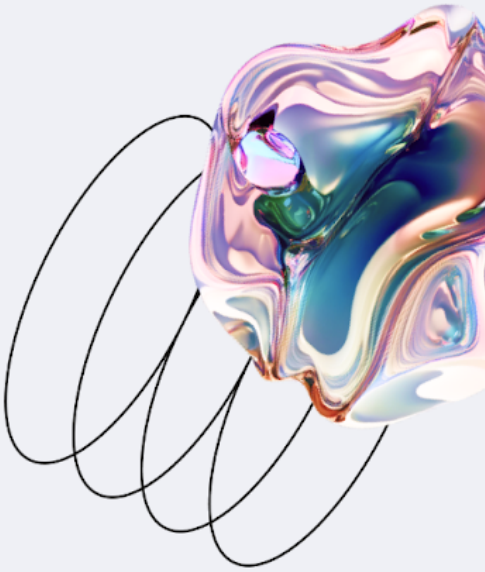
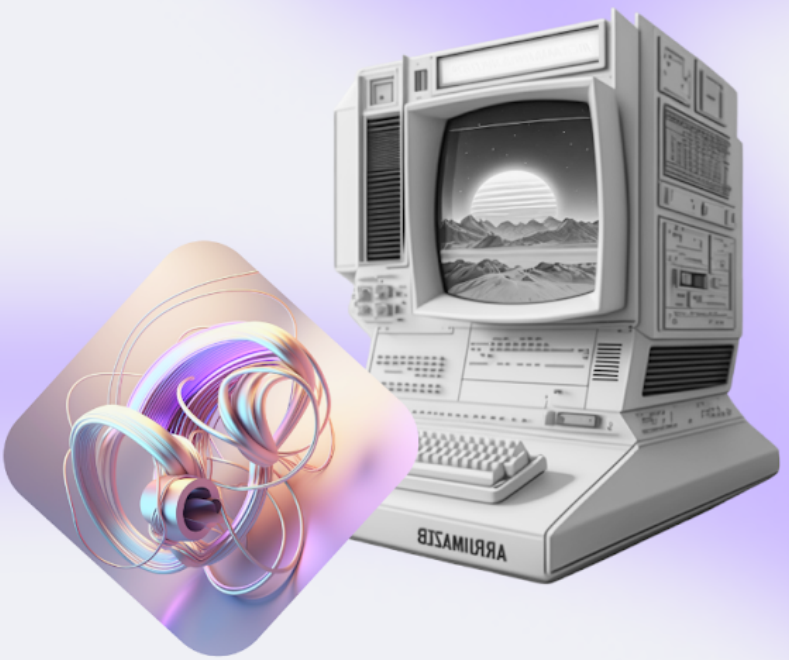
GeekBrains



Scrapy

Сбор и разметка данных



Оглавление

Словарь терминов

Краулинг (crawling) — процедура обнаружения и сбора информации о новых или прошедших обновление страницах для последующей загрузки в индекс поисковой системы.

Введение

Раньше для скрейпинга мы использовали парсеры BeautifulSoup и lxml. Однако они не предназначены для по-настоящему больших задач по сбору данных. Поэтому сегодня мы будем изучать фреймворк, который позволяет решать комплексные и сложные задачи, — Scrapy.

Scrapy – это бесплатный фреймворк для веб-краулинга и скрейпинга. Он находится в открытом доступе и написан на Python.

Краулинг (crawling) — процедура обнаружения и сбора информации о новых или прошедших обновление страницах для последующей загрузки в индекс поисковой системы.

В контексте веб-скрейпинга под краулингом понимается автоматизированный процесс систематического посещения веб-сайтов, навигации по их страницам и обнаружения релевантной информации на основе заранее определенных правил.

Отвечают за краулинг специальные роботы — краулеры. Их также называют поисковыми роботами, пауками или ботами. Пауки — это, по сути, скрипты на Python, которые определяют, как следует перемещаться по сайту, какую информацию извлекать и как ее обрабатывать. Пауки Scrapy хорошо настраиваются, их можно адаптировать к требованиям веб-скрейпинга.

Scrapy позволяет выполнять скрейпинг сотен страниц в минуту. Производительность зависит от мощности вашего процессора и от пропускной способности интернета.

Компоненты Scrapy

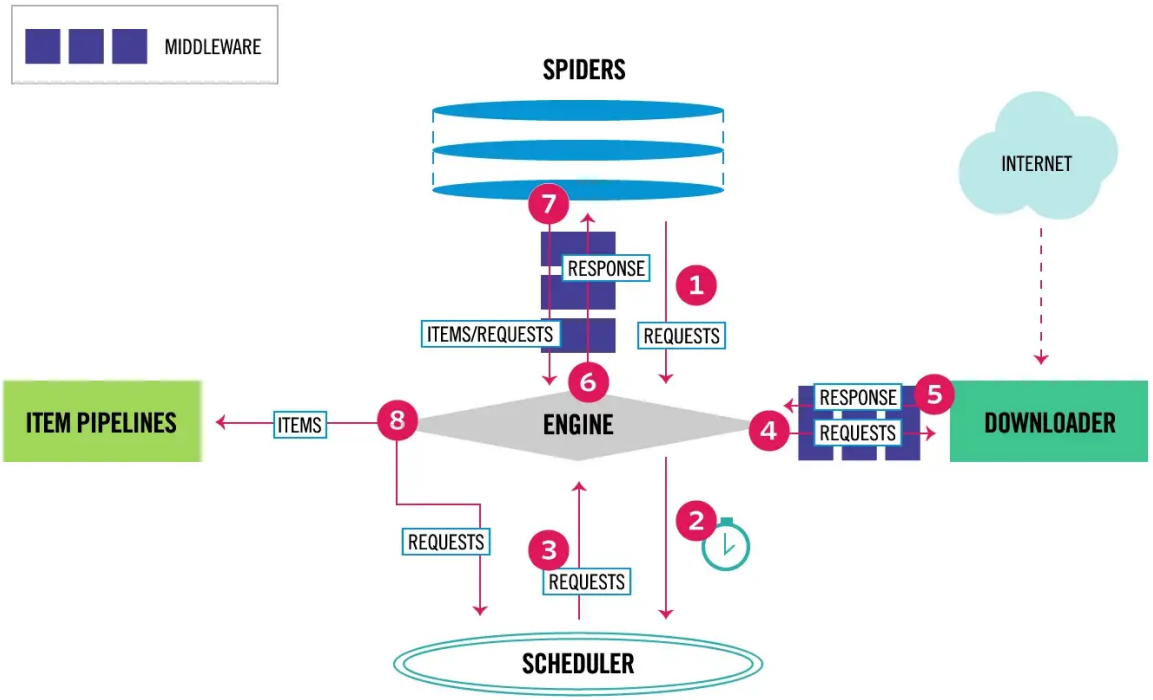
Scrapy состоит из пяти основных компонентов:

- **Spiders** (пауки) — компонент, в котором мы будем определять, что хотим извлечь из веб-страницы. В Scrapy есть пять видов пауков, определенных как python-классы:

- scrapy.Spyder
- CrawlSpider
- XMLFeedSpider
- CSVSpider
- SitemapSpider

- **Pipelines** — компонент относится к работе с данными, которые мы извлекаем. Например, если мы хотим обработать данные — очистить, удалить дубликаты или сохранить данные во внешней БД — используем pipelines.
- **Middlewares** — имеет отношение к запросу, отправленному на сайт, и ответу, который мы получаем. Например, если мы хотим обработать запрос путем внедрения (инъекции) пользовательских заголовков или использовать прокси, мы можем сделать это через middleware.
- **Engine** — отвечает за координацию между компонентами. Иными словами, обеспечивает согласованность всех выполняемых операций.
- **Scheduler** — отвечает за сохранение порядка выполнения операций. Технически говоря, это простая структура данных под названием «очередь» — абстрактный тип данных с дисциплиной доступа к элементам «первый пришел — первый вышел» или FIFO.

Давайте разберемся, как это все работает.



Источник: docs.scrapy.org

Схема выше дает ясное представление о работе Scrapy. Давайте разберем все шаги.

1. **Spiders.** Это сценарий на Python, который определяет, как следует перемещаться по сайту, какие данные извлекать и как их обрабатывать. Это основной компонент, который определяет, что будет делать веб-скрейпер.

2. **Engine.** Это центральный координатор, который управляет всем процессом скрейпинга. Он получает стартовые запросы от паука, взаимодействует с другими компонентами и управляет общим потоком данных.

3. **Scheduler.** Когда движок получает запросы от паука, он отправляет их планировщику. Планировщик ставит запросы в очередь и отправляет их обратно в движок по одному, когда приходит время их обрабатывать.

4. **Downloader.** Движок отправляет запросы на загрузчик, который отвечает за получение веб-страниц путем выполнения HTTP-запросов. Как только загрузчик получает HTTP-ответ, он отправляет его обратно движку.

5. **Middlewares.** Промежуточные модули загрузчика используются для обработки запросов и ответов между движком и загрузчиком. Они могут изменять запросы или обрабатывать ошибки, возникающие в процессе загрузки.

6. **Spiders (снова).** Движок передает загруженную веб-страницу (ответ) обратно пауку, который анализирует содержимое HTML, извлекает нужные данные и генерирует новые запросы для перехода по ссылкам или пагинации.

7. **Item Pipelines.** После извлечения данных паук отправляет их в конвейеры элементов. Конвейеры — это серия этапов обработки, которая позволяет очистить, проверить и преобразовать извлеченные данные. Каждый компонент конвейера обрабатывает данные определенным образом и передает их следующему компоненту конвейера.

8. **Data Storage.** После обработки данных конвейерами элементов они сохраняются в нужном формате (например, JSON, CSV или в базе данных) для дальнейшего анализа или использования.

Эти шаги повторяются до тех пор, пока в планировщике не останется ни одного запроса.

Подробный обзор архитектуры Scrapy в официальной документации: [Architecture overview — Scrapy 2.10.0 documentation].

Установка Scrapy и создание проекта

Переходим к работе в Scrapy. Переключаемся на свою виртуальную среду и устанавливаем Scrapy командой:

```
conda install -c conda-forge scrapy
```

После установки введем в командную строку команду scrapy. Во-первых, она поможет убедиться, что scrapy установился. Во-вторых, мы увидим список доступных команд, с которыми будем разбираться в ходе урока:

```
(GeekBrain) petrrubin@MacBook-Pro-Petr / % scrapy
Scrapy 2.7.1 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  commands
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  runspider     Run a self-contained spider (without creating a project)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy

  [ more ]      More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
(GeekBrain) petrrubin@MacBook-Pro-Petr / %
```

Чтобы создать новый проект Scrapy, используем команду startproject. За ней нужно ввести название проекта, например trading_economics:

```
scrapy startproject trading_economics
```

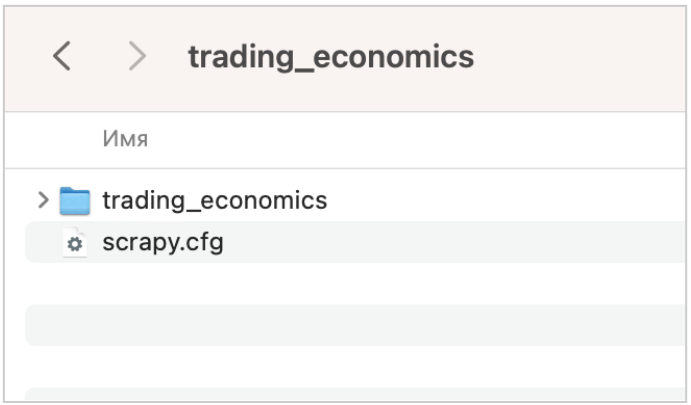
Мы создали новый проект Scrapy:

```
(GeekBrain) petrrubin@MacBook-Pro-Petr projects % scrapy startproject trading_economics
New Scrapy project 'trading_economics', using template directory '/Users/petrrubin/opt/anaconda3/e
nvs/GeekBrain/lib/python3.9/site-packages/scrapy/templates/project', created in:
    /Users/petrrubin/projects/trading_economics

You can start your first spider with:
    cd trading_economics
    scrapy genspider example example.com
(GeekBrain) petrrubin@MacBook-Pro-Petr projects %
```

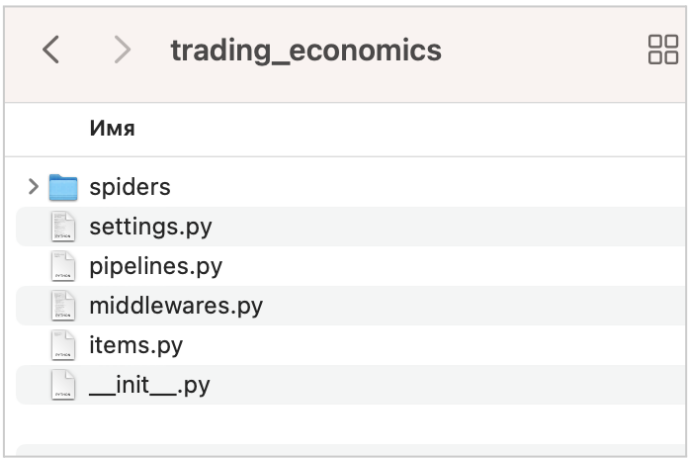
Здесь указано, что мы можем запустить нашего первого паука, перейдя в директорию trading_economics, а затем используя команду genspider для создания паука.

Но прежде посмотрим, что было создано, когда мы использовали команду startproject. Откроем папку проекта trading_economics:



В папке проекта мы видим, что была создана еще одна папка с названием проекта и файл scrapy.cfg. Этот файл очень важен для запуска пауков, которых вы создаете, также он используется для развертывания пауков.

Посмотрим, что внутри папки:



Папка spiders — место, где живут все пауки. Она содержит только пустой файл init.py.

Файл items.py используется для очистки данных, которые мы собираем, и для хранения данных в некоторых полях, которые мы создаем. Например, здесь есть поле name, равное scrapy.Field().

Файл middlewares.py имеет отношение к объектам запроса и ответа. Это то, что мы разбирали на схеме выше, где у нас есть два типа посредников (middleware), один из которых — spider middleware, отвечающий за возврат данных, а второй — downloader middleware, отвечающий за загрузку HTML-разметки сайта.

Файл middlewares.py содержит пользовательские классы, которые действуют как шлюзы для обработки запросов и ответов в процессе веб-скрейпинга. Их можно использовать для изменения запросов перед их отправкой загрузчику или для обработки ответов перед их передачей обратно пауку.

Классы, определенные в файле middlewares.py, могут помочь справиться с различными сценариями, например:

- **Управление перенаправлениями.** Если веб-сайт перенаправляет пользователя на другую страницу, middleware может обработать перенаправление и гарантировать, что паук продолжит работать, как ожидалось.
- **Обработка ошибок.** Если в процессе загрузки возникает ошибка (например, тайм-аут или неработающая ссылка), middleware может перехватить ошибку и предпринять соответствующие действия, например, зарегистрировать ошибку или повторить запрос.

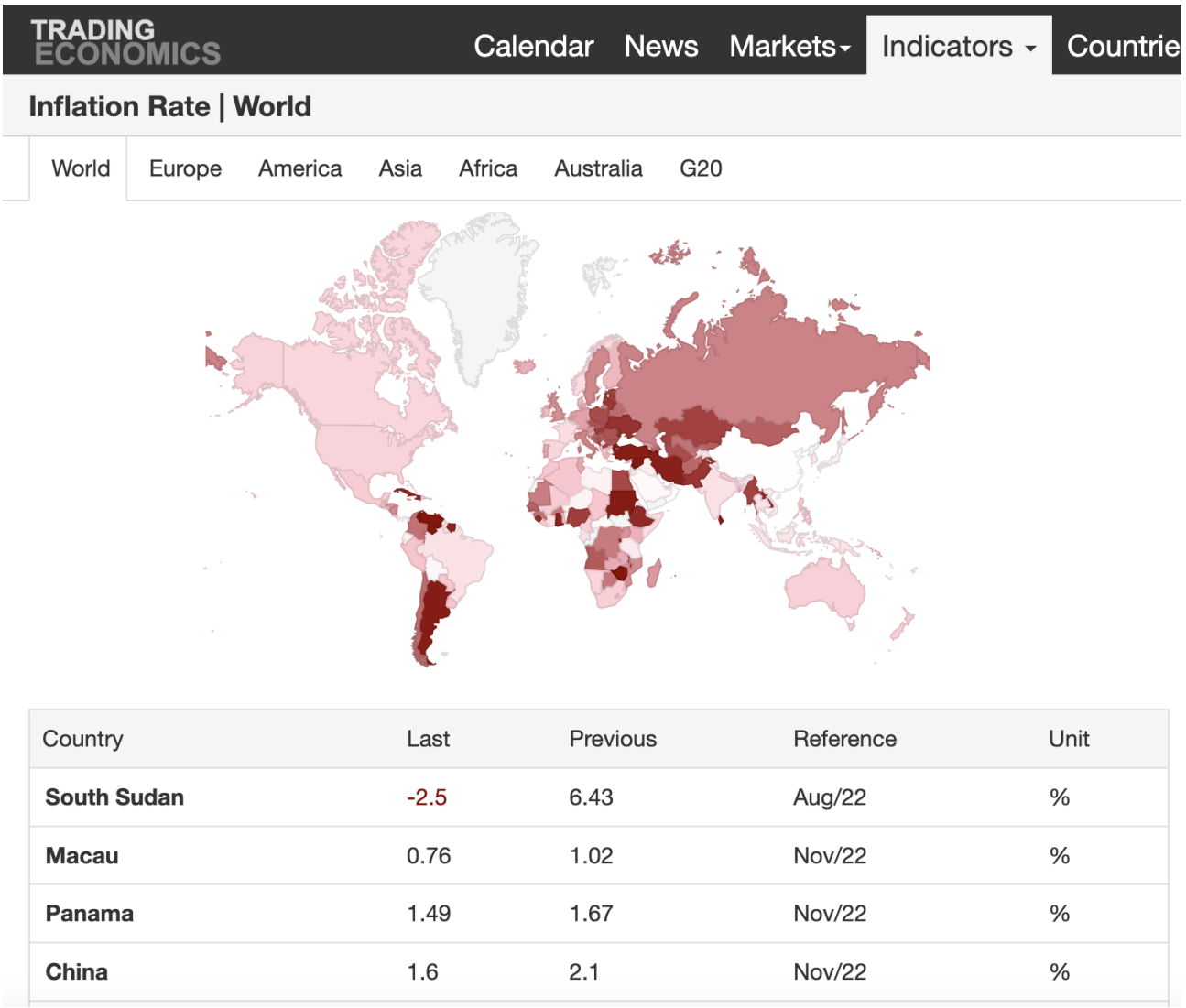
Здесь же можно настроить пользовательского агента и использование прокси-серверов.

Файл pipelines.py используется для хранения элементов, которые мы скрейпируем, в базе данных.

Файл settings.py нужен для настройки или добавления дополнительной конфигурации в проект.

Посмотрим сайт, с которым будем работать: Inflation Rate - Countries - List | World.

Сайт называется Trading Economics и содержит статистическую информацию по 196 странам мира. Мы собираемся выполнить скрейпинг страницы со статистикой по инфляции, а именно открыть ссылку каждой страны и получить данные из таблицы.



Country	Last	Previous	Reference	Unit
South Sudan	-2.5	6.43	Aug/22	%
Macau	0.76	1.02	Nov/22	%
Panama	1.49	1.67	Nov/22	%
China	1.6	2.1	Nov/22	%

Возвращаемся в терминал и сгенерируем паука. Вводим команды scrapy genspider, затем имя паука — назовем его countries — и URL сайта. Также удалим название протокола https://, потому что Scrapy по умолчанию будет использовать протокол HTTP и нам придется изменить его позже.

```
scrapy genspider countries
"tradingeconomics.com/country-list/inflation-rate?continent=world"
```

После выполнения команды в папке spiders у нас появился файл countries.py. Откроем файл — это паук. Собственно, это класс CountriesSpider, и этот класс наследует от класса spider, расположенного в модуле scrapy.

У каждого паука уникальное имя. Мы назвали его countries, поэтому свойство name имеет значение «countries». В одном проекте может быть несколько пауков, но у каждого паука должно быть уникальное имя.

Далее у нас есть список разрешенных доменов. Этот список должен содержать все доменные имена, к которым паук обращается. В списке разрешенных доменов не нужно прописывать протокол HTTP в начале адреса.

Далее есть список start_urls. В нем мы вписываем все ссылки, которые хотим скрейпировать, однако, как было сказано, Scrapy по умолчанию использует протокол http, а сайт, с которым мы работаем, использует https. Поэтому добавим здесь 's'.

Наконец, у нас есть функция parse, которая анализирует ответ от паука. Например, когда запрос отправляется на ссылку в списке start_urls, мы сможем получить ответ в функции parse.

Оболочка Scrapy

Мы переходим к работе в оболочке Scrapy. Это прекрасный инструмент для создания паука.

Scrapy Shell — это интерактивный инструмент командной строки, который поставляется вместе с фреймворком Scrapy. Он позволяет тестировать и отлаживать код веб-скрейпинга, например селекторы XPath или CSS, без необходимости запускать всего паука. Это особенно полезно, когда вы разрабатываете или совершенствуете логику скрейпирования.

Запустим оболочку командой в терминале:

```
scrapy shell
```

Теперь у нас есть доступ к некоторым объектам scrapy — к самому объекту scrapy, объекту crawler, item и settings:

Также и у нас есть некоторые полезные ссылки — это то, что мы будем использовать большую часть времени. Чтобы открыть URL, мы используем метод fetch. Мы можем передать URL целевого сайта в качестве аргумента или создать объект request и передать его в качестве аргумента этой функции. Затем у нас есть команда shelp и функция view для открытия ответа в браузере.

Итак, чтобы посетить веб-сайт с помощью оболочки scrapy, мы можем использовать метод fetch. В качестве аргумента вводим URL сайта.

```
fetch
("https://tradingeconomics.com/country-list/inflation-rate?contin
ent=world")
```

Альтернативный способ:

```
r =
scrapy.Request(url="https://tradingeconomics.com/country-list/inf
lation-rate?continent=world")

fetch(r)
```

Мы получим лог типа INFO: Spider opened. Затем у нас есть другой лог типа DEBUG: Attempting to acquire lock.

Далее у нас есть DEBUG: Crawled (200) <GET...

GET — это тип запроса, который был отправлен, а 200 — код состояния ответа, который означает, что Scrapy удалось выполнить запрос к этому URL и мы получили ответ.

Давайте посмотрим HTML-разметку сайта, которую мы получили. Вводим команду:

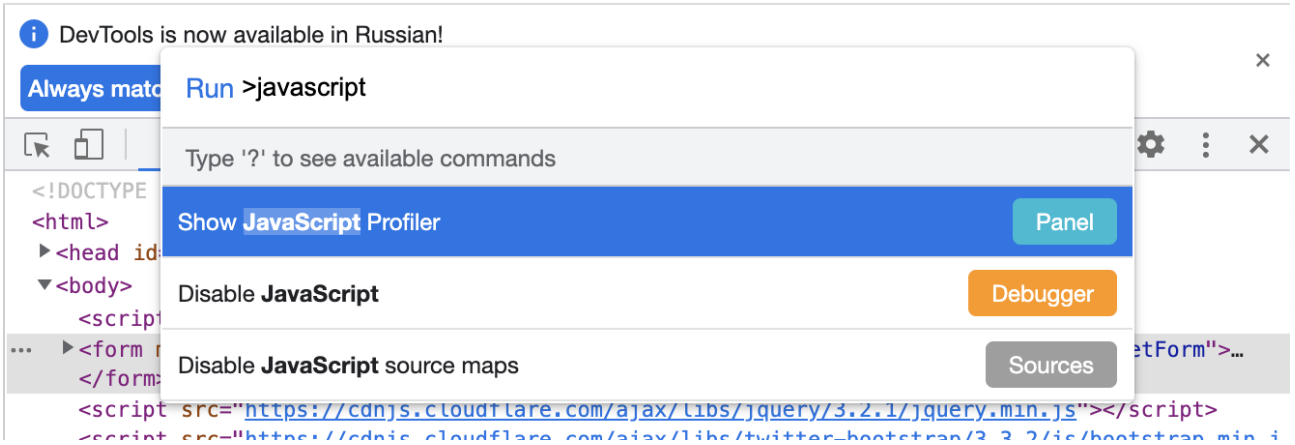
```
response.body
```

Из оболочки Scrapy можно также посмотреть, как пауки видят сайты. В командной строке вызовем функцию view и передадим объект response в качестве аргумента:

```
view(response)
```

Ответ будет открыт в браузере и автоматически сохранится во временном файле. Он выглядит так же, как и оригинальный, однако пауки не видят сайты так, как мы. Паук видит сайты без JavaScript.

Давайте отключим JavaScript: откроем DevTools, нажмем Ctrl + Shift + P, в поисковой строке введем «javascript» и выберем Disable JavaScript:



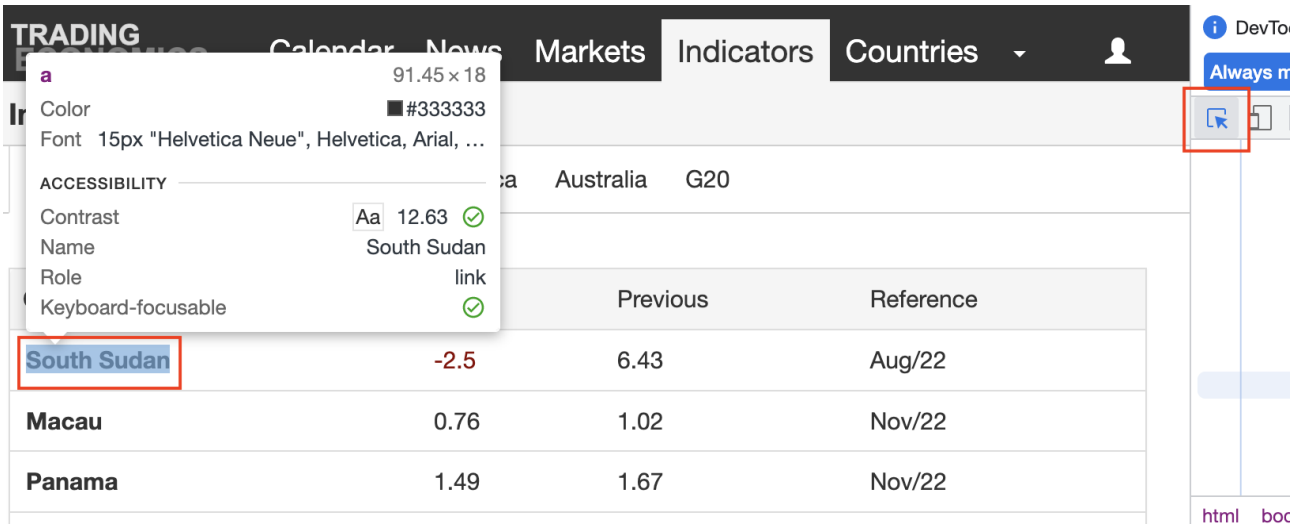
Обновим страницу. Теперь видно, что часть информации, в частности карта, не отображается. Это та версия сайта, которая видна Scrapy.

Скрейпинг с помощью Scrapy

Мы переходим к созданию паука. В VS Code открываем файл countries.py.

Перейдем в Chrome на страницу, информацию с которой собираемся скрейпить. Нам нужно получить название каждой страны и ссылку. Затем мы пройдем по всем ссылкам и получим данные из подлежащих страниц.

Включаем DevTools, выбираем Selection Tool и кликаем на названии первой страны:



В DevTools мы видим, что название страны и ссылка содержатся в теге <a>. Теперь нам нужно получить выражение XPath для доступа к этому элементу. На клавиатуре нажимаем Ctrl + F и в поисковой строке вводим выражение:

```
//td/a
```

Видим, что выражение указывает на нужные нам данные. Теперь перейдем в терминал и проверим, что все работает. Создадим переменную countries и воспользуемся командой response.xpath:

```
countries = response.xpath("//td/a")
countries
```

```
[In [5]: countries = response.xpath("//td/a")

[In [6]: countries
 Out[6]:
 [<Selector xpath='//td/a' data='<a href="/south-sudan/inflation-cpi">...'>,
  <Selector xpath='//td/a' data='<a href="/macau/inflation-cpi">\r\n    ...'>,
  <Selector xpath='//td/a' data='<a href="/panama/inflation-cpi">\r\n    ...'>,
  <Selector xpath='//td/a' data='<a href="/china/inflation-cpi">\r\n    ...'>,
  <Selector xpath='//td/a' data='<a href="/hong-kong/inflation-cpi">\r\n...'>,
  <Selector xpath='//td/a' data='<a href="/oman/inflation-cpi">\r\n     ...'>,
```

Мы получили список объектов Selector, содержащих атрибут data, представляющий собой полный элемент **a** или узел всех стран на странице.

Возвращаемся в VS Code. В функцию parse вписываем выражение:

```
    def parse(self, response):
        countries = response.xpath("//td/a")
```

Из списка countries нам нужно вытащить ссылки и названия стран. Пройдемся в цикле по всем объектам Selector, которые у нас находятся в списке countries. Так как каждый элемент списка — это объект Selector, мы можем применить к нему выражение XPath.

Создадим две переменных: name и link. Первая будет содержать название страны, вторая — ссылку. Прописываем выражение XPath применительно к каждой переменной. Для name нам нужно получить текстовое содержимое узла <a> — воспользуемся функцией text(). Поскольку мы выполняем выражение XPath для объекта Selector, а не для объекта response, то должны использовать точку перед выражением XPath ".//text()". Поскольку мы хотим вернуть данные в виде строки, то вызываем метод get(). Для ссылки на страну нам нужно значение атрибута href, поэтому заменим функцию text на @href. Для создания словаря используем оператор yield:

```
    def parse(self, response):
        countries = response.xpath("//td/a")
        for country in countries:
            name = country.xpath(".//text()").get()
            link = country.xpath(".//@href").get()
            yield {
                'country_name' : name,
                'link' : link}
```

Сохраняем файл. Возвращаемся в терминал, выходим из Scrapy shell командой exit(). Теперь запустим нашего паука:

```
scrapy crawl countries
```

Мы получили название каждой страны и соответствующие ссылки:

```
2022-12-27 15:48:44 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://tradingeconomics.com/country-list/inflation-rate?continent=world>
 (referer: None)
2022-12-27 15:48:44 [scrapy.core.scraper] DEBUG: Scraped from <200 https://tradingeconomics.com/country-list/inflation-rate?continent=world>
{'country_name': '\r\n                                        South Sudan\r\n                                    ', 'link': '/south-sudan/in
flation-cpi'}
2022-12-27 15:48:44 [scrapy.core.scraper] DEBUG: Scraped from <200 https://tradingeconomics.com/country-list/inflation-rate?continent=world>
{'country_name': '\r\n                                        Macau\r\n                                    ', 'link': '/macau/inflation-cpi'
}
2022-12-27 15:48:44 [scrapy.core.scraper] DEBUG: Scraped from <200 https://tradingeconomics.com/country-list/inflation-rate?continent=world>
{'country_name': '\r\n                                        Panama\r\n                                    ', 'link': '/panama/inflation-cp
i'}
2022-12-27 15:48:44 [scrapy.core.scraper] DEBUG: Scraped from <200 https://tradingeconomics.com/country-list/inflation-rate?continent=world>
{'country_name': '\r\n                                        China\r\n                                    ', 'link': '/china/inflation-cpi'
}
```

Следующая наша задача — научиться переходить по ссылкам с помощью Scrapy. Пока что мы получили данные только с одной страницы. Сейчас нам нужно сделать следующее: взять каждую ссылку на страну и отправить ей запрос.

При использовании оболочки Scrapy мы вызывали метод fetch, но, к сожалению, мы не можем использовать этот метод внутри паука. Вместо этого мы должны вызвать класс request.

Давайте перепишем yield на scrapy.Request(). В качестве аргумента укажем url = link, которая является ссылкой на страну.

```
    def parse(self, response):
        countries = response.xpath("//td/a")
        for country in countries:
            name = country.xpath(".//text()").get()
            link = country.xpath(".//@href").get()
            yield scrapy.Request(url=link)
```

Сохраним файл и вернемся к командной строке, чтобы запустить паука:

```
scrapy crawl countries
```

Мы получили ошибку — ValueError: Missing scheme in request url: /south-sudan/inflation-cpi. Отсутствие схемы означает, что Scrapy не знает, использует ли ссылка, по которой мы пытаемся перейти, протокол HTTP или HTTPS.

Вернемся к Chrome и посмотрим на элемент **a**. Его значение атрибута href не содержит ни значение протокола (HTTP или HTTPS), ни доменного имени tradingeconomics.com. Значит, этот URL является относительным, поэтому нам нужно это исправить и объединить доменное имя tradingeconomics.com со ссылкой на страну.

Создадим переменную absolute_url и используем функцию urljoin(), в качестве аргумента передадим link:

```
    def parse(self, response):
        countries = response.xpath("//td/a")
        for country in countries:
            name = country.xpath(".//text()").get()
            link = country.xpath(".//@href").get()
            absolute_url = response.urljoin(link)
            yield scrapy.Request(url=absolute_url)
```

Сохраним файл и еще раз запустим скрейпинг из терминала. На этот раз получим результат:

```
2022-12-27 16:13:41 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://tradingeconomics.com/south-sudan/inflation-cpi> (referer: https:/
/tradingeconomics.com/country-list/inflation-rate?continent=world)
2022-12-27 16:13:41 [scrapy.dupefilters] DEBUG: Filtered duplicate request: <GET https://tradingeconomics.com/south-sudan/inflation-cpi> - n
o more duplicates will be shown (see DUPEFILTER_DEBUG to show all duplicates)
2022-12-27 16:13:42 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://tradingeconomics.com/japan/inflation-cpi> (referer: https://tradi
ngeconomics.com/country-list/inflation-rate?continent=world)
2022-12-27 16:13:42 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://tradingeconomics.com/ecuador/inflation-cpi> (referer: https://tra
dingeconomics.com/country-list/inflation-rate?continent=world)
```

Подобным способом мы уже пользовались в прошлых уроках, однако в Scrapy есть альтернативный вариант без необходимости переделывать ссылку вручную — использование response.follow(). В качестве аргумента мы передадим относительный URL - link:

```
    def parse(self, response):
        countries = response.xpath("//td/a")
        for country in countries:
            name = country.xpath(".//text()").get()
            link = country.xpath(".//@href").get()
            #absolute_url = response.urljoin(link)
            yield response.follow(url=link)
```

Итак, мы успешно отправили запрос на каждую ссылку страны, теперь нам нужно получить информацию из этих страниц. Для этого нам нужно выполнить парсинг ответа, полученного от response.follow. Создадим еще один метод parse_country(), который будет принимать те же аргументы, что и метод parse(), то есть self и response, а также установим callback, равный self.parse_country, в ключевое слово yield. Таким образом, когда Scrapy посылает запрос к каждой ссылке страны, ответ будет отправлен в метод parse_country().

```
import scrapy
class CountriesSpider(scrapy.Spider):
    name = 'countries'
    allowed_domains = ['tradingeconomics.com']
    start_urls =
['https://tradingeconomics.com/country-list/inflation-rate?continent=
world']

    def parse(self, response):
        countries = response.xpath("//td/a")

        for country in countries:
            name = country.xpath(".//text()").get()
            link = country.xpath(".//@href").get()
```

```
            #absolute_url = response.urljoin(link)

            yield response.follow(url=link,
callback=self.parse_country)

    def parse_country(self, response):
```

Вернемся в Chrome и перейдем по первой ссылке страны. Наша задача — выполнить парсинг трех столбцов таблицы с показателями инфляции:

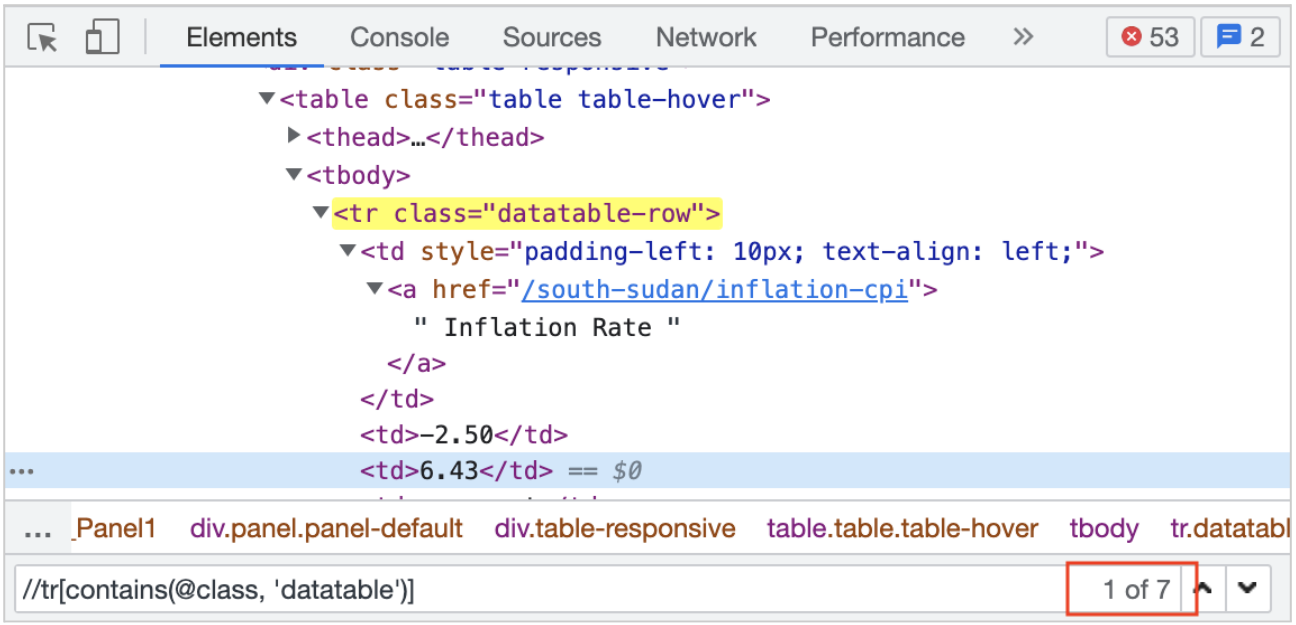
Related	Last	Previous	Unit	Reference
Inflation Rate	-2.50	6.43	percent	Aug 2022
Food Inflation	-5.30	1.66	percent	Aug 2022
Inflation Rate MoM	-1.45	4.97	percent	Aug 2022
CPI Transportation	16601.00	17103.01	points	Aug 2022
CPI Housing Utilities	10127.98	9246.63	points	Aug 2022
Core Consumer Prices	9730.66	9619.74	points	Aug 2022
Consumer Price Index CPI	16199.66	16438.56	points	Aug 2022

В качестве тренировки попробуйте самостоятельно написать XPath-выражение для извлечения данных первой строки, а именно названия — Inflation Rate и чисел — Last и Previous.

Сейчас нам нужно написать XPath-выражения для получения доступа ко всем строкам таблицы. Сделать это можно, например, так:

```
//tr[contains(@class, 'datatable')]
```

Как видно в строке поиска, найдено 7 элементов, что соответствует количеству строк в таблице (это число может быть другим, если изменится содержание сайта):



Теперь в методе parse_country создаем переменную rows, которая будет содержать нужные нам строки таблицы:

```
def parse_country(self, response):
        rows = response.xpath("//tr[contains(@class, 'datatable')]")
```

Далее в цикле будем проходить по строкам и извлекать название показателя инфляции, текущее и предыдущее его значение, а также с помощью ключевого слова yield записывать данные в словарь:

```
        for row in rows:
            related = row.xpath(".//td/a/text()").get()
            last = float(row.xpath(".//td[2]/text()").get())
            previous = float(row.xpath(".//td[3]/text()").get())

            yield {
                'related' : related,
                'last' : last,
                'previous' : previous
            }
```

Последнее, что нам осталось добавить в словарь — название страны. Название страны мы получаем внутри метода parse, в цикле for у нас есть выражение name = country.xpath(".//text()").get(). Нам нужно передать значение этой переменной в метод parse_country, чтобы вставить его в словарь. Чтобы отправить данные между двумя методами, мы можем использовать аргумент meta в response.follow. Аргумент meta — это словарь, в котором мы передаем пары «ключ-значение» из метода в метод вместе с самим запросом.

Нам нужно передать имя, поэтому установим значение meta следующим образом:

```
yield response.follow(url=link, callback=self.parse_country,
meta={'country_name' : name})
```

Теперь внутри метода parse_country создадим переменную name и присвоим ей значение по ключу, указанному в квадратных скобках:

```
name = response.request.meta['country_name']
```

Добавим в наш словарь название страны

```
            yield {
                'country_name' : name,
                'related' : related,
                'last' : last,
                'previous' : previous
            }
```

Сохраним и запустим код в терминале. Ответ должен быть таким:

Как видно, поля с названием страны и названием показателя инфляции содержат специальные символы. Чтобы их удалить, воспользуемся методом Python strip().

Полный код:

```
import scrapy

class CountriesSpider(scrapy.Spider):
   name = 'countries'
   allowed_domains = ['tradingeconomics.com']
   start_urls =
['https://tradingeconomics.com/country-list/inflation-rate?continent=world']

   def parse(self, response):
       countries = response.xpath("//td/a")
       for country in countries:
           name = country.xpath(".//text()").get().strip()
           link = country.xpath(".//@href").get()
```

```
            #absolute_url = response.urljoin(link)
            yield response.follow(url=link, callback=self.parse_country,
meta={'country_name' : name})

    def parse_country(self, response):
        rows = response.xpath("//tr[contains(@class, 'datatable')]")

        for row in rows:
            related = row.xpath(".//td/a/text()").get().strip()
            last = float(row.xpath(".//td[2]/text()").get())
            previous = float(row.xpath(".//td[3]/text()").get())
            name = response.request.meta['country_name']

            yield {
                'country_name' : name,
                'related' : related,
                'last' : last,
                'previous' : previous
            }
```

Чтобы указать агент пользователя в Scrapy, нужно изменить файл settings.py в проекте Scrapy.

Откроем файл settings.py в нашем проекте Scrapy. Найдем параметр USER_AGENT. По умолчанию он будет закомментирован. Откомментируйте эту строку и замените строку агента пользователя по умолчанию на желаемый агент пользователя.

Чтобы установить паузу или задержку между HTTP-запросами в Scrapy, вы можете использовать параметр DOWNLOAD_DELAY. Он позволяет указать количество секунд, которое нужно ждать между запросами, что поможет избежать ограничения скорости или блокировки целевого веб-сайта.

Наконец, по умолчанию Scrapy использует параметр CONCURRENT_REQUESTS для управления количеством одновременных запросов. Если вы хотите сделать своего паука еще более бережным к ресурсам сервера сайта, вы можете уменьшить количество одновременных запросов, установив меньшее значение.

Итак, когда все настройки сделаны, давайте экспортируем данные. Scrapy позволяет нам экспортировать данные в JSON, CSV или XML. В терминале мы используем ту же команду, что и для запуска паука, — scrapy crawl countries. Но для экспорта добавляем -o и название файла:

```
scrapy crawl countries -o inflation.json
```

В директории проекта появляется новый файл inflation.json, можно открыть его и посмотреть данные. Чтобы экспортировать данные в формат CSV или XML, используем ту же команду, но меняем расширение в названии файла на соответствующее — .csv или .xml.

Заключение

На лекции мы познакомились с очень мощным инструментом сбора данных из интернета. Того, что мы изучили сегодня, достаточно для выполнения большинства задач скрейпинга, однако функционал Scrapy этим не ограничен. На следующей лекции мы займемся скрейпингом картинок и файлов, а также изучим ряд дополнительных функций Scrapy.

Что можно почитать еще

1. Официальная документация Scrapy
2. Web scraping вашего сайта: непрошеные гости и как их встречают
3. Официальная документация Scrapy