ВЕБ-СКРЕЙПИНГ С BEAUTIFULSOUP

Как собрать данные, которых нет в готовых базах

Зайцев Илья
Telegram: @illakal

29.08.2026

СТРУКТУРА ЗАНЯТИЯ

01

Зачем это нужно

Переменная, которой нет в Bloomberg

02

Как устроена страница

HTTP, HTML, DOM и инспектор браузера

03

BeautifulSoup на практике

Ядро библиотеки и кейсы

04

Как не получить бан по IP

Вежливый парсинг и право

05

Границы инструмента

Альтернативы и ограничения

Установка: `pip install requests beautifulsoup4 lxml cssselect pandas matplotlib`

ЗАЧЕМ ПАРСИТЬ?

Настроения, внимание и разногласия инвесторов не продаются как датасет. Эти переменные не скачивают — их конструируют из открытых текстов.

Tone	Attention	Disagreement
Тональность	**Внимание**	**Разногласия**
Каким был информационный фон вокруг эмитента в этот день?	*Насколько розничный инвестор вообще заметил событие?*	*Расходятся ли участники рынка в оценке одной и той же новости?*
ГДЕ ЖИВУТ ДАННЫЕ	ГДЕ ЖИВУТ ДАННЫЕ	ГДЕ ЖИВУТ ДАННЫЕ
Новостные ленты, пресс-релизы, раскрытия	Поисковые запросы, просмотры, число упоминаний	Форумы, соцсети, комментарии к прогнозам

Что мы делаем сегодня: HTML-страница → DataFrame → переменная уровня «эмитент × день» → регрессия

Готовые индексы настроений существуют — но они закрыты, платны и не покрывают российский рынок.

ТАК СОБРАНЫ ДАННЫЕ В ЭТИХ РАБОТАХ

Во всех пяти случаях переменной не существовало в готовом виде — авторы написали парсер.

ГОД	АВТОРЫ И РАБОТА	ЧТО ПАРСИЛИ	КАКАЯ ПЕРЕМЕННАЯ
2015	**Da, Engelberg, Gao** *The Sum of All FEARS* **RFS 28(1)**	Частота поисковых запросов («recession», «bankruptcy»)	Индекс настроений FEARS → доходность и волатильность
2018	**Bartov, Faurel, Mohanram** *Can Twitter Help Predict Firm-Level Earnings?* **TAR 93(3)**	Сообщения в Twitter перед отчётностью	Предсказание сюрприза в прибыли
2019	**Green, Huang, Wen, Zhou** *Crowdsourced Employer Reviews and Stock Returns* **JFE 134(1)**	Отзывы сотрудников на Glassdoor	Изменение оценок → будущая доходность
2020	**Cookson, Niessner** *Why Don't We Agree?* **JF 75(1)**	Сообщения инвесторов на StockTwits	Мера разногласий → объём торгов
2023	**Cookson, Engelberg, Mullins** *Echo Chambers* **RFS 36(2)**	Подписки и лента на StockTwits	Информационные пузыри у розничных инвесторов

Общее: источник открыт, инструмент прост, а ценность — в том, что переменную никто до вас не собирал.

ПЯТЬ ШАГОВ, ИЗ КОТОРЫХ СОСТОИТ ПАРСЕР

Каждый шаг — это одна ячейка в ноутбуке. Дальше мы просто проходим их по порядку.

1	2	3	4	5
Запрос	**Разбор**	**Извлечение**	**Очистка**	**Сохранение**
`requests.get(url)`	`BeautifulSoup(html)`	`find_all / select`	`regex + to_float`	`pd.DataFrame`
Отправляем HTTP-запрос и получаем ответ	HTML-строка превращается в дерево тегов	Достаём нужные теги по классам и атрибутам	«1 234,5 ₽» превращается в число 1234.5	Панель «эмитент × день», готовая к анализу

Разделение труда

`requests` достаёт HTML. **`BeautifulSoup`** его разбирает.

BeautifulSoup сама в интернет не ходит — это частая причина недоумения новичков.

Где вы потеряете больше всего времени

Не на шагах 1–2, а на шаге 3: понять, по какому признаку отличить нужный тег от тысячи остальных. Именно этому посвящена основная часть занятия.

Установка: pip install requests beautifulsoup4 lxml cssselect pandas

HTTP: НЕОБХОДИМЫЙ МИНИМУМ

Полный протокол вам не нужен. Нужны один метод, четыре кода ответа и один заголовок — ровно то, что понадобится, когда код упадёт.

GET

Единственный метод, который вам нужен для парсинга: «отдай мне эту страницу». POST понадобится только для форм и авторизации.

Коды ответов, которые вы увидите

200	**OK** Парсим
403	**Forbidden** Обычно не нравится User-Agent — подставьте настоящий
404	**Not Found** Проверьте URL: возможно, сайт сменил структуру
429	**Too Many Requests** Замедлитесь. Вы на грани бана по IP

Заголовок, который решает половину проблем

```
HEADERS = {
    "User-Agent": "Mozilla/5.0 (compatible; academic-research/1.0;
                   +mailto:you@university.edu)",
    "Accept-Language": "ru-RU,ru;q=0.9",
}
resp = requests.get(url, headers=HEADERS, timeout=10)
```

По умолчанию requests представляется как `python-requests/2.31` — для многих сайтов это мгновенный 403.

Контакт в User-Agent — не формальность: администратор скорее напишет вам письмо, чем забанит подсеть университета.

.text или .content? Если на странице кракозябры — отдайте BeautifulSoup сырые байты resp.content, он определит кодировку сам.

HTML: РАЗМЕТКА, ИЗ КОТОРОЙ МЫ ДОСТАЁМ ДАННЫЕ

Страница — это дерево вложенных тегов. Парсинг сводится к одному вопросу: по какому признаку отличить нужный тег от всех остальных.

Как выглядит страница изнутри

```
<div class="news-feed">
  <article class="news-card" data-id="1">
    <a class="news-card__link"
       href="article_1.html">
      <span class="news-card__title">
        Прибыль Сбербанка выросла на 12%
      </span>
    </a>
    <time class="news-card__date"
          datetime="2026-08-23">вчера</time>
    <span class="news-card__ticker">SBER</span>
  </article>
</div>
```

Три слова, которыми мы описываем всё

Тег

`article, a, time, span`

Элемент разметки

Атрибут

`class="news-card", href="…"`

Свойство тега: имя = значение

Дерево (DOM)

`article → a → span`

Вложенность тегов друг в друга

Признаком почти всегда служит class.

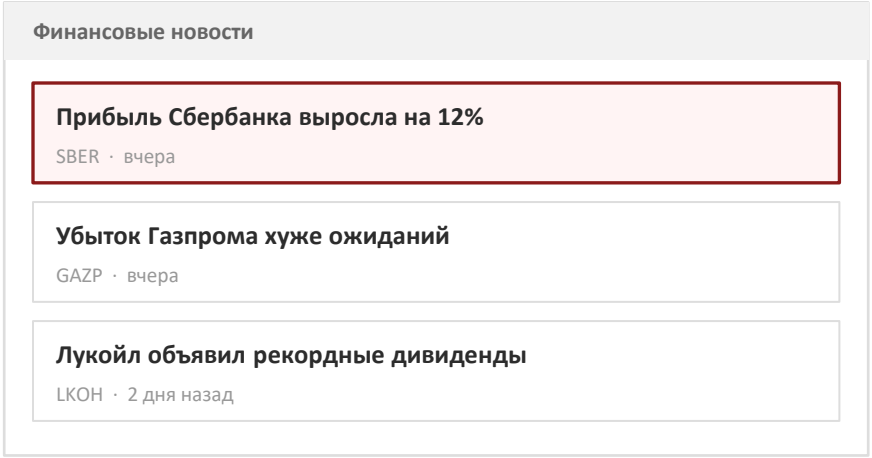
Разработчики называют классы осмысленно — news-card, price, article__body. Это и есть ваша зацепка.

Названия классов — договорённость конкретного сайта, а не стандарт. Поэтому парсер живёт ровно до ближайшего редизайна.

ЧТО ВИДИТЕ ВЫ И ЧТО ВИДИТ PYTHON

Самый практичный навык всего занятия: найти в браузере класс нужного элемента. Инспектор открывается по Ctrl+Shift+C (Cmd+Opt+C на macOS).

Глазами: страница в браузере



Инспектором: та же карточка в коде

```
<article class="news-card" data-id="3">
  <a class="news-card__link" href="/news/3">
    <span class="news-card__title">
      Прибыль Сбербанка выросла на 12%
    </span>
  </a>
  <time class="news-card__date"
        datetime="2026-08-23">вчера</time>
  <span class="news-card__ticker">SBER</span>
</article>
```

Порядок работы: 1) правый клик на нужном элементе → «Исследовать» · 2) в панели читаем class у самого элемента и у его родителя · 3) переносим этот class в find(...) или select(...)

Обратите внимание на <time>: видимый текст «вчера» бесполезен, а в атрибуте datetime лежит готовая машинная дата. Берите атрибут.

АТРИБУТЫ И СЕЛЕКТОРЫ

Два синтаксиса поиска делают одно и то же. Выбор между ними — вопрос вкуса, а не возможностей: find_all нагляднее, select короче для вложенных элементов.

Атрибуты, по которым ищут

`class`	Класс — главная зацепка парсера
`id`	Уникальный идентификатор на странице
`href`	Адрес ссылки в теге <a>
`datetime`	Машинная дата в теге <time>
`data-*`	Произвольные данные разработчика

XPath здесь нет — и не нужен.
Он мощнее, но это отдельный язык. Для 95% исследовательских задач CSS-селекторов хватает с запасом.

CSS-селекторы: шпаргалка

`.news-card`	все элементы с этим классом
`#main`	элемент с id="main"
`article.news-card`	article, у которого есть класс
`div.feed > article`	article — прямой потомок div.feed
`div.feed article`	article где угодно внутри div.feed
`a[href]`	a, у которого есть атрибут href
`td:nth-of-type(4)`	четвёртая ячейка строки

find_all("article", class_="news-card") ≡ select("article.news-card") — один и тот же результат.

REQUESTS + BEAUTIFULSOUP: РАБОЧАЯ СВЯЗКА

Первая библиотека приносит HTML, вторая его разбирает. Всё остальное занятие — вариации этих двух ячеек.

1. requests — достать

```
import requests

resp = requests.get(url, headers=HEADERS,
                    timeout=10)
print(resp.status_code)   # 200
html = resp.content       # сырые байты
```

2. BeautifulSoup — разобрать

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(html, "lxml")
cards = soup.find_all("article",
                      class_="news-card")
title = cards[0].find("span").get_text(strip=True)
```

Паттерн, которым написаны 90% парсеров

```
records = []
for card in soup.find_all("article", class_="news-card"):
    records.append({
        "date":   card.find("time")["datetime"],
        "ticker": card.find("span", class_="news-card__ticker").get_text(strip=True),
        "title":  card.find("span", class_="news-card__title").get_text(strip=True),
    })
df = pd.DataFrame(records)
```

Цикл по карточкам.
Словарь на карточку.
Список словарей в DataFrame.

Запомните эту форму — почти все ваши будущие парсеры будут выглядеть именно так.

BEAUTIFULSOUP: ШПАРГАЛКА

Слева — четыре способа найти один и тот же тег. Справа — всё, что вы будете из него доставать.

Найти элемент: четыре записи, один результат

```
# по тегу
soup.find("time")
# по тегу и классу  (class_ — с подчёркиванием!)
soup.find("time", class_="news-card__date")
# по произвольному атрибуту
soup.find("time", attrs={"datetime": "2026-08-23"})
# CSS-селектором
soup.select_one("time.news-card__date")
```

`find` → один тег или None. **`find_all`** → список (возможно пустой).
select_one / select — то же самое, но на языке CSS.

Достать содержимое

`tag.get_text(strip=True)`
видимый текст без лишних пробелов

`tag["href"]`
значение атрибута (KeyError, если нет)

`tag.get("href")`
то же, но None вместо ошибки

`tag.attrs`
все атрибуты словарём

`tag.name`
имя тега строкой

`tag.find_all("a", limit=5)`
ограничить число результатов

`soup.prettify()`
посмотреть дерево глазами

Всегда указывайте парсер явно: BeautifulSoup(html, "lxml"). Без второго аргумента библиотека выберет его сама — и по-разному на разных машинах.

НАВИГАЦИЯ ПО ДЕРЕВУ: КОГДА КЛАССА НЕТ

Половина реальных страниц устроена так: у нужного значения нет своего класса, но рядом стоит элемент, который опознать легко. Идём от него.

Задача: достать значения из таблицы отчётности

```
<table class="report">
  <tr><td class="label">Выручка</td>       <td>1 240,5</td></tr>
  <tr><td class="label">Чистая прибыль</td><td>318,2</td></tr>
</table>
```

У ячейки со значением нет класса. Но она — следующий сосед ячейки с классом label.

```
for cell in soup.find_all("td", class_="label"):
    value = cell.find_next_sibling("td")
    print(cell.get_text(strip=True), "→", value.get_text(strip=True))
```

```
Выручка → 1 240,5        Чистая прибыль → 318,2
```

Куда можно ходить

`.parent`
вверх на один уровень

`.find_parent("tr")`
вверх до нужного тега

`.children`
вниз, к прямым потомкам

`.find_next_sibling("td")`
вбок, к следующему соседу

`.find_previous_sibling()`
вбок, к предыдущему

`.find_next("span")`
вперёд по всему документу

Приём «найди якорь → сходи к соседу» выручает на страницах отчётности, где значения лежат в безымянных ячейках.

СКВОЗНОЙ КЕЙС: ЛЕНТА НОВОСТЕЙ → ПЕРЕМЕННАЯ

Полный путь за 20 минут: лента → карточки → внутрь статьи → DataFrame → тон → панель для регрессии.

Шаг 1 Собрать ленту

```
cards = soup.find_all(
    "article", class_="news-card")

for c in cards:
    parse_card(c)   # dict на карточку
```

Дата, тикер, заголовок, ссылка.
Проверки на None обязательны:
часть карточек всегда нестандартна.

Шаг 2 Зайти внутрь статьи

```
for url in news["url"]:
    raw = get_html(url, pause=1.0)
    body = parse_article(raw)
```

Заголовка мало — нужен текст.
Пауза между запросами
обязательна (следующий блок).

Шаг 3 Посчитать тон

```
def tone(text):
    w = WORD_RE.findall(text.lower())
    neg = sum(x in NEG for x in w)
    pos = sum(x in POS for x in w)
    return (pos - neg) / len(w)
```

Словарный подход: без ML,
зато прозрачно и воспроизводимо.
Нормируем на длину текста.

Шаг 4 Свернуть в панель

```
panel = (news
    .groupby(["ticker", "date"])
    ["tone"].mean()
    .reset_index())
```

Уровень «эмитент × день» —
именно в таком виде тон
попадает в регрессию.

Результат: panel[ticker, date] → news_tone. Объединяем с котировками — и можно оценивать реакцию доходности на тон новостного дня.

Словарь из 20 слов — учебное упрощение. В работах используют Loughran–McDonald или FinBERT, но конвейер сбора остаётся тем же.

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ: ДОЧИСТКА РЕЗУЛЬТАТА

Regex нужен не для разбора HTML, а для того, чтобы вытащить структуру из уже добытого текста. Это последние 10% работы — но без них данные в модель не положишь.

Четыре шаблона, покрывающих почти всё

проценты

```
r"\d+[.,]?\d*\s*%"
```

`12,4%  24,8%`

суммы

```
r"\d[\d\s\xa0]*[.,]?\d*\s*(млрд|млн)?\s*руб"
```

`1 234,5 млрд руб`

даты

```
r"\b\d{2}\.\d{2}\.\d{4}\b"
```

`24.08.2026`

тикеры

```
r"\b[A-Z]{4,5}\b"
```

`SBER  SBERP`

Функция, которая понадобится всегда

```
NUM = re.compile(r"-?\d[\d\s]*(?:[.,]\d+)?")

def to_float(s):
    m = NUM.search(str(s).replace("\xa0"," "))
    if not m: return None
    return float(m.group(0)
           .replace(" ","").replace(",","."))
```

```
"1 234,5"        → 1234.5
"24,8 %"         → 24.8
"80,4419 руб."   → 80.4419
"—"              → None
```

Мы ищем число, а не вычищаем из строки всё лишнее: второй подход ломается на «руб.» — точка остаётся в конце, и float() падает.

Захотелось распарсить HTML регуляркой — остановитесь. HTML не является регулярным языком: вложенность ломает любой шаблон.

ПОЧЕМУ САЙТЫ НЕ ЛЮБЯТ, КОГДА ИХ ПАРСЯТ

У сайтов есть на то основания: ваш скрипт создаёт нагрузку, за которую платит владелец сервера. Понимать, что именно вас выдаёт, полезнее, чем знать способы обхода.

Шесть признаков, по которым вас опознают как бота

Частота	50 запросов в секунду с одного IP	→ Пауза 1–3 сек
User-Agent	python-requests/2.31.0 в логах	→ Настоящий UA + контакт
Нет сессии	Каждый запрос без cookie, как первый визит	→ requests.Session()
Идеальный ритм	Запрос ровно каждые 1000 мс — люди так не кликают	→ Случайная добавка
Игнор robots.txt	Заход в явно закрытые разделы	→ Проверять до сбора
Реакция на 429	Продолжает «стучаться» после отказа сервера	→ Удлинение пауз между попытками

Шкала последствий

429
Временное замедление

▼

403
Ваш User-Agent закрыт

▼

бан IP
Часы или сутки без доступа

▼

бан подсети
Без доступа вся сеть вуза/предприятия

Последний пункт — не гипотеза. Заблокируют не вас лично, а адрес, с которого вы работаете: то есть всех коллег.

ВЕЖЛИВЫЙ ПАРСИНГ: ЧЕТЫРЕ ПРАВИЛА

Исследовательский сбор почти никогда не конфликтует с сайтом, если соблюдать эти четыре правила. Нарушив их, вы теряете доступ ровно перед дедлайном.

01 Читать robots.txt

```
from urllib.robotparser import RobotFileParser

rp = RobotFileParser()
rp.set_url(base + "/robots.txt"); rp.read()
rp.can_fetch("my-bot", url)   # можно?
rp.crawl_delay("my-bot")      # сколько ждать
```

Не закон, а соглашение. Но нарушение вместе с Terms of Service — стандартный аргумент в споре о неправомерном доступе.

02 Сессия и backoff

```
retry = Retry(total=5, backoff_factor=2,
    status_forcelist=[429, 500, 502, 503],
    respect_retry_after_header=True)
s = requests.Session()
s.mount("https://", HTTPAdapter(max_retries=retry))
```

Паузы 2, 4, 8, 16, 32 секунды. Если сервер прислал Retry-After — ждём ровно столько, сколько он просит.

03 Пауза со случайной добавкой

```
def polite_sleep(base=1.0, jitter=0.5):
    time.sleep(base + random.uniform(0, jitter))
```

Ровно одинаковые интервалы — характерная подпись бота. Если в robots.txt задан Crawl-delay, соблюдайте его.

04 Кэшировать: каждую страницу один раз

```
if path.exists():
    return path.read_bytes()      # из кэша
r = session.get(url); path.write_bytes(r.content)
```

Самая частая причина бана — отладка: скрипт запускают двадцать раз, и каждый прогон качает всё заново.

Чего мы не делаем: прокси-ротация, обход CAPTCHA, подмена отпечатка браузера. Не хочет отдавать данные — пишите владельцу, а не обходите защиту.

ПРАВО, ЭТИКА И ВОСПРОИЗВОДИМОСТЬ

Три вопроса, которые задаст рецензент, и на которые лучше иметь ответ до начала сбора.

Можно ли было собирать эти данные?

Открытый доступ ≠ разрешение на сбор. Смотрим три источника: robots.txt, Terms of Service сайта и режим самих данных. Персональные данные в РФ — ФЗ-152, в ЕС — GDPR: агрегируйте и обезличивайте до того, как положите в датасет.

Не навредил ли сбор самому источнику?

Нагрузка, которую вы создали, — часть исследовательской этики, а не только техники. Один запрос в секунду с паузами и кэшем не заметит никто. Сто параллельных потоков заметят все, включая юристов сайта.

Сможет ли кто-то повторить результат?

Сайт сменит вёрстку или удалит архив — и через год вы не воспроизведёте собственную таблицу. Сохранённый сырой HTML и есть ваши первичные данные: храните папку с кэшем вместе с кодом и указывайте дату сбора.

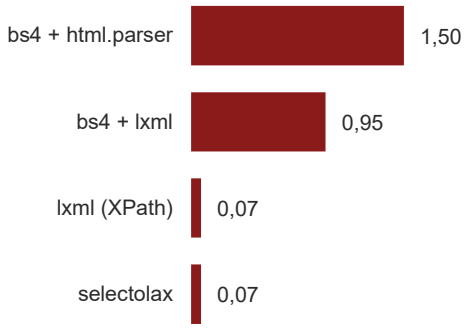
Практический совет: указывайте в User-Agent рабочую почту. Известны случаи, когда после письма администратора исследователю просто выгружали нужный датасет напрямую — это быстрее и для него, и для сайта.

АЛЬТЕРНАТИВЫ BEAUTIFULSOUP

BeautifulSoup — не единственный и не самый быстрый парсер. Он самый популярный, и стоит понимать, за счёт чего.

БИБЛИОТЕКА	СИЛЬНАЯ СТОРОНА	ПОЧЕМУ НЕ СТАЛА СТАНДАРТОМ
BeautifulSoup	Читаемый API, прощает кривой HTML, огромное сообщество и большое число примеров	Медленная: Python-обёртка поверх парсера
lxml	Очень быстрая (C), полноценный XPath	XPath — отдельный язык; невнятные ошибки; жёстче к «грязному» HTML
selectolax	Самая быстрая из распространённых (движок lexbor)	Только CSS, без XPath; маленькое сообщество пользователей
parsel	CSS и XPath в одном API, ядро Scrapy	Заточена под Scrapy, вне его экосистемы почти не встречается
pandas.read_html	Таблица → DataFrame одной строкой	Только настоящие <table>; ничего, кроме таблиц
html.parser	Ноль зависимостей, стандартная библиотека	Низкоуровневый: события вместо дерева, писать вручную долго

ЗАМЕРЕНО НА ДОКУМЕНТЕ 1,6 МБ



Разница в 20 раз.
На тысяче страниц она экономит секунды — а вежливые паузы между запросами займут 17 минут.

Исследователь пишет парсер один раз на один датасет: узкое место — ваши часы на отладку, а не секунды процессора. Порог переключения — десятки тысяч страниц.

ГДЕ ЗАКАНЧИВАЕТСЯ BEAUTIFULSOUP

Четыре границы, о которые вы обязательно споткнётесь. Первая — главная.

1. Она не выполняет JavaScript

BeautifulSoup видит только тот HTML, который сервер прислал в ответ на запрос. Если данные рисует скрипт уже в браузере — в исходнике их нет.

```
<div id="quotes">Загрузка...</div>
<script>
  fetch("/api/quotes").then(r => r.json())
     .then(d => render(d));
</script>
>>> soup.find(id="quotes").get_text()
'Загрузка...'      # ← всё, что вы получите
```

Как понять за 10 секунд, что страница – динамическая

Ctrl+U — исходный код страницы. **Ctrl+Shift+C** — инспектор.
Данных нет в первом, но есть во втором → страница динамическая.

Тогда: вкладка Network → ищем XHR-запрос. Часто это готовый JSON, и парсер вообще не нужен. Если нет — Selenium или Playwright, это отдельные лекции.

2. Она не делает запросов

Ни авторизации, ни cookie, ни повторов при ошибке. Это зона ответственности requests: сессии, заголовки, backoff.

3. Она медленная на объёмах

Смягчается выбором парсера: BeautifulSoup(html, "lxml") в 1,6 раза быстрее html.parser при том же API. Всегда указывайте парсер явно.

4. Она не знает смысла страницы

Селектор span.news-card__title живёт до ближайшего редизайна. Парсер — не вечный код: закладывайте, что через полгода его придётся чинить.

Из четырёх ограничений три обходятся аккуратным кодом. Первое не обходится вовсе — оно определяет, каким инструментом решается задача. И про преодоление этого ограничения будет отдельная лекция.

ГРАБЛИ: ГДЕ ВЫ ПОТЕРЯЕТЕ ВРЕМЯ

Примеры ошибок, с которыми можно провозиться долго

find вернул None

```
AttributeError: 'NoneType'
object has no attribute 'text'
```

```
el = soup.find(...)
if el: el.get_text(strip=True)
```

На реальной странице часть карточек всегда нестандартна — реклама, анонс, спецпроект.

class — ключевое слово

```
SyntaxError:
find("a", class="news")
```

```
find("a", class_="news")
find("a", attrs={"class":
"news"})
```

Подчёркивание в конце — не опечатка в туториале, а обязательная часть синтаксиса.

Несколько классов сразу

```
class="card card--hot card--big"
find(class_="card card--hot") →
None
```

```
select_one("div.card.card--hot")
```

В атрибуте class хранится список. bs4 сверяет строку целиком — CSS-селектор надёжнее.

.text у списка

```
ResultSet object has
no attribute "text"
```

```
for c in cards:
    c.get_text(strip=True)
```

find_all возвращает список, а не тег.

Кодировка и \xa0

```
float('1\xa0234.5') → ValueError
Прибыль → ÐÑÐ¸Ð±ÑÐ»Ñ
```

```
BeautifulSoup(resp.content,
"lxml")
to_float(s)   # см. блок regex
```

Неразрывный пробел выглядит как обычный, но ломает astype(float). Проверяйте первым делом.

Бонус: BeautifulSoup(html) без второго аргумента берёт тот парсер, который найдёт — и на машине у другого человека это может быть другой парсер.

ИТОГИ

requests достаёт, BeautifulSoup разбирает

Главное – не путать, кто за что отвечает

Парсинг сводится к поиску признака

Найти class в инспекторе важнее, чем помнить синтаксис find_all

Цикл → словарь → DataFrame

Одна форма, которой написаны почти все ваши будущие парсеры

Пауза, сессия, кэш

Про эти вещи помним, если не хотим словить бан по IP

Сырой HTML — это первичные данные

Без него результат невоспроизводим уже через полгода

BS4 не выполняет JavaScript

Для этого нужны другие инструменты, про которые поговорим позже

РЕСУРСЫ И ЧТО ДАЛЬШЕ

ДОКУМЕНТАЦИЯ

BeautifulSoup
`beautiful-soup-4.readthedocs.io`

Requests
`requests.readthedocs.io`

selectolax
`github.com/rushter/selectolax`

Словари Loughran–McDonald
`sraf.nd.edu`

РАБОТЫ, ГДЕ ДАННЫЕ СОБРАНЫ СКРЕЙПИНГОМ

Da, Engelberg, Gao (2015)

The Sum of All FEARS. RFS 28(1), 1–32

Bartov, Faurel, Mohanram (2018)

Can Twitter Help Predict Firm-Level Earnings? TAR 93(3), 25–57

Green, Huang, Wen, Zhou (2019)

Crowdsourced Employer Reviews. JFE 134(1), 236–251

Cookson, Niessner (2020)

Why Don’t We Agree? JF 75(1)

Cookson, Engelberg, Mullins (2023)

Echo Chambers. RFS 36(2), 450–500

Спасибо за внимание