Содержание

[1. Что такое XML и где он используется 3](#_Toc211605439)

[2. Пространства имён в XML: назначение, объявление, влияние на выборки 5](#_Toc211605440)

[3. Схемы и валидация: DTD и XSD 6](#_Toc211605441)

[4. XPath: адресация узлов, фильтры, позиционные выборки, практическое применение 7](#_Toc211605442)

[5. HTML и подготовка к BeautifulSoup: особенности разметки, отбор селекторов, обработка значений 8](#_Toc211605443)

1. Что такое XML и где он используется

XML — текстовый формат для представления структурированных данных в виде иерархии. У каждого документа есть один корневой элемент, внутри которого располагаются вложенные элементы, текстовые узлы и атрибуты. Элемент описывает сущность предметной области, а атрибут закрепляет компактные признаки или вспомогательные метаданные. Такое разбиение делает документ предсказуемым: структуру можно читать человеком, а программу можно научить извлекать нужные поля одними и теми же правилами.

Синтаксис XML требует дисциплины. Пары тегов обязаны корректно закрываться, вложенность не должна нарушаться, атрибуты берутся в кавычки, специальные символы в тексте экранируются сущностями. Для включения «сырого» содержимого, где встречаются угловые скобки и амперсанд, используется раздел CDATA. Комментарии допускаются, но логика данных на них не строится. Эта строгость обеспечивает одинаковое поведение парсеров в разных средах и снижает вероятность неоднозначностей при интеграции.

В повседневной работе различают два уровня корректности: благополучие (well-formedness) и валидность (validity). Благополучный документ соответствует синтаксису и разбирается в дерево. Валидный документ, помимо этого, удовлетворяет формальной схеме, где описаны состав, порядок и типы элементов. На входе рабочих конвейеров сначала проверяют благополучие, затем применяют валидацию по схеме, а только после этого запускают преобразования и загрузку. Такой порядок предотвращает накопление ошибок, связанных с неверными типами и отсутствующими обязательными полями.

XML применяют для обмена между системами в сценариях, где важна прозрачная структура: каталоги и прайс-листы, расписания, отчёты, конфигурации сервисов, сообщения интеграционных шины. Формат удобен для длинного жизненного цикла: его можно версионировать, хранить примеры «эталонных» файлов, проверять соответствие схемам, документировать изменения. Основные риски связаны с человеческими ошибками: несоответствие заявленной и фактической кодировок, неэкранированные специальные символы, пропущенные закрывающие теги, дубли атрибутов, путаница с пространствами имён. Для снижения рисков используют набор простых практик: фиксируют единую кодировку (обычно UTF-8), сохраняют эталонный образец, включают проверку по схеме в процесс приёмки, явно указывают пространства имён и используют устойчивые адреса к данным.

С точки зрения извлечения, XML удобен тем, что хорошо сочетается с XPath. Адреса к элементам и атрибутам записываются компактно и однозначно, что облегчает и разработку, и тестирование. Если документ использует пространства имён, адреса формулируют с префиксами. Это снимает двусмысленность и обеспечивает воспроизводимость выборок при переносе между инструментами и средами.

from lxml import etree

def main() -> None:

    # Минимальный, самодостаточный XML с пространством имён ns

    xml = """<?xml version="1.0" encoding="UTF-8"?>

<catalog xmlns:ns="http://ex.com/ns">

  <ns:item id="1"><ns:price currency="RUB">490.00</ns:price></ns:item>

  <ns:item id="2"><ns:price currency="RUB">120.00</ns:price></ns:item>

</catalog>"""

    # Строим дерево; важна корректная кодировка

    root = etree.fromstring(xml.encode("utf-8"))

    # Карта префиксов для XPath — без неё выборки по ns:\* будут пустыми

    ns = {"ns": "http://ex.com/ns"}

    # Все id элементов ns:item

    ids = root.xpath("//ns:item/@id", namespaces=ns)

    # Все цены в рублях, которые строго больше 200

    rub\_prices\_over\_200 = [

        float(x) for x in root.xpath(

            "//ns:price[@currency='RUB' and number(text())>200]/text()", namespaces=ns

        )

    ]

    print("IDs:", ids)

    print("RUB > 200:", rub\_prices\_over\_200)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition) [Электронный ресурс]. Доступно: https://www.w3.org/TR/xml

2. Пространства имён в XML: назначение, объявление, влияние на выборки

Пространства имён решают коллизии одноимённых тегов из разных словарей. Каждому пространству соответствует URI, а в документе с ним связывают префикс. Элемент и атрибуты, помеченные префиксом, интерпретируются как принадлежащие данному словарю. В корневом элементе можно объявить несколько пространств для разных частей документа. Также встречается дефолтное пространство имён, когда элементы без префикса автоматически относятся к объявленному URI.

Наличие пространств влияет на адресацию. В XPath-выражениях недостаточно указать только локальное имя тега: нужно задействовать префикс, а парсеру передать словарь соответствий «префикс → URI». Если этого не сделать, запросы будут возвращать пустой результат, даже когда нужные элементы присутствуют. Такая ситуация часто воспринимается как «парсер не видит данные», хотя причина в отсутствии контекста пространств в самом запросе.

Правильная работа с пространствами начинается с систематизации. В проекте фиксируют, какие префиксы используются и каким URI они соответствуют. При изменениях поставщик уведомляет потребителей, а тестовые примеры обновляются. В коде словарь пространств передают явно и централизованно, чтобы не дублировать строки и не расходиться в трактовках. Если документ использует дефолтное пространство, это учитывают при построении адресов и не опираются на «голые» имена тегов без префиксов. Такой подход убирает «слепые пятна» и позволяет уверенно применять XPath к любым участкам документа, независимо от глубины и сложности вложенности.

Отдельный нюанс касается атрибутов. По умолчанию атрибуты считаются безымянными (без пространства) и адресуются без префикса. Однако в некоторых схемах встречаются атрибуты из пространств имён; в таких случаях они также помечаются префиксом и адресуются с учётом словаря. Это редкий случай, но его следует учитывать при переносе готовых выражений между проектами.

# 3. Схемы и валидация: DTD и XSD

Схема формализует структуру XML-документа и делает формат проверяемым. DTD описывает состав и порядок элементов, предоставляет базовые возможности определения сущностей, но практически не работает с типами и ограниченно взаимодействует с пространствами имён. XSD, в отличие от DTD, построена на самом XML и предназначена для строгой типизации. В XSD указывают, что является строкой, числом, датой, логическим значением, задают ограничения по диапазонам, длинам, шаблонам и перечислениям, определяют обязательность элементов и их кратность. Поддержка пространств имён и композиции сложных типов делает XSD удобной для современных интеграций.

Валидация по XSD включается в процесс приёмки. Документ сначала проверяется на благополучие, затем валидируется по схеме. Типичные ошибки выявляются сразу: символы валюты в числе, строка вместо даты, пропущенный обязательный элемент, неверный порядок, значение вне перечисления допустимых. По результатам формируется отчёт, а документ с нарушениями отклоняется. Это предотвращает дальнейшие затраты на попытки «лечить» входные данные на поздних этапах и снижает вероятность тихих искажений в аналитике.

Версионность схемы — часть эксплуатационной практики. При изменениях формата выпускают новую версию XSD, сопровождают заметками по миграции, обновляют эталонные примеры и тесты. На стороне потребителя поддерживается каталог схем с привязкой к версиям поставки. В автоматизации приёмки настройка указывает, по какой версии схемы следует проверять каждый поток. Это исключает случайные несовместимости и облегчает параллельную поддержку нескольких источников.

Связь со средой выполнения проста: XSD, эталонный пример и проверка в пайплайне хранятся рядом с кодом извлечения. В unit-тестах подтверждают, что документы «проходят» схему и что XPath-адреса, от которых зависит извлечение, остаются валидными. Такой комплект позволяет сдерживать эффект от изменений на стороне источника и быстро диагностировать, в каком именно слое возникло нарушение — в синтаксисе, в структуре по схеме или в логике адресации.

W3C. XML Schema Part 2: Datatypes (Second Edition) [Электронный ресурс]. Доступно: https://www.w3.org/TR/xmlschema-2/

# 4. XPath: адресация узлов, фильтры, позиционные выборки, практическое применение

XPath — язык выражений для адресации узлов в XML-дереве. Путь строится от контекста и определяет последовательность шагов к нужным элементам, их атрибутам и текстовым значениям. Служебные символы и функции позволяют применять фильтры по значению и позиции, нормализовать пробелы, вычислять числовые выражения и выбирать соседние элементы. Конструкции // и / отличают поиск по всему дереву от движения по строгому пути. Адрес к атрибуту строится через @, а извлечение текста — через text() или прямое обращение к содержимому.

Фильтры уточняют выборки. Предикаты по атрибутам позволяют сузить набор элементов до тех, что обладают нужным признаком, например требуемой валютой. Позиционные выражения ограничивают выбор до первого или первых N элементов. Для работы с человеческим текстом применяется нормализация пробелов и поиск подстроки. Это даёт средство устойчиво выбирать нужный блок даже в длинных документах, где соседние разделы похожи друг на друга.

Практическая роль XPath — устранить двусмысленность в требованиях и тестах. Там, где ранее фигурировал словесный указатель («первая цена в рублях»), теперь присутствует выражение, которое однозначно понимают и люди, и программа. Такое выражение повторно применимо и легко переносится: его можно использовать в проверках соответствия во время приёмки, в скриптах прототипирования и в коде извлечения. Если документ использует пространства имён, выражение сохраняет вид с префиксами, а в исполняющий код добавляют словарь соответствий «префикс → URI».

В рабочих проектах выражения XPath располагают рядом с обработчиками и сопровождают краткими комментариями о назначении и допущениях. При изменениях формата обновляют и схему, и выражения, а минимальный набор тестовых документов позволяет убедиться, что корректировки не приводят к регрессиям в нецелевых местах. Такой подход обеспечивает стабильность и ускоряет диагностику в случае расхождений между документацией и фактическим содержимым.

W3C. XML Path Language (XPath) 3.1 [Электронный ресурс]. Доступно: https://www.w3.org/TR/xpath-31/

# 5. HTML и подготовка к BeautifulSoup: особенности разметки, отбор селекторов, обработка значений

HTML — разметка веб-страниц, которая допускает неполноту и ошибки в структуре. Парсеры и браузеры «лечат» недостающие закрывающие теги, исправляют вложенность и формируют DOM-дерево даже при дефектах исходника. Это упрощает извлечение: почти любую страницу можно разобрать. Но это увеличивает чувствительность к мелким изменениям верстки: переименование класса, появление дополнительного контейнера, перестановка блоков. В условиях изменений предпочтительнее опираться на признаки, которые сохраняются дольше: стабильные классы, идентификаторы, атрибуты data-\*, закономерное соседство с заголовками разделов. Для адресации в HTML чаще применяют CSS-селекторы, поскольку они короче и привычнее, но XPath остаётся полезным для сложной структурной навигации и выборок «по соседству».

from bs4 import BeautifulSoup

import re

def main() -> None:

    # Упрощённый фрагмент карточки товара; близок к реальному HTML

    html = """

<div class="card">

  <a class="title" href="/p/1">Товар A</a>

  <img src="/img/a.jpg" alt="">

  <span class="price">490 ₽</span>

</div>

"""

    # Для прототипов подойдёт "html.parser"; для скорости используйте "lxml"

    soup = BeautifulSoup(html, "html.parser")

    # Название — текст ссылки в .card .title

    name = soup.select\_one(".card .title").get\_text(strip=True)

    # Цена — чистим всё, кроме цифр, приводим к int

    price\_text = soup.select\_one(".price").get\_text()

    price = int(re.sub(r"\D+", "", price\_text))

    # Картинка — берём значение src у <img>

    image = soup.select\_one(".card img")["src"]

    print("Name:", name)

    print("Price:", price)

    print("Image:", image)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

BeautifulSoup — библиотека Python для разборки HTML и XML в удобное дерево с простыми методами поиска. Она поддерживает несколько движков: встроенный html.parser (без зависимостей, умеренная скорость, терпимость к дефектам), lxml (высокая производительность, поддержка и HTML, и XML, хорошая стыковка с XPath), html5lib (максимальная совместимость с поведением браузеров при меньшей скорости). Выбор движка зависит от качества исходной разметки и производительности, которая требуется в задаче. В типичных сценариях применяют lxml из-за баланса скорости и качества.

Базовые операции в BeautifulSoup включают поиск по имени тега и атрибутам, выбор по CSS-селектору, получение одного узла, навигацию к соседним и родительским элементам, извлечение текста и значений атрибутов. Важный прикладной шаг — нормализация значений перед использованием: удаление нецифровых символов из цен, приведение к целому или вещественному типу, уплотнение пробелов в названиях, преобразование дат к единому формату. Ещё один аспект — кодировка: при работе с сетевыми источниками проверяют заголовки сервера, при сомнениях переключаются на lxml и перехватывают ошибки декодирования, чтобы не терять документ целиком из-за единичных символов.

В устойчивых реализациях селекторы делают «короче и умнее». Вместо длинных цепочек с позиционными шагами предпочитают признаки смысловой привязки: имена классов, подстроки в ссылках, уникальные атрибуты. Селекторы версионируют в комментариях, а на извлекаемых участках закладывают запасные выражения для случаев, когда основной признак временно отсутствует. Ошибки отсутствия узла обрабатывают явно, чтобы сообщать о пропущенном элементе, а не падать по исключению. Для повторяемости используют небольшие фикстуры: несколько «версий» разметки, на которых можно быстро проверить, не сломался ли извлекатель после изменений на стороне источника.

Если у источника есть официальный API или XML/JSON-фид, он предпочтительнее, чем парсинг HTML. Формализованный интерфейс стабильнее и проще тестируется, а код извлечения становится короче и прозрачнее. Парсинг HTML применяют, когда альтернативы отсутствуют или витрина содержит единственные данные. В таких случаях перечисленные практики позволяют удерживать устойчивость на приемлемом уровне и контролировать эффект от небольших правок в интерфейсе.

from bs4 import BeautifulSoup

def main() -> None:

    html = """

<table id="prices">

  <tr><th>Товар</th><th>Цена</th></tr>

  <tr><td>А</td><td>490</td></tr>

  <tr><td>B</td><td>120</td></tr>

</table>

"""

    # "lxml" хорошо справляется с неидеальной разметкой

    soup = BeautifulSoup(html, "lxml")

    rows = []

    # Пропускаем строку заголовков (первая строка <tr>)

    for tr in soup.select("#prices tr")[1:]:

        cols = [td.get\_text(strip=True) for td in tr.select("td")]

        if len(cols) != 2:

            # На случай неожиданных строк — защитный барьер

            continue

        name, price\_text = cols

        try:

            price = int(price\_text)

        except ValueError:

            # Если в цене мусор — пропускаем строку

            continue

        rows.append({"name": name, "price": price})

    print(rows)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Рэй, Э. Т. Изучаем XML. — СПб.: Символ-Плюс, 2001. — 408 с. — ISBN отсутствует. Просмотр в Google Books

Richardson, L. Beautiful Soup Documentation (bs4) [Электронный ресурс]. Доступно: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Свейгарт, Э. Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих. — М.: Вильямс, 2017.

Mozilla Foundation. MDN Web Docs: CSS-селекторы [Электронный ресурс]. Доступно: https://developer.mozilla.org/ru/docs/Web/CSS/CSS\_selectors