предусмотрены настраиваемые параметры и изменяемые свойства, а так же методы работы с ними через файл основной конфигурации settings.py

Основная конфигурация (settings.py)

Помимо настроек по-умолчиню, наш проект можно отрегулировать под собственные нужды.

Для корректного парсинга, рекомендую, в настройках указать, тот браузер, с помощью которого исследуем целевой сайт. В противном случае, будут появляться другие теги, либо атрибуты, из-за целевого отображения сайта под конкретный браузер:

```
USER_AGENT = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/122.0.0.0 Safari/537.36"
```

Количество одновременных подключений указываю 10, т.к. сайт proxyscrape.com на бесплатном тарифе выделяет 100 прокси, но до 10 одновременных к ним подключений, больше сессий ставить нет смысла:

```
CONCURRENT_REQUESTS = 10
```

Чтобы не палиться, отключаем прием печенек:

```
COOKIES_ENABLED = False
```

Включаем поддержку карусель прокси:

```
DOWNLOADER_MIDDLEWARES = {
   'rotating_proxies.middlewares.RotatingProxyMiddleware': 100,
   'rotating_proxies.middlewares.BanDetectionMiddleware': 110,
}
```

Указываем путь к списку прокси:

```
ROTATING_PROXY_LIST_PATH = '../../proxy_list.txt'
```

Включаем пайплайны, т.е. постобработку полученного контента:

```
ITEM_PIPELINES = {
   "splash.pipelines.SplashPipeline": 300, # обработчик по-умолчанию с
приоритетом ниже чем для фото
   "splash.pipelines.PhotoPipeline": 200, # обработчик для фото
}
```

Обязательно указать место для сохранения скачанных фоток, иначе никаких сообщений об ошибке не появится:

```
IMAGES_STORE = "photos"
```

Определим те инфоблоки, которые хотим включать в выходной файл CSV:

```
FEED_EXPORT_FIELDS = ['name', 'photos', 'description', 'date'])
```

Тонкая настройка экспортируемого формата CSV:

```
FEEDS = {
         query+'.csv': {
            'format': 'csv',
            'item_export_kwargs': {
               'include_headers_line': False,
               'delimiter': '\t',
            },
         }}
```

В результате тонких настроек CSV файл будет иметь разделитель полей табуляцию, не будет включать заголовки и будет иметь то имя, которое передается вместе с переменной query.

Заключение

В качестве тестового поиска, с помощью моего решения, ввел тему "Spring" (Весна).

На выходе получаем следующие данные:

- Папка с фотографиями весны высокого качества
- CSV файл с данными об изображениях