

Методичка по Xpath/XPath 2

(практическое руководство для парсинга СМИ и социальных медиа)

Создано Балдыновым И. О., 2025

Содержание:

1. Операторы и Функции:

Операторы:

- div
- mod
- to

Строковые функции:

- codepoints-to-string()
- compare()
- ends-with()
- lower-case()
- normalize-space()
- replace()
- starts-with()
- string-join()
- string-length()
- string-to-codepoints()
- tokenize()
- translate()
- upper-case()

Логические функции:

- boolean()
- matches()
- not()
- exists()

Функции для работы с последовательностями:

- distinct-values()
- position()
- reverse()
- subsequence()

Функции для работы с числами:

- number()
- floor()
- ceiling()
- round()

Функции для работы с датами:

- current-date()
- day-from-date()
- month-from-date()
- year-from-date()

2. [Условная конструкция if-then-else](#)
3. [Итерация for](#)
4. [Шаблоны и другие возможности XPath](#)

Операторы:

div выполняет деление чисел с получением дробного результата ($5 \text{ div } 2 = 2.5$), в отличие от схожего оператора **idiv**, который возвращает целое число округляя вниз ($5 \text{ idiv } 2 = 2$). При работе со строками **div** используется для вычисления длины строки и определения позиций символов, например, при разбиении строки на части.

Пример использования со строками:

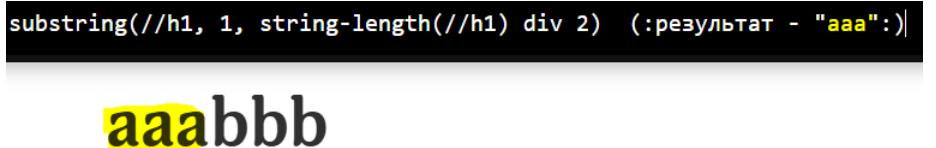
При делении строки пополам важно учитывать длину строки, т.к. четная/нечетная длина выдаст, по умолчанию, разный результат (Рис.1, Рис.2):

Первая половина строки:

Длина строки четная (строка делится ровно пополам), первые 3 символа "aaa" (рис.1)

```
substring(//h1, 1, string-length(//h1) div 2)
```

```
substring(//h1, 1, string-length(//h1) div 2) (:результат - "aaa":)
```



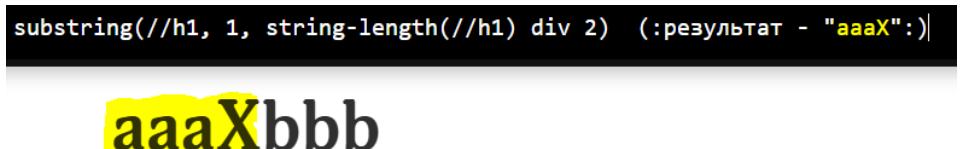
aaabb

[В МИРЕ](#) 3 часа назад [Обсудить в телеграм](#)

Рисунок 1 Длина строки четная.

С четной строки (рис.1) брать большую половину ("aaab"): **substring(//h1, 1, string-length(//h1) div 2 + 1)**

```
substring(//h1, 1, string-length(//h1) div 2) (:результат - "aaaXbbb":)
```



aaaXbbb

[В МИРЕ](#) 3 часа назад [Обсудить в телеграм](#)

Рисунок 2 Длина строки нечетная

Если строка нечетная (Рис. 2) , то при делении пополам, по умолчанию, округляется в большую сторону.

Для того, чтобы с нечетной строки (Рис.2) брать меньшую половину ("aaa") можно воспользоваться функцией **floor()** округляющей в меньшую сторону: **substring(//h1, 1, floor(string-length(//h1) div 2))**

Вторая половина строки (рис.3, рис.4):

```
substring(//h1, string-length(//h1) div 2) (рис.3)
```

```
substring(//h1, string-length(//h1) div 2) (:Результат: "Xbbb":)
```

aaa**X**bbb

Рисунок 3 Средний символ + вторая половина строки

```
substring(//h1, string-length(//h1) div 2 + 1) (рис.4)
```

```
substring(//h1, string-length(//h1) div 2 + 1) (:Результат: "bbb":)
```

aaa**X**bbb

[В МИРЕ](#) 3 часа назад [Обсудить в телеграм](#)

Рисунок 4 Вторая половина строки

Средний символ строки:

```
substring(//h1, string-length(//h1) div 2, 1) (:Результат: "X":)
```

```
substring(//h1, string-length(//h1) div 2, 1) (:Результат: "X":)
```

aaa**X**bbb

[В МИРЕ](#) 3 часа назад [Обсудить в телеграм](#)

Рисунок 5 Средний символ строки.

mod — арифметический оператор для вычисления остатка от деления. Применяется для фильтрации, проверки кратности, создания шаблонов и циклических структур.

mod полезен при выборке ссылок, заголовков или иных важных составляющих статьи, например необходимо извлечь только четные или нечетные ссылки:

(//a[h3])[position() mod 2 = 0] – четные ссылки (Рис. 6)

(//a[h3])[position() mod 2 = 0]

The screenshot shows a news website with a navigation bar at the top. Below the navigation bar, there are two columns of news items. The left column is for March 6, and the right column is for March 5. Each news item is a yellow box containing a timestamp and a headline. To the right of the news items is a sidebar with various advertisements.

Новости	Эксклюзивы	Политика	Экономика	Происшествия	Спорт	МК ТВ
ПОКАЗАТЬ ПО ДАТЕ						
6 МАРТА 2025						
01:29	Кадыров рассказал об успехах ВС РФ в Часовом Яре					
01:17	Ирландия собирается отправить спецназ на Украину					
01:12	Василий Небензя напомнил в ООН о значении 80-летия Победы над нацизмом					
00:56	У россиян появится право на повторное получение бесплатного участка					
00:46	Макрон созывает в Париже весь генералитет Европы ради «безопасности» Зеленского					
00:27	Режиссер Андреасян объявил, за что ненавидит Тарковского					
00:01	Вслед за «Русским домом» из Баку прогоняют волонтеров Красного Креста					
00:01	Госдума предложила объявить амнистию к 80-летию Победы в ВОВ					
5 МАРТА 2025						
23:46	Минфин вводит новые правила определения налогового резидентства					
23:30	Пушков уличил Стармера и Макрона в манипуляции Зеленским ради продолжения войны					

Рисунок 6 Четные ссылки

(//a[h3])[position() mod 2 = 1] – нечетные ссылки (Рис. 7)

(//a[h3])[position() mod 2 = 1]

The screenshot shows a news website with a navigation bar at the top. Below the navigation bar, there are two columns of news items. The left column is for March 6, and the right column is for March 5. Each news item is a yellow box containing a timestamp and a headline. To the right of the news items is a sidebar with various advertisements.

НОВОСТИ	ЭКСКЛЮЗИВЫ	ПОЛИТИКА	ЭКОНОМИКА	ПРОИСШЕСТВИЯ	СПОРТ	МК ТВ
ПОКАЗАТЬ ПО ДАТЕ						
6 МАРТА 2025						
01:29	Кадыров рассказал об успехах ВС РФ в Часовом Яре					
01:17	Ирландия собирается отправить спецназ на Украину					
01:12	Василий Небензя напомнил в ООН о значении 80-летия Победы над нацизмом					
00:56	У россиян появится право на повторное получение бесплатного участка					
00:46	Макрон созывает в Париже весь генералитет Европы ради «безопасности» Зеленского					
00:27	Режиссер Андреасян объявил, за что ненавидит Тарковского					
00:01	Вслед за «Русским домом» из Баку прогоняют волонтеров Красного Креста					
00:01	Госдума предложила объявить амнистию к 80-летию Победы в ВОВ					
5 МАРТА 2025						
23:46	Минфин вводит новые правила определения налогового резидентства					
23:30	Пушков уличил Стармера и Макрона в манипуляции Зеленским ради продолжения войны					

Рисунок 7 Нечетные ссылки

Также приведем несколько примеров выборки:

Найти все числа, кратные 5:

(1 to 50)[. mod 5 = 0]

Результат: (5, 10, 15, 20, 25, 30, 35, 40, 45, 50)

Выбрать каждый третий элемент из списка узлов:

//li[position() mod 3 = 0]

Результат: Возвращает все элементы, расположенные на 3-й, 6-й, 9-й и т.д. позициях.

Является ли год високосным:

for \$year in (2000 to 2024) return if (\$year mod 4 = 0 and (\$year mod 100 != 0 or \$year mod 400 = 0)) then concat(\$year, " Високосный") else concat(\$year, " Не Високосный")

to – это диапазонный оператор, который создает последовательности чисел, например:

(1 to 5) -> результат (1, 2, 3, 4, 5)

sum(1 to 10) -> Сумма всех чисел от 1 до 10

for \$i in (1 to 5) return \$i * \$i -> Результат: (1, 4, 9, 16, 25)

Калькулятор: for \$i in (1 to 10) return concat(3, " x ", \$i, " = ", 3 * \$i) -> результат: 3 x 1 = 3, 3 x 2 = 6 и т.д.

if (string-length("//h1) = (1 to 10)) then "Да" else "Нет" -> соответствие условию (если длина строки внутри //h1 находится в диапазоне от 1 до 10 включительно, то результат — 'Да', иначе — 'Нет')

Оператор to часто используется в сочетании с функцией position(), для создания диапазона значений, позволяющего задавать интервалы элементов. Например, брать ссылки с нужной позиции:

Примеры:

1. Выбор первых 5 элементов:

(//a[@class="link"])[position() = 1 to 5] -> Выбирает ссылки: 1, 2, 3, 4, 5.

2. Выбор элементов с 3-го по 7-й:

(//div)[position() = 3 to 7] -> Выбирает <div> на позициях 3, 4, 5, 6, 7.

3. Выбор последних 4 элементов (если всего 10):

//span[position() = 7 to 10] -> Выбирает на позициях 7, 8, 9, 10.

4. Выбор первых 5 заголовков h2, но только если их длина больше 10:

//h2[position() = 1 to 5][string-length() > 10] -> Выбирает <h2> с длиной текста > 10, если их позиция от 1 до 5.

ФУНКЦИИ:

compare() используется для сравнения двух значений и возвращает:

-1, если первое значение меньше второго compare("apple", "avocado");

0, если они равны compare("apple", "apple"),

1, если первое значение больше второго compare("avocado", "apple") .

Пример 1: Фильтрация по диапазону строк, выберем строки в алфавитном диапазоне от "А" до "П" (в диапазон не попадает сама "П")

```
//table[@class="standard"]//td/a[compare(@title, "A") >= 0 and compare(@title, "П") <= 0]
```

(также можно решить с помощью substring()):

```
//table[@class="standard"]//td/a[substring(@title, 1, 1) >= "A" and substring(@title, 1, 1) <= "П"] – в диапазон попадает "П")
```

Ранг	Язык	Родной	Второй	Общее число говорящих на нём
1	Китайский язык ^[10]	1,4 миллиарда	от 300 миллионов	от 1,7 миллиарда
2	Английский язык ^[8]	380 миллионов	260 миллионов	от 640 миллионов
3	Испанский язык ^[11]	425 миллионов	до 125 миллионов	до 550 миллионов
4	Арабский язык ^[12]	280 миллионов	от 80 миллионов	до 380 миллионов
5	Французский язык ^[13]	80 миллионов	до 200 миллионов	до 280 миллионов
6	Русский язык	~ 118 миллионов	до ~ 140 миллионов	до 258 миллионов
7	Португальский язык ^[14]	200 миллионов	до 30 миллионов	до 230 миллионов
8	Немецкий язык ^[15]	80 миллионов	до 80 миллионов	до 160 миллионов

Рисунок 8 Фильтрация по диапазону

<https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA%D0%B8%D0%BC%D0%B8%D1%80%D0%BD>

Пример 2: если @title != text() //table[@class="standard"]//td/a[not(compare(@title,.) = 0)]

(аналогично: for \$x in //table[@class="standard"]//td/a return \$x[\$x/@title != \$x/text()]

"/>

Рисунок 9

ends-with(), starts-with()

Функция **starts-with()** проверяет, начинается ли строка с указанного текста, например, нужно найти все строки начинающиеся со слова "Источник":

XPath: `//p[starts-with(., "Источник")]`

Результат: Будут найдены все `<p>` начинающиеся со слова "Источник", в других частях текста будут игнорироваться (рис.10)

The screenshot shows a news article from the website 'Mirovoe Obzrenie'. At the top, there is a yellow highlighted box containing the XPath expression `//p[starts-with(., "Источник")]`. Below the header, the article title is 'Гены-спасители: учимся у тихоходок и витамина С' (Genes-saviors: learning from tortoises and Vitamin C). A green checkmark icon is placed to the left of the title. Below the title, there is a yellow box containing the source link 'Источник: https://tehnowar.ru/'. The main text of the article discusses the potential of CRISPR technology to help astronauts survive in space by modifying their genes to withstand radiation and cancer. A red X icon is placed to the left of the first paragraph of the text.

Рисунок 10

Выбор заголовков по началу фразы:

Исходная строка: `<h1>Breaking News: Important Update</h1>`

Найти заголовки статей, начинающиеся с "Breaking News":

`//h1[starts-with(text(), "Breaking News")]`

Исключение элементов, начинающихся с символа:

Исходная строка: `Внутренняя ссылка`

`Внешняя ссылка`

Исключить все ссылки, начинающиеся с # (якорные ссылки):

`//a[not(starts-with(@href, "#"))]`

Фильтрация классов по префиксу:

Найти элементы с классами, начинающимися на "article":

Исходная строка: <div class="article-summary">Краткое описание</div>

<div class="article-full">Полный текст</div>

//*[@class and starts-with(@class, "article")]

Использование ends-with()

Поиск изображений по расширению:

Исходная строка:

Найти все изображения с расширением .png:

//img[ends-with(@src, ".png")]

Фильтрация ссылок по домену:

Исходная строка: Официальный сайт

Обычный сайт

Найти ссылки, заканчивающиеся на .gov:

//a[ends-with(@href, ".gov")]

Совместное использование starts-with() и ends-with():

Фильтрация ссылок по началу и окончанию:

Найти ссылки, начинающиеся с "http" и заканчивающиеся на ".html":

//a[starts-with(@href, "http") and ends-with(@href, ".html")]

Фильтрация классов по начальной и конечной части:

Найти элементы, у которых @class начинается на "hotNews" и заканчивается на "-title" (рис. 11):

//a[starts-with(@class, "hotNews") and ends-with(@class, "-title")]/@class

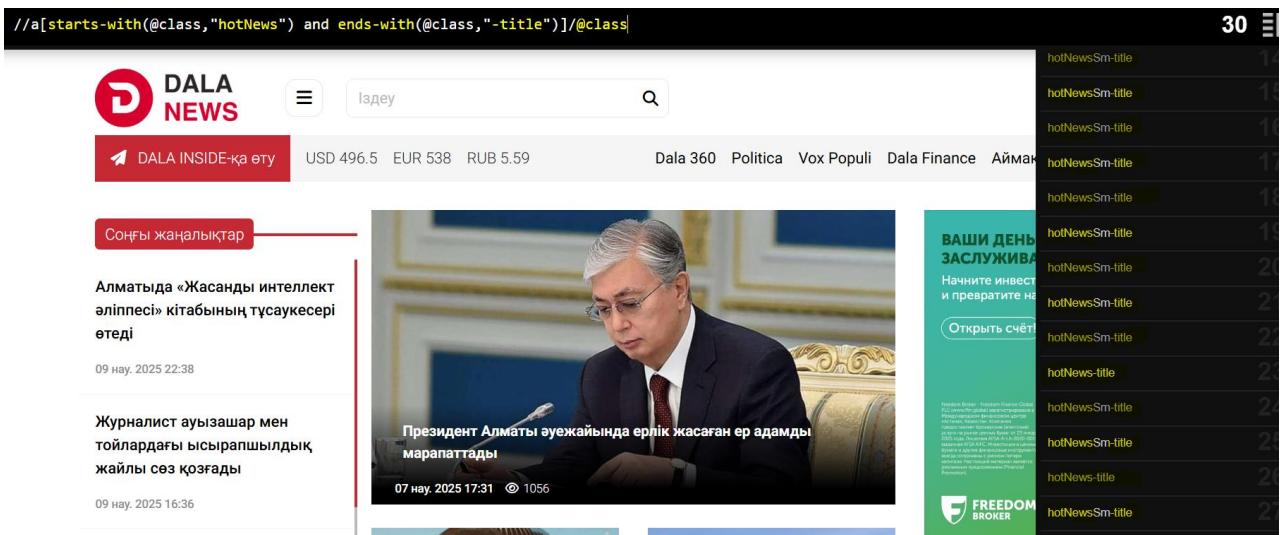


Рисунок 11

Альтернативное решение: //a[matches(@class, '^hotNews.*-title\$')]

Исключение нежелательных ссылок:

Исключить ссылки, которые начинаются с # или заканчиваются на .pdf:

//a[not(starts-with(@href, "#")) and not(ends-with(@href, ".pdf"))]

Расширенные примеры:

Преобразование ссылок с if-then-else

Если @href начинается с "http:", заменить на "https:", иначе оставить без изменений:

```
for $x in //a return if (starts-with($x/@href, "http:")) then concat("https:", substring($x/@href, 6)) else $x/@href
```

Извлечение доменных имен из .com-ссылок

Получить только домены из ссылок, заканчивающихся на .com:

```
for $x in //a[ends-with(@href, ".com")] return substring-before(substring-after($x/@href, "://"), "/")
```

lower-case(), upper-case()

lower-case преобразует все буквы строки в нижний регистр (маленькие буквы):

lower-case("ИСТОЧНИК") -> Результат: "источник"

В результате можем брать, например, ссылки на новости, не обращая внимания на регистр:

<a>"Подробнее"<a/>, <a>"ПОДРОБНЕЕ"<a/> или <a>"ПодробнЕЕ"<a/> и т.п.:

Применив функцию `lower-case()` - //a[contains(lower-case(.),"подробнее")] будет учитываться любой регистр.

Также можно работать с названиями классов и других атрибутов, просто преобразуем класс с помощью функции:

//div[lower-case(@class) = "main"] -> Результат: находит элементы <div>, где атрибут @class равен "main" (без учета регистра).

upper-case() работает аналогично ф-ции `lower-case()`, но преобразует все буквы строки в верхний регистр (большие буквы)

`upper-case("источник")` -> Результат: "ИСТОЧНИК"

Пример: //span[contains(upper-case(.),"источник")]

Случай совместного использования ф-ций. Если в заголовках статей первая буква не заглавная, то можно исправить это с помощью `lower-case()` и `upper-case()`, чтобы первая буква всегда была заглавной, а остальные — строчными:

`concat(upper-case(substring(normalize-space(/h1), 1, 1)), lower-case(substring(normalize-space(/h1), 2)))`

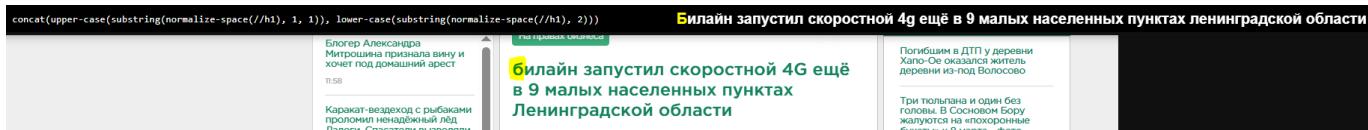


Рисунок 12 Заголовок начинается с "билин" – исправляем на "Билайн"

Также данный способ можно использовать при ошибке кодировки (рис. 13):

A screenshot of a news article from 'akhty-news.ru'. The title 'Вероятно, у статьи неправильная кодировка.' is highlighted in yellow, indicating a decoding error. Below the title, there is a detailed analysis of the character distribution in the title, which is also highlighted in yellow.

Рисунок 13 Ошибка кодировки

if (/h1 = upper-case(/h1)) then concat(upper-case(substring(/h1, 1, 1)), lower-case(substring(/h1, 2))) else /h1

normalize-space(): удаляет все пробелы в начале и в конце строки, а между словами удаляет только дублирующие пробелы.

Т.е., если в строке пробелы стоят до первого слова или после последнего слова либо если между словами более 1 пробела, то функция уберет лишние пробелы:

Строка: <div class="text"><пробелы> Источник: <более 1 пробела> www.fax.ru<пробелы></div>

Запрос: normalize-space(//div[@class="text"])

Результат: "Источник: www.fax.ru"

Применение для точного поиска текста:

Функция normalize-space() полезна для поиска элементов по точному совпадению текста, учитывая пробелы. Если использовать функцию contains() для поиска элементов, то есть риск обрезания абзацев, т.к. через contains() поиск осуществляется по куску текста. Поэтому если искать точное совпадение, например какого-либо слова (фото, источник, видео и т.п.), то normalize-space() идеально подходит.

Например если в строке с текстом " видео " есть лишние пробелы, запрос //h1[text()="видео"] не сработает. Дополнив выражение функцией:

//h1[normalize-space(text())="видео"], то результат будет корректным, так как пробелы в начале и в конце игнорируются.

//a[normalize-space(translate(., ' ', ' ')) = "Ссылка на регистрацию"] – найдет по точному совпадению даже если неразрывн пробел

Разница между text() и . (точка)

//h1[normalize-space(text())="Погода сегодня"]: применяется только к текстовому узлу текущего элемента (расположенный в самом h1) (рис. 14).

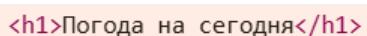


Рисунок 14

//h1[normalize-space(.)="Погода на завтра"]: применяется ко всему содержимому элемента, включая все вложенные узлы (на всю глубину вложенности) (рис. 15).

```
<h1 class="headline" style="animation-delay: 1.3s;">
  <span class="title__outline"> == $0
    > <svg class="title__outline-svg title__outline-svg--first" width="497"
      ...
    </svg>
    "Погода на завтра"
  </span>
</h1>
```

Рисунок 15

Еще пример:

```

<div class="head">
    <h1>Текст 5</h1>
    <h1>Текст 1</h1>
    <h1>
        <div> Текст 1</div>
    </h1>
</div>

//h1[normalize-space(text())="Текст 1"] найдет только <h1>Текст 1</h1>
//h1[normalize-space(.)="Текст 1"] найдет оба: <h1>Текст 1</h1> и <div>Текст 1 </div>

```

Поиск элементов с текстом и пустых элементов

```

<div class="subhead">
    <h2> </h2>
    <h2> Текст 1</h2>
    <h2> </h2>
    <h2> Текст 2 </h2>
    <h2> Текст 5 </h2>
</div>

```

Элементы с текстом: Если нужно найти все `<h2>` в которых есть какой-либо текст, кроме пробелов, то применив `//h2 [normalize-space ()]` найдем все `//h2` с текстом ("Текст 1", "Текст 2", "Текст 5"), пустые `<h2>` буду игнорироваться.

Пустые элементы (без текста) `<h2>`, применим функцию `not()`:

`//h2 [not (normalize-space ())]` будут найдены все пустые элементы `<h2>`

Применение к классам и атрибутам

`normalize-space()` также, как и к тексту применим к классам и прочим атрибутам:

Убираем пробелы: `<div[@class = "<пробелы>article<пробелы>"]>`, используем

`//div[normalize-space (@class) ="article"]`

Найдем все пустые ссылки: `перезагрузка` (ведёт на текущий URL)
`//a [not (normalize-space (@href))]`

Сочетание с другими функциями

При работе с другими функциями `normalize-space()` необходим, как и с примером, который уже был рассмотрен выше с функцией `not()`, рассмотрим еще пару примеров:

Использование с `starts-with()`:

Нужно найти тег `<p>` текст которого начинается со слова "Источник", но в строке до слова стоят пробелы: `<p>" Источник: rbc.ru"` `</p>`

выражение //p[starts-with(., "Источник:")] не работает, т.к. мы ищем по правилу, где строка начинается именно со слова "Источник:", то добавим ф-цию тем самым убрав лишние пробелы и все заработает:

```
//p[starts-with(normalize-space(.), "Источник:")]
```

Использование с string-length():

Найти дату в теге <time> и используем функцию string-length(), которая измеряет длину строки и позволяет по количеству символов найти дату формата "31.05.2025", т.е. ровно 10 символов. И зачастую бывают лишние пробелы и выражение

```
//time[string-length() = 10]
```

 не сработает, снова используем normalize-space() и избавимся от пробелов: //time[string-length(normalize-space(.)) = 10], и тем самым функция string-length() сработает.

Ф-ция normalize-space() очень важна, т.к. она обеспечивает корректную обработку строк с лишними пробелами, табами или переносами строк. Без неё многие операции, такие как поиск по началу строки, сравнение длины или логические проверки ([например, на сравнение](#)), могут давать неверные результаты.

replace() заменяет заданную последовательность (точное совпадение) символов в строке на указанный текст. Стандартная конструкция:

```
replace(xpath-путь, "что заменить", "на что заменить").
```

Пример замены символа "&" на пробел:

Строка: <p>Заголовок&номер&один</p>

Запрос: replace("//p, "&", " ") -> Результат: "Заголовок номер один"

В Xpath replace() это 1 из 3 функций, которая позволяет использовать регулярные выражения, а также Unicode для замены сложных шаблонов текста. Можно находить и заменять не только конкретные подстроки, но и текст, соответствующий определённым условиям.

Пример замены всех целочисленных значений на * (звездочку) используя regex:

Строка: <p>"Номер карты: 1234 5678 9101 1121"</p>

XPath: replace("//p, '\d', '*')

Результат: "Номер карты: **** * * * * * * *"

Рассмотрим некоторые практические примеры использования регулярных выражений:

Задача: Извлечь из строки "КИНДЕРINFO 23.01.2025 ПРАЗДНИКИ ДЛЯ ДЕТЕЙ" дату ("23.01.2025") (рис. 16)

Решение: replace("//p[@class='text'], '.*?(\d{2}\.\d{2}\.\d{4}).*', '\$1')



Рисунок 16

Разбор регулярного выражения:

.*?: пропускает все символы до первого совпадения с датой.

(\d{2}\.\d{2}\.\d{4}): извлекает группу, представляющую дату в формате дд.мм.гггг

.*: пропускает оставшиеся символы после даты.

\$1: заменяет весь текст на содержимое первой группы (дату).

В результате используя replace() + регулярные выражения можем найти точное совпадение заданной последовательности (даты и др.) в строках, где "плавающий контекст" и зацепиться с помощью других функций таких как substring-before(), substring-after() и пр. нельзя.

(Шаблоны распространенных форматов дат)

Бывают случаи, когда в строке разделителем слов служат управляющие символы, например, вместо обычного пробела указан неразрывный пробел из-за которого может возникнуть проблема с извлечением даты. С помощью replace() и Unicode* найдем и заменим данные символы:

Строка: " понедельник 11:54, 16 декабря 2024 "

Запрос: `replace("//p, '\u00A0', ' ')`

Результат: "понедельник 11:54, 16 декабря 2024"

Т.о. мы избавились от неразрывного пробела заменив его на обычный.

string-join(): объединяет последовательность строк в одну строку, используя указанный разделитель.

Например, есть дата, которая разделена на 3 отдельные строки:

```
<div[@class="date"]>
  <span>10</span>
  <span>01</span>
  <span>2024</span>
</div>
```

Используем `string-join(div[@class="date"]/span, "-")` и в качестве разделителя подставим тире, в результате получим единую строку "10-01-2024".

Рассмотрим более сложный пример, где нужны значения даты (день, месяц, год) и расположены они не последовательно: <https://ncrb-ru/news/detail/samoregulyatoru-vrayali-otvetstvennost/> (актуальность ссылки 27.01.2025)

Для данного примера (рис.17) есть различные варианты решений, рассмотрим два:

1. С функциями `string-join()` и `position()` с оператором `mod` найдем нечетные элементы:

```
string-join(//div[@class="top-news_data news_data"]/span[position() mod 2 = 1], ".")
```

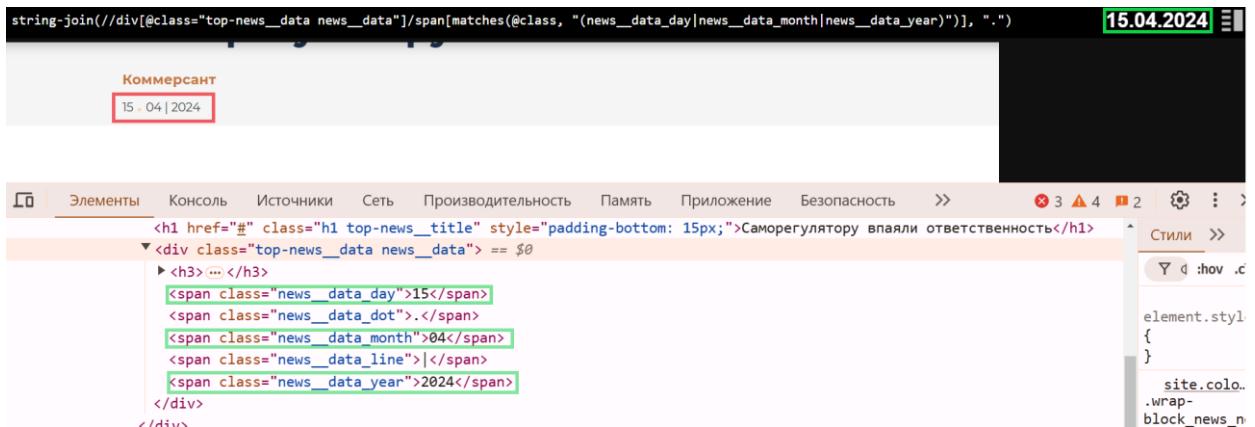


Рисунок 17

2. Применим функцию `matches()`, что дает возможность использовать регулярные выражения, и тем самым оптимизируем запись просто перечисляя названия классов.

```
string-join(//div[@class="top-news_data news_data"]/span[matches(@class, "(news_data_day|news_data_month|news_data_year)")], ".")
```

Разбор:

`//div[@class="top-news_data news_data"]/span` - находим путь до нужных элементов

`matches(@class, "(news_data_day|news_data_month|news_data_year)")` - через оператор "|" ("ИЛИ") перечисляем нужные классы

`(news_data_day|news_data_month|news_data_year)`

`string-join(..., ".")` – используем функцию для объединения строк и используем точку в качестве разделителя.

У каждого решения свои "плюсы-минусы", в случае с поиском по нечетным элементам есть вероятность, что в другой статье источника они будут расположены в другом порядке и выражение не сработает, но оно простое и понятное. Либо 2й вариант может показаться более громоздким и трудночитаемым, но он позволяет перечислять классы упрощенной записью. Также в примерах выше, не прибегая к `string-join()`, можно применить функцию `concat()` просто объединив все строки в нужном порядке: `concat(//span[@class="news_data_day"], ".", //span[@class="news_data_month"], ".", //span[@class="news_data_year"])`

Еще пример reverse() + string-join():

https://rkcoop.narod.ru/news/konkurs_luchshij_pirog_sredi_kooperatorov/2025-03-21-143

string-length() функция возвращает количество символов в строке, включая пробелы и спецсимволы.

Некоторые варианты работы с функцией string-length():

1. Определение длины строки:

Исходная строка : <p>"Нагатинский затон"</p>

Применяем функцию: string-length(//p)

Результат: длина строки 17 символов

2. Фильтрация по длине строки с операторами сравнения (=, !=, >, <, >=, <=):

//h1[string-length(.) = 5] - найдёт все <h1>, где текст состоит ровно из 5 символов

//p[string-length(.) != 6] - все теги <p>, где текст не равен 6 символам

//p[string-length(text()) > 10] - все <p>, где текст длиннее превышает 10 символов

//p[string-length(text()) >= 4] - все <p>, где текст содержит 4 или больше символов

3. Использование логических операторов: and, or, not:

//p[string-length(.) >= 2 and string-length(.) < 200] - найдет все <p>, где текст от 2 до 199

//p[string-length(.) = 5 or string-length(.) = 7] - найдет все <p>, где текст содержит 5 или 7 символов

//p[string-length(.) >= 2 and string-length(.) != 6] - все <p>, где текст не менее 2 символов и не равен 6

4. Проверка на пустую строку:

Использование для проверки, является ли элемент пустым:

//div[string-length(normalize-space(.)) = 0] - найдёт все <div>, которые пустые или содержат только пробелы.

Альтернативный вариант: //div[not(normalize-space(.))]

5. Сравнение длины строки между элементами:

//p[string-length() = string-length(//h1)] - найдет все `<p>` у которых длина строки совпадает с ближайшим `<h1>` (например у заголовка и подзаголовка длина строки одинаковая и скорее всего название у них совпадает) (рис. 18)

//p[string-length(.) > string-length(//h1)] - все `<p>`, у которых текст длиннее, чем у `<h1>`.

The screenshot shows a search results page with the query `//p[string-length() = string-length(//h1)]`. The first result is a weather forecast for Kirovskoe on March 7. The headline is "Завтра в Кировском ожидается теплая и солнечная погода". Below it is a weather card with a sun icon, the temperature +9°, and a forecast for the next day: "7 марта — день тюльпанов, а ведь именно эти цветы ассоциируются у нас с весной". To the right, there's a sidebar with a news feed titled "ЛЕНТА ТЕКСТОВ" showing a snippet about warm and sunny weather.

Рисунок 18 Найден один `<p>` совпадающий по длине строки с `<h1>`.

6. Проверка длины атрибутов:

//a[string-length(@href) > 20] - найдёт все `<a>`, где url длиннее 20 символов

7. Проверка длины строки без "лишних пробелов" (функция normalize-space()):

Исходная строка с пробелами: `<p>"Фото: 15 "</p>`

Применяем функции: `//p[string-length(normalize-space(.))]`

Результат: Преобразует строку "Фото: 15 " в "Фото: 15", длина 8 символов

8. Подсчёт символов без пробелов:

`string-length(translate(., ' ', ''))` - выражение удаляет все пробелы и возвращает длину строки без них (функция `translate()` удаляет пробелы, а затем `string-length()` проверит длину строки без пробелов)

9. Использование с функцией substring(). Для динамического разбиения ([оператор div](#)) текста в зависимости от длины:

Поиск 1й половины строки:

Исходный текст: `< h1 >"xxxYYY"</h1>`

XPath: `substring(//h1, 1, string-length(//h1) div 2)`

Результат: "xxx"

Поиск 2й половины строки:

Исходный текст: < h1 >" xxxYYY "</h1>

XPath: substring("//h1, string-length("//h1) div 2 + 1)

Результат: "YYY"

Удаление последнего символа в строке:

Исходный текст: < time >"1 декабря 20230</time>" (необходимо удалить последний символ - 0)

XPath: substring("//h1, 1, string-length("//h1) - 1)

результат: "1 декабря 2023"

10. Извлечение даты в формате "дд.мм.гггг", где количество символов, как правило, не меняется и равно 10:

Стока: <time >" 02/05/2025</time>

XPath: //time[string-length(normalize-space(.)) = 10]

(Функция normalize-space() удаляет начальные и конечные пробелы, а также лишние пробелы между словами, что позволяет точно определить длину строки без лишних пробелов. Стока после такой обработки содержит ровно 10 символов).

string-to-codepoints(): Преобразует строку в список чисел, где каждое число — это код символа из таблицы Unicode (прим. <https://ss64.com/ascii.html>). Например, символ "A" станет числом 65, а символ "п" — числом 1087.

codepoints-to-string(): Делает обратное преобразование: берёт список чисел (кодов символов) и преобразует обратно в строку. Например, числа [65, 66, 67] станут строкой "ABC".

Пример:

string-to-codepoints ("ABC") вернёт [65, 66, 67]

codepoints-to-string ([65, 66, 67]) вернёт "ABC"

1. Найдем и удалим из строки - "понедельник&nbsp16:50&nbsp10&nbspфевраля&nbsp2025" символы неразрывного пробела (&nbsp), из-за которых иногда проблематично извлечь дату (рис. 19):

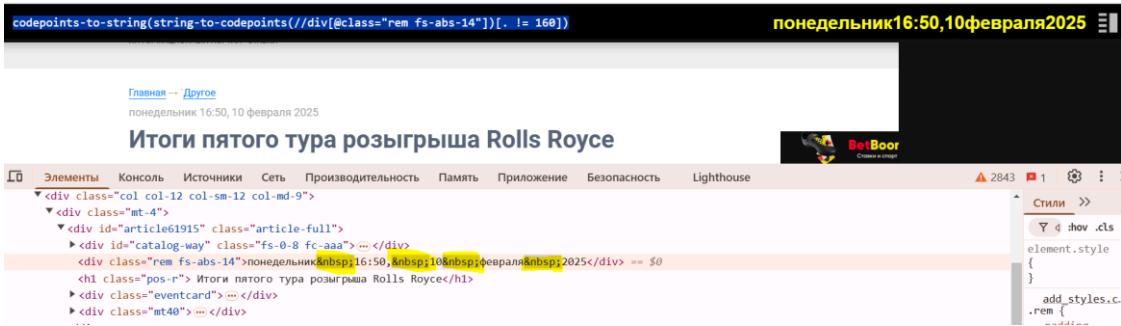


Рисунок 19

Решение:

```
codepoints-to-string(string-to-codepoints(//div[@class="rem fs-abs-14"])[. != 160])
```

Разбор:

`//div[@class="rem fs-abs-14"]` – путь до даты (найдем `<div>` с классом "rem fs-abs-14")

`string-to-codepoints(//div[@class="rem fs-abs-14"])` – функция преобразует каждый символ в строке в "коды символов Unicode", в результате получим последовательность [1087, 1086, 1085, 1077, 1076, 1077, 1083, 1100, 1085, 1080, 1082, 160, 49, 54, 58, 53, 48, 44, 160, 49, 48, 160, 1092, 1077, 1074, 1088, 1072, 1083, 1103, 160, 50, 48, 50, 53]. Нужные элементы под номером 160 (неразрывный пробел) (рис. 20)

URL:	https://www.stavki.ru/article/61915
Заголовок:	Итоги пятого тура розыгрыша Rolls Royce
Дата и время:	10.02.2025 16:50:00
Автор:	
Номер выпуска:	
Текст(включая изображения по тексту),	plain
1087 1086 1085 1077 1076 1077 1083 1100 1085 1080 1082 160 49 54 58 53 48 44 160 49 48 160 1092 1077	: [plain]

Рисунок 20

`[. != 160]` – с помощью фильтра исключим элементы под номером 160, в результате получим последовательность [1087, 1086, 1085, 1077, 1076, 1077, 1083, 1100, 1085, 1080, 1082, 49, 54, 58, 53, 48, 44, 49, 48, 1092, 1077, 1074, 1088, 1072, 1083, 1103, 50, 48, 50, 53]

`codepoints-to-string()` – функция преобразует отфильтрованную последовательность "code points" обратно в строку. В результате получим дату без неразрывного пробела: "понедельник16:50,10февраля2025"

2. Взять из строки "пятница02/03/2025года" только цифры:

```
codepoints-to-string(string-to-codepoints("пятница02/03/2025года") [. >= 48 and . <= 57])
```

```
codepoints-to-string(string-to-codepoints("пятница02/03/2025года") [. >= 48 and . <= 57]) 02032025
```

Рисунок 21

Пояснение: как и в предыдущем примере, применяем похожий фильтр [. >= 48 and . <= 57], используя операторы сравнения, чтобы выбрать нужный диапазон ("коды символов Unicode" от 48 до 57, которые соответствуют символам цифр от 0 до 9). В результате из строки извлекаются только цифры (02032025), а все остальные символы удаляются.

tokenize() разбивает строку на последовательность подстрок (как бы на отдельные слова) по заданному разделителю, это **1 из 3 функций** принимающая регулярные выражения.

Пример:

Стока: `<div>"Lunes, 02 de septiembre de 2024 a las 11:47 pm"</div>`

Применяем функцию: `tokenize("//div, " ")`

Первый аргумент — **XPath-путь**

Второй аргумент — **разделитель, в данном случае пробел " "**

Результат: последовательность из 10 строк разделенная пробелами: ("Lunes," "02" "de" "septiembre" "de" "2024" "a" "las" "11:47" "pm")

Задача: извлечь из строки дату - "02 septiembre 2024" (день месяц год).

Рассмотрим несколько решений:

1. Простое, но громоздкое:

```
concat(tokenize(normalize-space("//div[@class='articulo_fecha mobile']"), " ") [2], " ", tokenize(normalize-space("//div[@class='articulo_fecha mobile']"), " ") [4], " ", tokenize(normalize-space("//div[@class='articulo_fecha mobile']"), " ") [6])
```

Мы объединяем подстроки под индексами 2, 4 и 6: "02", "septiembre", "2024". Недостаток решения— повторяется длинное выражение с `normalize-space()`.

2. Через итерацию for (рис. 22)

```
for $x in normalize-space(//div[@class="articulo_fecha mobile"]) return concat(tokenize(normalize-space($x), " ")[2], " ", tokenize($x, " ")[4], " ", tokenize($x, " ")[6])
```

Такой способ проще читается и повторно использует переменную \$x.



Рисунок 22

3. Альтернативный вариант с string-join():

```
string-join(tokenize(normalize-space(//div[@class="articulo_fecha mobile"]), " ")[position() = (2, 4, 6)], " ")
```

Это компактное и наглядное решение, особенно при извлечении нескольких элементов по индексам.

Разные примеры:

1. Разделение по пробелам:

```
tokenize("яблоко груша банан", "\s+")
```

Результат: ("яблоко", "груша", "банан")

* \s+ обрабатывает любые пробельные символы (пробел, табуляция, перенос строки).*

2. Разделение по запятым с пробелами и без:

```
tokenize("понедельник, вторник, среда", ",\s*")
```

Результат: ("понедельник", "вторник", "среда")

* Шаблон ",\s*" работает даже если пробелов нет

3. Посимвольное разбиение:

Самый простой вариант использовать tokenize("hello", "") *

Результат: ("h", "e", "l", "l", "o")

* Но пустой шаблон ("") не сработает в ..., т.к. требуется поддержка определенной реализации XPath.

Для того, чтобы работало в ... можно использовать "обходной путь":

```
 tokenize(replace("hello", "(.)", "$1,"), ",")[. != "]
```

Пошаговое объяснение:

Добавляем запятую после каждого символа:

Используем replace("hello", "(.)", "\$1,"):

(.) — захватывает любой символ

"\$1," — заменяет его на символ + запятую

Результат: "h,e,l,l,o,"

Разбиваем строку по запятой:

Применяем tokenize("h,e,l,l,o,", ","):

Делит строку по запятым.

Результат: ("h", "e", "l", "l", "o", "") (пустая строка в конце из-за запятой после о).

Убираем пустые элементы:

Фильтруем список: [. != "]

Финальный результат: ("h", "e", "l", "l", "o")

4. Извлечение слов без пунктуации:

```
 tokenize("Сегодня! Хороший день?", "\W+")[normalize-space()]
```

Результат: ("Сегодня", "Хороший", "день")

* \W+ удаляет небуквенные символы. [normalize-space()] фильтрует пустые строки.*

5. Разделение даты (дд.мм.гггг):

```
 tokenize("30.10.2023", "\.")
```

Результат: ("30", "10", "2023")

* Точка необходимо экранировать \.*

6. Если в строке >= 3 символов "#"

```
//p[count(tokenize(., '#')) >= 3]
```

Другие примеры:

7. Извлечение IP-адреса:

```
 tokenize("192.168.1.100", "\.")
```

Результат: ("192", "168", "1", "100")

* Каждый октет извлекается отдельно.*

8. Удаление HTML-тегов:

```
 tokenize("<div>Текст</div><p>Параграф</p>", "<[^>]+>")[normalize-space()]
```

Результат: ("Текст", "Параграф")

* Шаблон <[^>]+> удаляет все теги. [normalize-space()] убирает пустые строки.*

9. Подсчёт слов в предложении:

```
 count(tokenize("Сколько здесь слов?", "\s+"))
```

Результат: 3

* \s+ разделяет по пробелам, count() считает элементы.*

10. Разделение пути в Windows:

```
 tokenize("C:\\\\Program Files\\\\MyApp", "\\\\+")
```

Результат: ("C:", "Program Files", "MyApp")

* Экранирование обратного слеша: \\\+ вместо [\\\\\\\\].*

11. Универсальный парсинг дат:

```
 tokenize("2024/12.31", "-/.")
```

Результат: ("2024", "12", "31")

* Работает для разделителей: -, /, .*

translate() заменяет или удаляет отдельные символы в строке по правилам:

```
 translate(строка, "символы_для_замены", "новые_символы").
```

В отличие от replace(), которая работает с подстроками (последовательностями), translate() оперирует только символами.

Варианты применения на примере одной строки:

Строка: <p>"a1b2c3"</p>

```
//translate("//p, "123", "") -> результат: "abc"  
//translate("//p, "abc", "") -> результат: "123"  
//translate("//p, "abc", "ABC") -> результат: "A1B2C3"  
//translate("//p, "123", "!!!") -> результат: "a!b!c!"  
//translate("//p, "bc", "КО") -> результат: "а1К2О3"
```

Другие примеры:

Удаление пробелов и дефисов:

```
translate("+7 (999) 123-45-67", " ()-", "") -> Результат: 79991234567
```

Кириллица из верхнего в нижний регистр:

```
translate("ПРИМЕР ТЕКСТА", "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЭЮЯ",  
"абвгдеёжзийклмнопрстуфхцчшщьыэюя") -> Результат: "пример текста"
```

Латиница из нижнего в верхний регистр:

```
translate("example text", "abcdefghijklmnopqrstuvwxyz", "ABCDEFGHIJKLMNOPQRSTUVWXYZ") ->  
Результат: "EXAMPLE TEXT"
```

Удаление нецифровых символов:

```
translate("Цена: 1,200.50 ₽", "Цена: ₽", "") -> Результат: "120050"
```

Удаление гласных:

```
translate("Hello World", "aeiouAEIOU", "") -> Результат: "Hll Wrld"
```

Удаление эмодзи:

```
translate("Скидка 50%!", "\ud83d\udcbb", "\ud83d\udcbb", "") -> Результат: "Скидка 50%!"
```

Замена нескольких символов:

```
translate("apple", "ap", "PA") -> Результат: PAAlle
```

Использование в комбинации с функцией NOT():

Выбираем все элементы, у которых текст содержит только цифровые значения:

```
//*[not(translate(., "0123456789", ""))] -> Результат: т.е. используем not() для обратного значения и  
из строки вырезается все кроме цифр.
```

Ищем элементы, которые содержат/не содержат определенный текст (русский, английский, цифры, а также регистр):

//*[not(translate(*, "абвгдёйзийклмнoprстуфхцчшъыъэюя", ""))] -> Результат: выборка всех элементов у которых текст не содержит русские буквы в нижнем регистре.

Также можно использовать Unicode:

translate("Цена: 100€", "\u20AC", "\$") -> Результат: "Цена: 100\$"

matches() Проверяет соответствие строки куску текста или регулярному выражению.

Возвращает true/false.

matches() отчасти похожа на функцию contains() для простого поиска фрагментов текста, но важным отличием является использование регулярных выражений и наличие третьего параметра.

Особенности синтаксиса:

matches(\$input, \$regex, \$flags)

\$input — Текст, который проверяем.

\$regex — кусок строки или регулярное выражение.

\$flags (не обязательный параметр) — Пример: i(Игнорирование регистра), s(Точка включает перенос строки) и т.д.

Примеры:

1. Проверка наличия цифр:

Исходная строка: <p>"abc123"</p>

XPath: //p[matches(., '\d')]

Результат: true (строка содержит цифры)

2. Поиск слова без учёта регистра:

Исходная строка: <div>ИСТОЧНИК:</div>

XPath: matches(., "источник", 'i')

Результат: true (найдено слово "источник", игнорируя регистр)

3. Проверка формата даты (дд.мм.гггг):

Исходная строка: "30.10.2023"

XPath: matches("30.10.2023", "^\\d{2}\\.\\d{2}\\.\\d{4}\$")

Результат: true (соответствует формату даты)

4. Игнорирование пробелов в начале текста:

Исходная строка: <p>" Крупнейший"</p>

XPath: //p[matches(., '^\\s*Крупнейший')]

Результат: true (слово "Крупнейший" найдено, даже если перед ним пробелы)

5. Поиск email:

Исходная строка: "user@mail.com"

XPath: matches("user@mail.com", "^[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,}\$", "i")

Результат: true (корректный email)

6. Многострочный режим:

Исходная строка: "Строка1\\nСтрока2"

XPath: matches("Строка1\\nСтрока2", "^Строка2\$", "m")

Результат: true (поиск по второй строке)

Флаги:

i — игнорирование регистра.

m — ^ и \$ работают для каждой строки.

s — точка . включает перенос строки.

x — игнорирует пробелы в шаблоне.

Практическое применение:

Поиск элементов с классами "error" или "warning"

XPath: //*[matches(@class, 'error|warning', 'i')]

Проверка наличия букв в строке:

XPath: matches("123a", '[A-Za-z]')

Результат: true (есть буква)

distinct-values() — возвращает уникальные значения из последовательности, такие как строки, числа и даты. Функция работает только с значениями, а не с узлами.

Например, `distinct-values(//a/text())` удалит повторяющиеся заголовки, но `distinct-values(//a)` вызовет ошибку, потому что //a — это узлы, а не значения. Примеры:

Строки: `distinct-values(("apple", "orange", "apple", "banana"))` вернет ("apple", "orange", "banana"), исключая дубликат "apple".

Числа: `distinct-values((1, 2, 3, 1, 4, 2))` вернет (1, 2, 3, 4), удаляя повторяющиеся числа.

Булевые значения: `distinct-values((true(), false(), true()))` вернет (true(), false()), исключая повторяющееся true()

Фильтрация дублей ссылок:

Исходные значения: повторяющиеся ссылки (рис. 23)

```
string-join(distinct-values(//div[@id="link"]/p/a), " ")
```

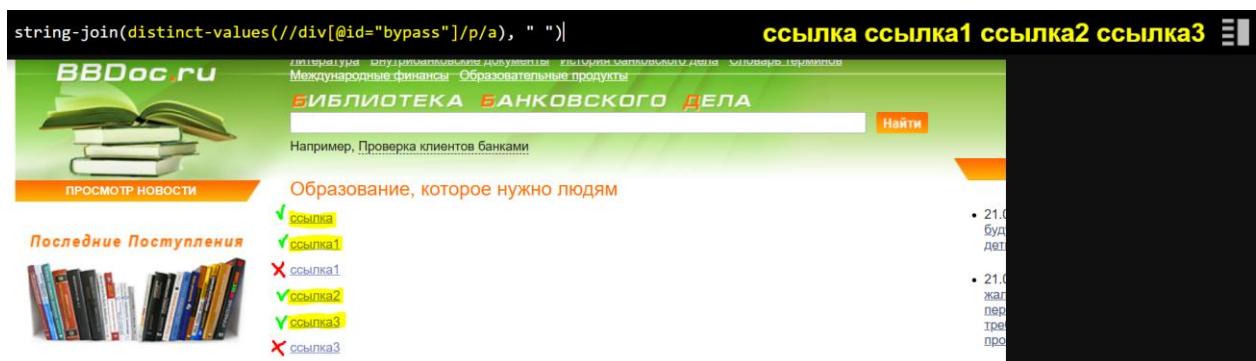


Рисунок 23

Фильтрация дублей статей по заголовку:

Если в ленте новостей встречаются одинаковые статьи, то их можно отфильтровать – в этом случае из всех дублей будет выбрана только одна (например, из трех одинаковых статей будет выбрана только одна) (рис. 24)

```
for $x in distinct-values(//*[@class='default_box_content']//a[contains(@href, 'nalchik/news')]/text()) return (//*[@class='default_box_content']//a[contains(@href, 'nalchik/news')])[text() = $x][1]
```

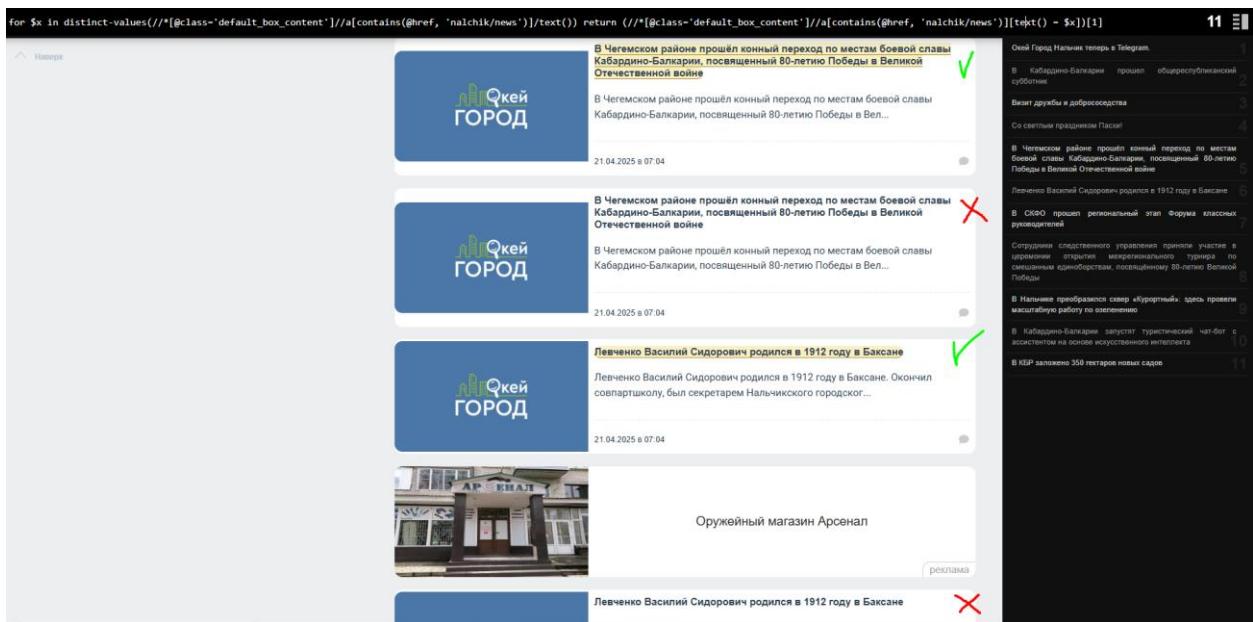


Рисунок 24

reverse() возвращает последовательность элементов в обратном порядке, т.е. первый элемент становится последним, а последний – первым.

Если текст разбит на абзацы (теги `<p>`) внутри блока (`//div[@itemprop="articleBody"]//p`), то функция `reverse()` изменит порядок этих абзацев (рис. 25). Если же применять `reverse()` к самому контейнеру (`//div[@itemprop="articleBody"]`), то порядок абзацев не изменится (останется прежним).

`reverse("//div[@itemprop='articleBody']//p")`

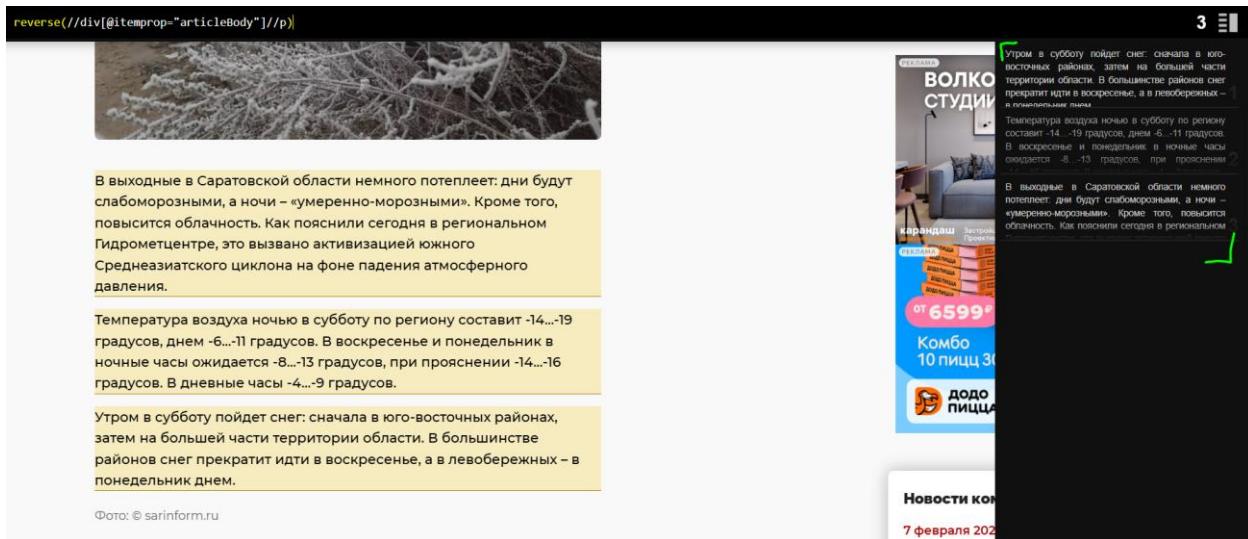


Рисунок 25

Если необходимо применить `reverse()` к 1 строке, для того чтобы последнее слово стало первым, то необходимо будет разбить все слова с помощью `tokenize()`, а затем снова соединить их в 1 строку.

На двух последующих рисунках показаны несколько вариантов с заголовком:

`string-join(reverse(tokenize("//h1, \" ")), " ")` – заголовок в обратном порядке -> "Три Два Один" (рис. 26)



Рисунок 26

`string-join(reverse(tokenize("//h1, '(?<=.')')), "")` – зеркальный заголовок (рис.27)



Рисунок 27

Еще пример `reverse()` + `string-join()`:

https://rkcoop.narod.ru/news/konkurs_luchshij_pirog_sredi_kooperatorov/2025-03-21-143

subsequence() возвращает часть последовательности, начиная с указанной позиции (с 1) и, опционально, ограничивает количество элементов. Если длина не задана, возвращаются все элементы от начальной позиции до конца последовательности.

Синтаксис:

`subsequence($sequence, $start, $length?)` :

\$sequence — xpath-путь

\$start — позиция, с которой начинается извлечение (индексация начинается с 1).

\$length (опционально) — количество элементов для извлечения. Если не указано, возвращаются все элементы, начиная с \$start.

`subsequence("//a[@class='More'][2, 3])` – ссылки с 2 по 4 позиции (рис. 28)

`subsequence("//a[@class='More'][2,)` – ссылки со 2 позиции и до конца

`subsequence("//a[@class='More'][1, count("//a[@class='More'])])` - последняя ссылка

`subsequence("//a[@class='More'][1, count("//a[@class='More']) - 1])` - две последние ссылки

Рисунок 28

Альтернативой `subsequence()`, являются различные комбинации функции `position()`. Например, `subsequence(//a, 2, 3)` эквивалентно:

```
(//a)[position() >=2 and position() <=4] либо  
(//a)[position() = 2 to 4]
```

Важно помнить, что `position()` предназначен для выборки элементов, удовлетворяющих условию – например выбрать только четные/нечетные элементы `(//a)[position() mod 2 = 0]` или `(//a)[position() mod 2 = 1]`, а с `subsequence()` такую выборку сделать нельзя, т.к. функция работает только с последовательностями.

number() преобразует значение (строка, узел или другой тип) в число. Если значение не может быть корректно преобразовано, возвращается `Nan` (Not-a-Number).

Примеры:

Преобразование строки в число: `number("123.45")` -> Результат: 123.45

Преобразование узлов в числа: Если аргумент является узлом, `number()` попытается извлечь числовое значение из текста этого узла. Например, если в теге `<p>"78"</p>`, то запрос `number(//p)` вернет число 78

Преобразование строки с пробелами: `number(" 3.14 ")` Результат: 3.14

Преобразование нечисловой строки: `number("abc")` Результат: `Nan`

Арифметическая операция: `number("10") + number("5")` Результат: 15

Фильтрации узлов с числовым содержимым (если в строке только цифры):

`//*[number() = number()]` - целые числа и с плавающей запятой

Преобразование булевых значений в числа: `number(true())` вернет 1, а `number(false())` вернет 0

Рассмотрим пример посложнее, где `number()` является ключевой функцией для фильтрации ссылок на статьи за определенные годы (прим.: <https://sarinform.ru/news>).

Со страницы опроса (рис. 29) необходимо выбрать **все статьи 2025 года и старше**, то есть исключить статьи, опубликованные **раньше 2025 года**. Используем выражение:

```
for $x in //div[@class="news_date"]
return if (number(substring-before(tokenize(normalize-space($x),
" ")[3], ",")) >= 2025)
then $x
else ()
```

Разбор:

1. **путь до дат:** `//div[@class="news_date"]` Выбираем все элементы `<div>` с классом `news_date`, содержащие даты.
2. **итерация for:** `for $x in //div[@class="news_date"]` Перебираем все найденные элементы и помещаем их в переменную `$x`.
3. **удаление лишних пробелов:** `normalize-space($x)` Функция `normalize-space()` удаляет лишние пробелы в строке.
4. **разделение строки на части:** `tokenize(normalize-space($x), " ")`;
`tokenize()` разбивает строку на части по пробелу. Например, строка "22 февраля 2025, 22:22" преобразуется в: "22", "февраля", "2025,", "22:22". Нам нужна третья часть — "2025,".
5. **удаление запятой:** `substring-before(tokenize(normalize-space($x), " ")[3], ",")`;
`substring-before(..., ",")` убирает запятую, оставляя только "2025".
6. **преобразование в число**^{*}: ф-ция `number()` преобразует строку "2025" в числовой тип и теперь мы можем использовать операторы сравнения для того, чтобы отфильтровать нужные годы `>= 2025`
7. **фильтрация по году:** используем условие `if (number(...) >= 2025) then $x else ()`. Если год 2025 или больше, возвращаем `$x`, иначе ничего `else ()`.

* Без `number()` операторы `>=`, `<=`, `>`, `<` сравнивают строки как текст (по Unicode-символам, но не как числа). `number()` приводит строку к числу, позволяя сравнивать значения математически.

for \$x in //div[@class="news__date"] return if(number(substring-before(tokenize(normalize-space(\$x), " ")[3], ",")) >= 2025) then(\$x) else()

Номер	Дата	Заголовок
1	22 февраля 2025, 22:22 ✓	Двое разбились у Воскресенского кладбища
2	22 февраля 2025, 15:27	В Балтае и Новых жилье для врачей
3	22 февраля 2025, 15:09	Под Саратовом м подростка погибл
4	22 февраля 2025, 15:00	22 февраля 2025, 14:20
5	22 февраля 2025, 14:01	22 февраля 2025, 14:01
6	22 февраля 2025, 13:01	В Энгельсе в жил лестница рухнула
7	22 февраля 2025, 12:45	Реконструкция С сутки меняется сх автобусов, закры
8	22 февраля 2025, 12:16	22 февраля 2025, 11:44
9	22 февраля 2025, 11:05	22 февраля 2025, 11:05
10	22 февраля 2025, 10:23	22 февраля 2025, 10:23
11	22 февраля 2025, 09:39	22 февраля 2025, 09:39
12	22 февраля 2025, 09:17	22 февраля 2025, 09:17
13	22 февраля 2025, 07:00	22 февраля 2025, 07:00
14	22 февраля 2025, 06:00	22 февраля 2025, 06:00
15	21 февраля 2025, 23:09	21 февраля 2025, 23:09
16	21 февраля 2025, 22:29	21 февраля 2025, 22:29
17	21 февраля 2025, 21:25	21 февраля 2025, 21:25
18	21 февраля 2025, 21:17	21 февраля 2025, 21:17
19	21 февраля 2025, 21:09	21 февраля 2025, 21:09
20		открытие ВКЛ
21		Расчет Рентабельности ЧП Мон
22		ВИДНЫЙ

Рисунок 29 Выбраны статьи >= 2025

Для более простого и понятного можно также использовать регулярные выражения (точный поиск года "2025"):

```
for $x in //div[@class="news__date"] return if (number(replace(normalize-space($x), "^.*?(\d{4}).*$", "$1")) >= 2025) then $x else ()
```

И т.о. мы избавимся от привязки к позиции элемента, как в случае с разбором выше, где мы использовали tokenize()[3].

Условная конструкция if-then-else()

В XPath 2.0 появилась конструкция if then else, которая позволяет добавлять условия прямо в выражение. Теперь можно проверять наличие данных, выбирать между разными значениями или форматами, и подставлять нужный результат в зависимости от ситуации. if then else в XPath работает так же, как условный оператор в языках программирования и помогает управлять логикой обработки данных.

Рассмотрим использование условных конструкций if-then-else для обработки ситуаций, когда на веб-странице отсутствуют текст, заголовок или любой другой элемент.

Будут показаны примеры, как подставлять альтернативные значения, чтобы такого рода статьи доходили до системы.

Конструкция if-then-else работает только с одиночными значениями и не предназначена для перебора множеств (например, при манипуляции с элементами из рубрики опроса). Для обработки множества элементов используется итерация for. Поэтому в данном разделе рассматривается if-then-else в чистом виде — на уровне отдельных статей. Но в следующем разделе, посвященному for, будут приведены комбинированные примеры for с if-then-else.

Также применим конструкцию every-satisfies в связке с if-then-else. Конструкция будет полезна если в источнике, в статьях разные правила извлечения текста (в одной статье //div[@class="text"], в другой //article[@id="articlebody"], в третьей...). В таких случаях нужно проверить, что все эти блоки действительно пустые, прежде чем подставлять альтернативу (заголовок, изображение и т.д.).

Синтаксис:

if (условие) then выражение1 else выражение2

условие — любое булево выражение (возвращает true или false).

выражение1 — результат, если условие истинно.

выражение2 — результат, если условие ложно.

Примеры:

1. Простой выбор значения: if (price > 100) then "дорого" else "нормально"
2. Проверка наличия элемента: if (exists("//author")) then //author else "Автор не указан"
3. С условием по строке: if (//title[contains(., "рецензия")]) then "Обзор" else "Другое"

1. Условные конструкции: обработка отсутствующих данных:

1.1. Простая проверка на отсутствие/наличие

В некоторых статьях на сайте отсутствуют заголовки (в html-коде отсутствует тег `<h1>`). В этом случае, если заголовка нет, мы можем использовать, например, первый абзац как его замену, при этом не затрагивая статьи с заголовками. И в таких случаях обычно не обойтись без условной конструкции. Рассмотрим несколько ф-ций, которые позволят реализовать данную логику:

If(not("//h1)) then //b[@class="lead"] else //h1 (:Если нет тега `<h1>` подставим 1й абзац иначе `<h1>`:)

ЛИБО

If(exists("//h1)) then //h1 else //b[@class="lead"] (:Если есть `<h1>` тогда `<h1>` иначе 1й абзац:)

Теперь в статьях без заголовков будет 1й абзац обрабатываемой статьи (рис.30), а статьи с уже имеющимся заголовком останутся без изменений и не будут затронуты правилом.

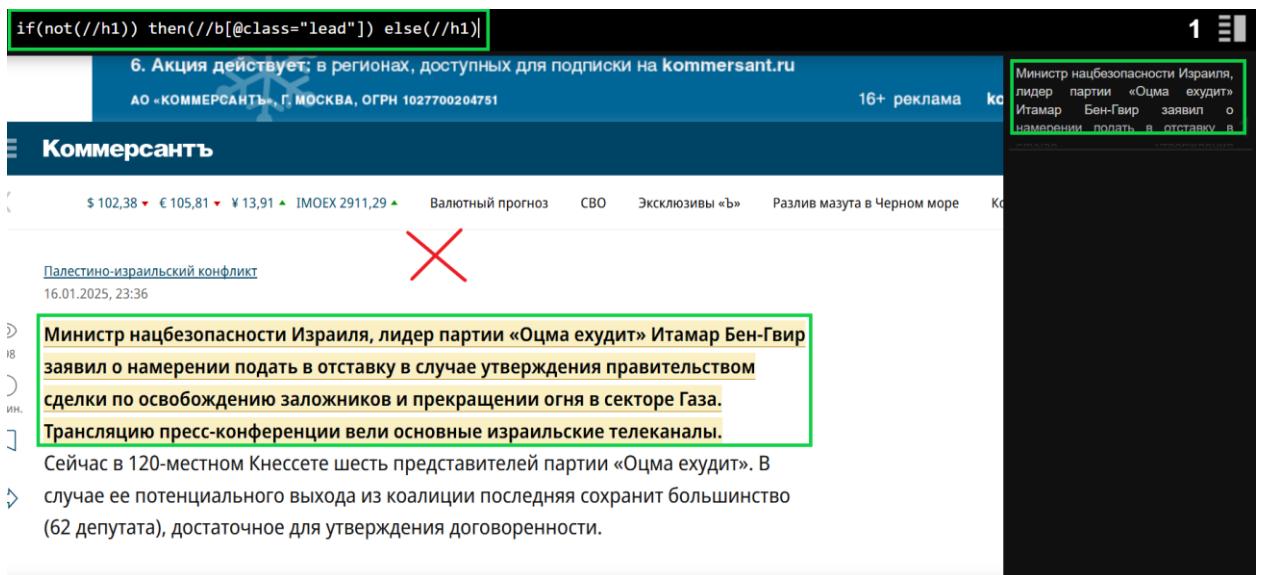


Рисунок 30

1.2. Строгая проверка: пустой или отсутствующий

Если на сайте источника попадаются статьи не только с отсутствующим `<h1>`, но и с пустым `<h1>`, т.е. тег есть, но он без текстового наполнения. В таком случае можем доработать выражение из пп. 1.1.:

```
if(boolean(string(//h1)) = false()) then //b[@class="lead"] else //h1  
if(not(normalize-space(string(//b[@class="lead"])))) then(//h1) else(//b[@class="lead"])
```

Важное отличие от выражения из пп. 1.1., что `boolean(string(//h1)) = false()` учитывает оба случая: тег пустой либо отсутствует.

1.3 Пустой текст или заголовок: подставляем различные элементы

Заголовок + изображение (img):

Часто в статье отсутствует текст, а вместо него только изображение, видео или другой медиаэлемент. В таком случае можно просто подставить заголовок либо дату вместо отсутствующего текста, либо подставить несколько элементов сразу, например, заголовок и изображение (прим.:

<http://sevcspsd.ru/2025/03/27/%D0%BF%D1%80%D0%BE%D0%BA%D1%83%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0-%D0%B3%D0%BE%D1%80%D0%BE%D0%B4%D0%B0-%D1%81%D0%B5%D0%B2%D0%B0%D1%81%D1%82%D0%BE%D0%BF%D0%BE%D0%BB%D1%8F-%D0%B8%D0%BD%D1%84%D0%BE/> ;

```
if(/*[@class="entry-content"])[not(normalize-space())]) then(/h1[@class="entry-title"],  
/*[@class="entry-content"]/figure/img) else(/*[@class="entry-content"])
```

Возможно более предпочтительное использование: if (boolean(normalize-space(/h1[@class="aentry-post_title"]))) then //h1[@class="aentry-post_title"] else /*[@class="aentry-head_date"]

<https://hokkrok.livejournal.com/2590700.html>

Пояснение:

If(...) — если основной текст пустой (содержит только изображение, либо вообще пуст).
then (...) — тогда подставляем заголовок, изображение.
else (...) — иначе если текст непустой, используем его как есть.

Точка + img:

Можно вместо текста, например, подставлять точку + img:

```
if(/*[@itemprop="articleBody"])[not(normalize-space())]) then(string("."),  
//div[@itemprop="articleBody"]//img[@class="thumb"]) else /*[@itemprop="articleBody"]  
(прим.: https://nod.su/871-dd-liz.html )
```

**Есть различные вариации, как заменить пустой или отсутствующий элемент, примеры источников можно поискать в ... -> Ресурсы -> Комментарии (в поле комментарии ввести "Если нет") и поиск выведет разные примеры (рис. 31).*

Рисунок 31

1.4 "Мусор" в основном тексте.

Часто при извлечении основного текста правило "подтягивает" лишние элементы ("мусор") — заголовок, дата, реклама, "подвал" и т.п., и такие элементы попадают в поле "Правила для исключений из текста". Это затрудняет проверку на пустоту: стандартная конструкция not(normalize-space(.)) не сработает, так как текст статьи будет формально непустым.

В таких случаях важно проверить поле "Правила для исключений из текста". Если оно не пустое, необходимо добавить условие для удаления ненужного фрагмента.

Способ удаления зависит от объёма "мусора":

Если он встречается единично (например, отдельный тег, слово или фраза), то достаточно обычного сравнения в условии.

Если же мусорных фрагментов несколько, могут понадобиться дополнительные функции, например concat().

Единичный "мусор":

Если статья пустая (без текста), но правило извлечения текста включает в себя какой-либо одиночный мусорный элемент. В примере это "подвал" статьи с тегом `<aside>`, в котором располагается ненужный текст (рис. 32).



Рисунок 32

Один из вариантов решения, это сравнить содержимое с заведомо "мусорным" блоком:

```
if (not(normalize-space(//*[@class="page_content type_page_content"]))) or normalize-space(//*[@class="page_content type_page_content"]) = normalize-space(//aside) then
  //*[@class="page_content type_page_content"]/preceding::header/h1 else
  //*[@class="page_content type_page_content"]
```

Разбор по пунктам:

`not(normalize-space(...))` — проверяем, что текст пустой

`normalize-space(...) = normalize-space(//aside)` — путем сравнения проверяем, что в тексте только "мусор" (в данном случае `<aside>`). Т.е. текст состоит только из содержимого тега `<aside>`

`then ...` — если текст пустой или состоит только из "мусора", подставляем заголовок `<h1>`.

`else ...` — иначе используем текст как есть `//*[@class="page_content type_page_content"]`

ЕСЛИ текст состоит только из одного слова или короткой фразы вроде:

"Источник", "Возврат к списку" и т.п.:

Применим аналогичную конструкцию, как в предыдущем примере, но сравним основной текст `<div[@class="body"]>` с тегом `<a>` в котором есть конкретное словосочетание — "Возврат к списку":

```
if (not(normalize-space(//div[@class="body"])) or normalize-space(//div[@class="body"]) = //a[normalize-space(.) = "Возврат к списку"]) then //h1 else //div[@class="body"]
```

Неразрывный пробел ():

Иногда встречаются статьи, которые выглядят пустыми, но правило `not(normalize-space(...))` на них не срабатывает. Это происходит потому, что в тексте содержатся неразрывные пробелы (), которые `normalize-space()` не удаляет. Увидеть данные элементы можно в коде страницы (рис. 33).

Чтобы корректно обработать такие случаи, нужно заменить все пробельные символы, включая неразрывные, на пустую строку с помощью `replace(...)`, и только потом проверить результат.



```
<div class="article_wrap"> == $0
  <a href="/news/foto/2019/1" target="_blank">
    
  </a>
  &nbsp; &nbsp;
  > <a href="/news/foto/2019/2" target="_blank">...</a>
</div>
</div>
```

Рисунок 33

Вместо `not(normalize-space(...))` используем `replace()`:

```
if (//div[@class="article_wrap"] [not(replace(., '[\s\u00A0]', ''))] ) then (//h1[@class="h2"], //div[@class="article_wrap"] //img) else (//div[@class="article_wrap"])
```

Разбор: [not(replace(., '\s\u00A0', ''))] - используется регулярное выражение [\s\u00A0], которое означает - удалить все пробельные символы (\s) и неразрывные пробелы (\u00A0, Unicode-код).

<https://belmedpreparaty.com/news/patsientam/pamyatka-avtolyubitelyu/>

<https://www.omsvrn.ru/content/voronezhcam-rasskazali-komu-pokazano-uzi-shei-po-oms>

<https://govoritadygea.ru/index.php?newsid=5>

Вырезание все без исключения пробелов:

```
if( //div[@class="tmpl-contentblock"] [not(normalize-space(translate(., ' ', '')))] or
normalize-space(translate(//div[@class="tmpl-contentblock"], ' ', ' ')) = normalize-
space(translate(//p[@class="cms_matdate"], ' ', ' ')) ) then ((//h1)[1],
//p[@class="cms_matdate"]/following-sibling::p//img) else ((//div[@class="tmpl-
contentblock"])
```

Сравнение в правильном порядке:

```
if(normalize-space(string(//section[@id="text"])) = normalize-
space(concat(string(//h1[@id="page-title"]), ' ', string(//div[@class="news-view-item-story-
date"]))) then(string("."), //a[@class="op-image"]/img, //div[@class="video"])
else(//section[@id="text"]))
```

Поля новости:

URL:	https://library-news.view/31833/
Заголовок:	ПОЗДРАВЛЕНИЕ ГУБЕРНАТОРА А. В. БОГОМАЗА С 81-Й ГОДОВЩИНОЙ ОСВОБОЖДЕНИЯ БРЯНСКОГО КРАЯ ОТ НЕМЕЦКО-ФАШИСТСКИХ ЗАХВАТЧИКОВ И ДНЕМ Г. БРЯНСКА
Дата и время:	17.09.2024 10:17:14
Автор:	
Номер выпуска:	
Текст(включая изображения по тексту):	
<p>ПОЗДРАВЛЕНИЕ ГУБЕРНАТОРА А. В. БОГОМАЗА С 81-Й ГОДОВЩИНОЙ ОСВОБОЖДЕНИЯ БРЯНСКОГО КРАЯ ОТ НЕМЕЦКО-ФАШИСТСКИХ ЗАХВАТЧИКОВ И ДНЕМ Г. БРЯНСКА 17.09.2024</p> <p>Г. 17.09.2024 ПОЗДРАВЛЕНИЕ ГУБЕРНАТОРА А. В. БОГОМАЗА С 81-Й ГОДОВЩИНОЙ ОСВОБОЖДЕНИЯ БРЯНСКОГО КРАЯ ОТ НЕМЕЦКО-ФАШИСТСКИХ ЗАХВАТЧИКОВ И ДНЕМ Г. БРЯНСКА</p>	
Изображения вне текста:	
Медиа-контент вне текста:	

Сравнение по длине строки: string-length(normalize-space(string(//section[@id='text']))) = string-length(normalize-space(concat(string(//h1[@id='page-title']), ' ', string(//div[@class='news-view-item-story-date']))))

Пример сравнения текста содержащим только неразрывный пробел:

```
if(boolean(string(//div[@class="text-content bvi-speech"])) = false() or normalize-
space(//div[@class="text-content bvi-speech"]) = normalize-space(codepoints-to-string(160)))
```

```
then(string("."), //div[@class="news_page_img img pic"]/img, //p[iframe])
else("//div[@class="text-content bvi-speech"]")

if(not(normalize-space(translate(//*[@class='news-item-text clearfix'], codepoints-to-
string((160,5760,8192,8193,8194,8195,8196,8197,8198,8199,8200,8201,8202,8203,8232,8233,
8239,8287,12288,65279,8204,8205,8206,8207,8288)), ''))) then(//div[@class='news-item']/h1,
//*[@class='news-item-text clearfix']//img) else(//*[@class='news-item-text clearfix'])
```

- `//*[@class='news-item-text clearfix']`

Берём содержимое блока с текстом новости.

- `translate(..., codepoints-to-string(...), '')`

Заменяем на пустую строку все «проблемные» невидимые символы:

- неразрывный пробел 160 (U+00A0)
- разные типы пробелов (U+2000–U+200B, U+202F, U+3000 и др.)
- управляющие знаки (65279 — BOM, 8204–8207 — LRM/RLM и т.д.).

- `normalize-space(...)`

Убирает обычные пробелы, табы, переносы строк по краям и сводит внутренние пробелы к одному.

→ Если текст был пустой или состоял только из «мусора» (в том числе NBSP), то результат будет пустая строка.

- `not(...)`

Проверка: пусто ли поле после очистки.

- `if(...) then(...) else(...)`

- Если пусто → берём заголовок (`//div[@class='news-item']/h1`) и картинку.
- Если не пусто → берём текст из блока.

Несколько мусорных элементов (множественный "мусор"):

Если же в "пустом" тексте мусорных элементов несколько, т.е. в правило извлечения текста могут попадать пробелы, дата, навигационная панель, заголовок и пр (Рис). То используем `concat()` для того, чтобы объединить весь мусор в единый текст и сравнить его с основным правилом извлечения текста.

The screenshot shows the Firebug developer tool interface. The left panel displays the HTML structure with a red box highlighting the main content area. The right panel shows the CSS styles applied to the selected element, with a specific rule for '.tmpl-contentblock' highlighted.

```

QUERY
//div[contains(@class, 'tmpl-contentblock')]/text()

RESULTS (4)
Извещение о проведении в 2026 году на территории Вологодской области. Государственная кадастровая оценка земельных участков
27 марта 2025
Дата создания: 27-03-2025
Извещение о проведении в 2026 году на территории Вологодской области. Государственной кадастровой оценки земельных участков
27 марта 2025
Извещение

```

```

<div class="tmpl-contentblock clearfix"> == $0
  ::before
    <p style="text-align:right" class="cms_matdate">
      <b>27 марта 2025</b>
    </p>
  <p style="text-align:center;">
    <a href="/tinybrowser/fulls/images/news/2025/05/izveschenie_o_kadastrovoy_ocenke_zemelnyh_uchastkov_page-0001.jpg" rel="prettyPhoto[tinybrowser/images/news/2025/05]" title="Извещение о проведении в 2026 году на территории Вологодской области. государственной кадастровой оценки земельных участков">
      
    </a>
  </p>
  <div class="editlog">Дата создания: 27-03-2025</div>
</div>
</div>
<div class="after">
</div>

```

Рисунок 34

Задача: Определить, есть ли в основном блоке полезный текст или только "мусор".

Цель примера: При отсутствии полезного текста подставить заголовок с изображением.

Что делает конструкция: Сравнивает содержимое блока с объединённым "мусором".

Если совпадает или блок пуст — делает замену.

Решение:

```

if (
  //div[contains(@class, 'tmpl-contentblock')] [not(normalize-
space())]
  or
  normalize-space(//div[contains(@class, 'tmpl-contentblock')])
=
  normalize-space(concat(//p[@class='cms_matdate'],
  //div[@class='editlog']))
)
then (
  //div[@class='maintitle'],
  //div[contains(@class, 'tmpl-contentblock')] //img
)
else (
  //div[contains(@class, 'tmpl-contentblock')]
)

```

<http://xn----7sbqb4asd9a8d.xn--p1ai/izveshenie-o-provedenii-v-2026-godu-na-territoriyi-vologodskoy-oblasti-gosudarstvennoy-kadastrovoy-otcenki-zemel-nykh-uchastkov.html>

<https://xn--34-6kcadiv3bzakcsr.xn--p1ai/%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D0%BA%D0%B0-%D0%BF%D0%BE-%D1%81%D1%83%D0%B1%D1%81%D0%B8%D0%B4%D0%B8%D1%8F%D0%BC/>

1.5. Вложенное условие: подстановка при отсутствии текста и заголовка

Иногда на странице отсутствует и основной текст, и заголовок. Чтобы всё же извлечь полезную информацию, можно использовать вложенные условия — if внутри блока then.

Логика:

- если отсутствует текст
 - и если при этом отсутствует заголовок,
 - используем дату публикации и изображение;
 - иначе — заголовок и изображение;
- если текст есть — используем его напрямую.

Пример:

```
if (not(normalize-space(//*[@class='entry-content'])))  
then  
    if (not(normalize-space(//h1[@class='entry-title'])))  
        then (//p[@class='post-meta']/span[@class='published'] ,  
        //p[@class='post-meta']/following-sibling::img)  
    else (//h1[@class='entry-title'] , //p[@class='post-  
meta']/following-sibling::img)  
else (//*[@class='entry-content'])
```

Пояснение:

entry-content — основной текст;

entry-title — заголовок;

published — дата публикации;

img — изображение, идущее сразу после метаданных.

<https://xn--34-9kcp5akho5d3bm.xn--p1ai/> ;

exists() с вложенным условием в else()

```
if (exists(//div[@class="info-row-properties-articles-down"]/span[contains(., '.20') and string-  
length(normalize-space(.)) = 10]))  
then
```

```

//div[@class="info-row-properties-articles-down"]/span[contains(., '.20') and string-
length(normalize-space(.)) = 10]
else
  if (exists("//div[@class='info-row-article-properties-down']/span[contains(., '.20') and string-
length(normalize-space(.)) = 10])) 
    then
      //div[@class="info-row-article-properties-down"]/span[contains(., '.20') and string-
length(normalize-space(.)) = 10]
    else
      //span[@class="info-row-properties-bottom-line-video-page"]

```

2. Все блоки пустые: every + if-then-else

Конструкция every + if-then-else используется, когда нужно проверить каждое из нескольких XPath-правил на отсутствие текста. Например, на сайте источника правила извлечения текста, в одной статье `//*[@class="itemFullText"]`, в другой `//div[@class="newscontainer-itemIntroText itemIntroText"]`, в третьей `//div[@class="newscontainer-itemFullText itemFullText"]` и т.д. В таких случаях нужно проверить каждый из этих блоков на пустоту (т.е. не содержат основного текста статьи), прежде чем подставлять альтернативу — заголовок и изображение.

<https://rossony.vitebsk-region.gov.by/novosti-regiona> ;

Задача: Обработать случаи, когда в источнике используются разные правила для извлечения текста и каждое нужно проверить на пустоту с помощью every.

Цель: Если полезный текст отсутствует, то подставить заголовок и изображение.

Что делает конструкция: Проверяет, все ли указанные текстовые блоки пустые. Если да — возвращает заголовок и img; если нет — основной текст из любого непустого блока.

```

if (
  every $x in (
    //*[@class="itemFullText"],
    //div[@class="newscontainer-itemIntroText itemIntroText"],
    //div[@class="newscontainer-itemFullText itemFullText"]
  )
  satisfies not(normalize-space($x))
)
then (

```

```

//h2[@class='newscontainer-itemTitle contentheading'],
//div[@class="newscontainer-itemFullText itemFullText"]/p/img
)
else (
    //div[contains(@class, 'itemFullText') or contains(@class,
'itemIntroText')]
)

```

<https://edu-nv.ru/news/25-novosti/9963-s-prazdnikom>

```

if (
    every $x in (
        /*[@class="itemIntroText"],
        /*[@class="itemFullText"]
    )
    satisfies not(normalize-space($x))
)
then (
    /*[@class="itemTitle"],
    //div[@class="news_itemImageBlock"]//img
)
else (
    /*[@class="itemIntroText"], /*[@class="itemFullText"])

```

Итерация for

В XPath 2.0 выражение **for** позволяет перебрать последовательность значений или узлов и выполнить заданное выражение для каждого элемента, возвращая результирующую последовательность. Это одна из основных возможностей языка для создания динамических запросов.

Синтаксис:

Найдем все элементы `<a>` и умножим их содержимое на 2 (сработает только с цифрами):

```

<div>
    <a>1</a>
    <a>2</a>
    <a>3</a>
</div>

```

```
for $x in //a return $x * 2 -> результат: 2,4,6
```

Разбор:

for – ключевое слово, которое инициирует итерацию по элементам последовательности.
Аналогия: "Для каждого элемента в последовательности выполнить..." .

\$x – имя переменной, в которое, как правило, помещается XPath-путь(//a). Имя переменной должно начинаться с \$ и может содержать буквы, цифры и некоторые символы. Лучше использовать понятные имена, например: \$link, \$head и т.п., но обычно это просто \$x.

In – ключевое слово, которое связывает переменную с последовательностью.

return - ключевое слово, после которого описывается что сделать с текущим элементом.

Другие примеры:

```
for $n in (1, 2, 3) return concat("Число ", $n) -> результат: "Число 1", "Число 2", "Число 3"
```

```
for $y in ("apple", "banana", "cherry") return upper-case(substring($word, 1, 3)) ->  
результат: "APP", "BAN", "CHE"
```

Конструкция for часто используется вместе с условием if-then-else. Например, если в наборе данных отсутствуют такие элементы, как заголовки или даты, то можно с помощью for и условия подставить вместо отсутствующего заголовка дату или любой другой элемент. Т.е. если из рубрики опроса берутся 10 статей и некоторых из них отсутствуют заголовки, то вместо отсутствующих заголовков будет подставлена дата, а в тех статьях, где заголовки присутствуют, ничего меняться не будет (подстановка даты не производится).

Важно учитывать, что замена множества элементов (например, в рубрике опроса) отличается от замены одиночного элемента (заголовка, текста или даты в одной статье). В случае одиночного элемента можно использовать просто if-then-else, хотя это не всегда удобно, т.к. если XPath-путь слишком длинный, его приходится повторять в разных частях условия, тогда как в for можно связать XPath-путь с переменной.

1. Извлечение данных из множества строк с помощью FOR

1.1. В рубрике опроса необходимо получить даты, расположенные непосредственно в строках с заголовками (прим.: "01.03.2025 | Условия бронирования круизов-2025 и персональные льготы на теплоходах ТК «ВолгаWolga» с 1 по 31 марта 2025 г." и др... (рис. 35))

The screenshot displays a grid of news items from the VolgaWolga.ru website. Each item is a card with a green header containing the year '2025' and the title 'УСЛОВИЯ БРОНИРОВАНИЯ И ЛЬГОТЫ'. Below the title is a list of names: '«Владимир Маяковский», «Борис Поповой», «Юрий Никулин» и «Михаил Кутузов», «Павел Бажов», «Александр Невский»'. The cards also mention 'круизы-2025 на теплоходах'. The dates listed are: 01.03.2025, 01.02.2025, 16.01.2025, 28.12.2024, 01.12.2024, and 16.11.2024. A central image shows a man and a woman standing together, holding a framed certificate.

Рисунок 35

Если попытаться извлечь элементы с помощью функции `substring()` по выражению:

`substring("//span[@class='h4']/a[contains(@href,'/novosti/')], 1, 10)` – берем первые 10 слов (дата в строке), то оно не сработает (рис. 36), поскольку данное выражение рассчитано на один элемент, а не на множество.

```
substring("//span[@class='h4']/a[contains(@href, "/novosti/")], 1, 10)|
```

Invalid XPath

Рисунок 36

Чтобы правило применялось ко всем элементам множества, используется конструкция `for`:

`for $x in //span[@class='h4']/a[contains(@href,'/novosti/')] return substring($x, 1, 10)` (рис. 37)

Рисунок 37

1.2. Правильно достаем закриптованные ссылки:

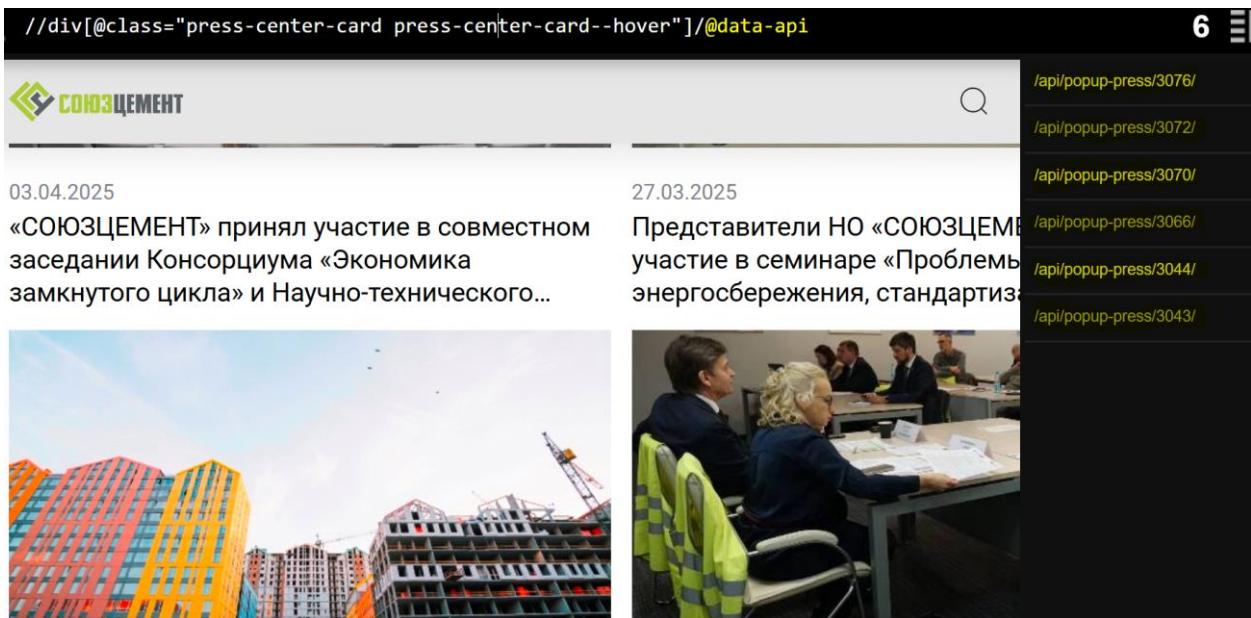


Рисунок 38 (прим.: <https://soyuzcem.ru/press-center/novosti/> актуальность ссылки 15.04.2025;)

На рис. 38 показаны «заскриптованные» ссылки (@data-api) иногда авторы сайта используют их вместо стандартного в таком случае использование concat() без for не сработает:

```
concat("https://soyuzcem.ru/press-center/novosti/?id=", substring("//div[@class='press-center-card press-center-card-hover']/@data-api, string-length("//div[@class='press-center-card press-center-card-hover']/@data-api) - 4))
```

Используя concat() + for правило сработает (рис. 39):

```
for $x in //div[@class="press-center-card press-center-card--hover"]/@data-api return concat("https://soyuzcem.ru/press-center/novosti/?id=", substring($x, string-length($x) - 4))
```

Ссылка	Статус
https://soyuzcem.ru/press-center/novosti/?id=3076	OK - Правило 1
https://soyuzcem.ru/press-center/novosti/?id=3072	OK - Правило 1
https://soyuzcem.ru/press-center/novosti/?id=3070	OK - Правило 1
https://soyuzcem.ru/press-center/novosti/?id=3066	OK - Правило 1
https://soyuzcem.ru/press-center/novosti/?id=3044	OK - Правило 1

Рисунок 39 (прим.: <https://soyuzcem.ru/press-center/novosti/> актуальность ссылки 15.04.2025)

2. Обработка альтернативных сокращений названий месяцев

2.1 Правило "бтр". * В случаях, когда нужно заменить название месяцев (нояб., сент., февр.) на стандартные форматы (ноя., сен., фев.) также применим выражение for + условие. Данный случай относительно частый при починке источников

Допустим, у нас есть следующий HTML-код:

```
<div class="link">
    <a href="#">февр.</a>
    <a href="#">янв.</a>
    <a href="#">декабрь</a>
    <a href="#">нояб.</a>
    <a href="#">окт.</a>
    <a href="#">сент.</a>
</div>
```

Применим XPath-выражение:

```
for $x in //div[@class="link"]/a return if (matches($x, "(нояб|сент|февр)."))
then translate($x, "бтр", "") else $x
```

Разбор выражения:

for \$x in //div[@class="link"]/a - перебирает все элементы <a>

matches(\$x, "(нояб|сент|февр).") - проверяет совпадение, содержит ли текст элемента указанные сокращения месяцев, разделенных точкой (функция matches() позволяет использовать регулярку для краткости).

Если совпадение найдено, то translate(\$x, "бтр", "") удаляет буквы "б", "т", "р".

"нояб." → "ноя."; "сент." → "сен."; "февр." → "фев."

Для остальных любых значений (например, "окт.", "декабрь", "янв." и т.п.) применяется else \$x, и они остаются без изменений, работает исходное основное правило //div[@class="link"]/a.

Итоговый результат: "сен.", "окт.", "ноя.", "декабрь", "янв.", "фев."

2.2. Или ситуация, когда авторы пишут название месяцев на "свое усмотрение" (рис. 40)

То в таком случае как раз подойдет условная конструкция с дополнительным условием else if:

```
for $x in //div[contains(@class,"articleAttributes")]/time return if (contains ($x, "Мрт")) then replace($x,
"Мрт", "Мар") else if (contains($x, "Нбр")) then replace($x, "Нбр", "Ноя") else $x
```

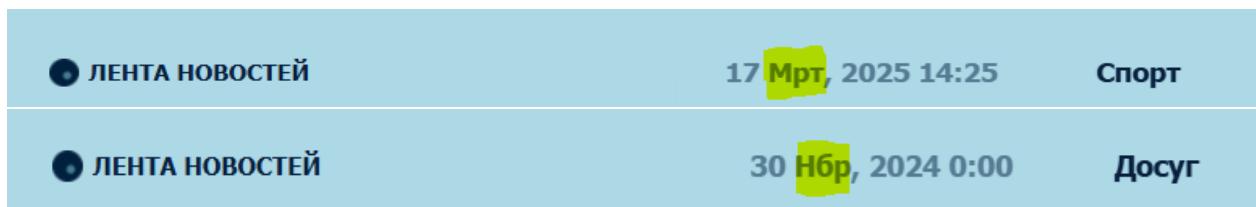


Рисунок 40 (прим.: <https://oratert.am/article/105484/ru> актуальность ссылки 15.04.2025;)

* Обычно ошибки встречаются только в названиях трех месяцев: "нояб", "сен" и "февр". Удаляя последний символ в каждом из этих слов, мы получаем правильные сокращения: "ноя", "сен" и "фев". Интересно, что в первых трех буквах этих названий отсутствуют буквы "б", "т", "р", что делает исправление безошибочным.

2.3. Также возможен вариант объединения случаев в пп. 2.1 и 2.2 (если в источнике встречаются оба варианта неправильных сокращений: "нояб", "сент", "февр", "март", "нбр"), рассмотрим на примере того же источника (рис. 40):

Для наглядности распишем выражение с отступами:

```
for $x in //div[contains(@class, "articleAttributes")]/time
return
if (matches($x, "(нояб|сент|февр)."))
then
translate($x, "бтр", "")
else if (matches($x, "Март"))
then
replace($x, "Март", "Мар")
else if (matches($x, "Нбр"))
then
replace($x, "Нбр", "Ноя")
else
$x
```

3. Замена пустых значений:

Подставить дату если нет заголовка (рис. 41):

Активное долголетие – Здоровая Рязань	19.03.25	Всероссийская акция "Музыка на все времена"	19.03.25
Сообщаем вам, что вы можете стать участниками проекта «Активное долголетие – Здоровая Рязань». Это региональный оздоровительно-образовательный проект для граждан старшего возраста. [...]	?	С 25 марта по 15 апреля 2025 г. в рамках межведомственного культурно-образовательного проекта «Культура для школьников» пройдет Всероссийская акция «Музыка на все времена».	?
Активное долголетие – Здоровая Рязань	17.03.25	Дни Защитника Отечества в театре: мужество и сила духа на сцене и за её пределами	13.03.25
Сообщаем вам, что вы можете стать участниками проекта «Активное долголетие – Здоровая Рязань». Это региональный оздоровительно-образовательный проект для граждан старшего возраста. [...]	?	С 25 марта по 15 апреля 2025 г. в рамках межведомственного культурно-образовательного проекта «Культура для школьников» пройдет Всероссийская акция «Музыка на все времена».	?
Активное долголетие – Здоровая Рязань	13.03.25	Дни Защитника Отечества в театре: мужество и сила духа на сцене и за её пределами	13.03.25
Сообщаем вам, что вы можете стать участниками проекта «Активное долголетие – Здоровая Рязань». Это региональный оздоровительно-образовательный проект для граждан старшего возраста. [...]	?	С 25 марта по 15 апреля 2025 г. в рамках межведомственного культурно-образовательного проекта «Культура для школьников» пройдет Всероссийская акция «Музыка на все времена».	?
Активное долголетие – Здоровая Рязань	21.02.25	Замена и отмена спектаклей в феврале 2025	20.02.25
Сообщаем вам, что вы можете стать участниками проекта «Активное долголетие – Здоровая Рязань». Это региональный оздоровительно-образовательный проект для граждан старшего возраста. [...]	?	Директор Рязанского театра драмы вернулся из рабочей поездки в Иран	?
Активное долголетие – Здоровая Рязань	04.02.25	"Знатоки женского сердца, февраль"	04.02.25
Сообщаем вам, что вы можете стать участниками проекта «Активное долголетие – Здоровая Рязань». Это региональный оздоровительно-образовательный проект для граждан старшего возраста. [...]	?	"Знатоки женского сердца, повтор"	04.02.25
Активное долголетие – Здоровая Рязань		"Знатоки женского сердца"	11.02.25

Рисунок 41 (Прим.: https://www.rzndrama.ru/ru/rasskazyvaem/novosti.html?ru_start=0 актуальность ссылки 15.04.2025)

for \$x in //a[h6] return if(\$x[not(normalize-space())]) then \$x/preceding::div[@class="date_bg"] else \$x
Пояснение:

- **\$x** – имя переменной, в которую помещается XPath-путь (//a[h6])
- **return** – ключевое слово, после которого описывается что сделать с текущим элементом.
- **if(\$x/h6[not(normalize-space())])** - для каждого элемента \$x (из выборки //a[h6]) проверяется, содержит ли его дочерний элемент <h6> непустой текст (после удаления лишних пробелов).

- **then** \$x/preceding-sibling::a[1] если `<h6>` оказывается пустым, то вместо текущего элемента возвращается ближайший предыдущий элемент `<a>` (\$x/preceding-sibling::a[1]). **Ключевой момент:** в данной части условия именно явное использование переменной \$x обеспечивает корректное определение отправной точки для поиска. Если указать просто путь до даты (`//div[@class="date_bg"]`), то соответствующие элементы могут многократно дублироваться (рис. 42):

for \$x in //a[h6] return if(\$x[not(normalize-space())]) then //div[@class="date_bg"] else \$x
- else \$x в противном случае возвращается сам элемент \$x (когда заголовок "в порядке")



Рисунок 42 Многократное дублирование

4. Корректировка дат при интерпретации будущим годом:

Если в рубрике опроса дата указана в формате dd ММММ, и это приводит к ошибочной интерпретации будущей датой (например, текущая дата — январь 2025, и система воспринимает прошлогоднюю дату "10 декабря" как декабрь 2025 — "Скорректирована дата в будущем"), то для устранения ошибки можно принудительно подставить предыдущий год (2024) для нужных месяцев. Т.о. если нужно загрузить много статей, к примеру, из архива, то **правило подойдет для поля - "Правила извлечения ссылок пейджинга"**.

Из статьи: if((//div[@class="news-detail__content-header-data"])[1][contains(., "декабр")]) then(replace((//div[@class="news-detail__content-header-data"])[1], '.*?(\d{1,2} [а-яА-Я]+).*', '\$1 2024')) else(replace((//div[@class="news-detail__content-header-data"])[1], '.*?(\d{1,2} [а-яА-Я]+).*', '\$1'))

Если дата содержит слово "декабрь", то добавляем к дню и месяцу предыдущий год (2024), чтобы явно указать прошлый год. Иначе получим "Скорректирована дата в будущем".

Из рубрики опроса (архив на 3 месяца): for \$x in //div[@class="news-item__date"] return if(\$x[matches(., "декабр|ноябр|октябр")]) then(replace(\$x, '.*?(\d{1,2} [а-яА-Я]+).*', '\$1 2024')) else(replace(\$x, '.*?(\d{1,2} [а-яА-Я]+).*', '\$1'))

Проверяются даты за последние три месяца. Если найдены месяцы "октябрь", "ноябрь" или "декабрь", к дате добавляется 2024. Для остальных — только день и месяц (рис.43).

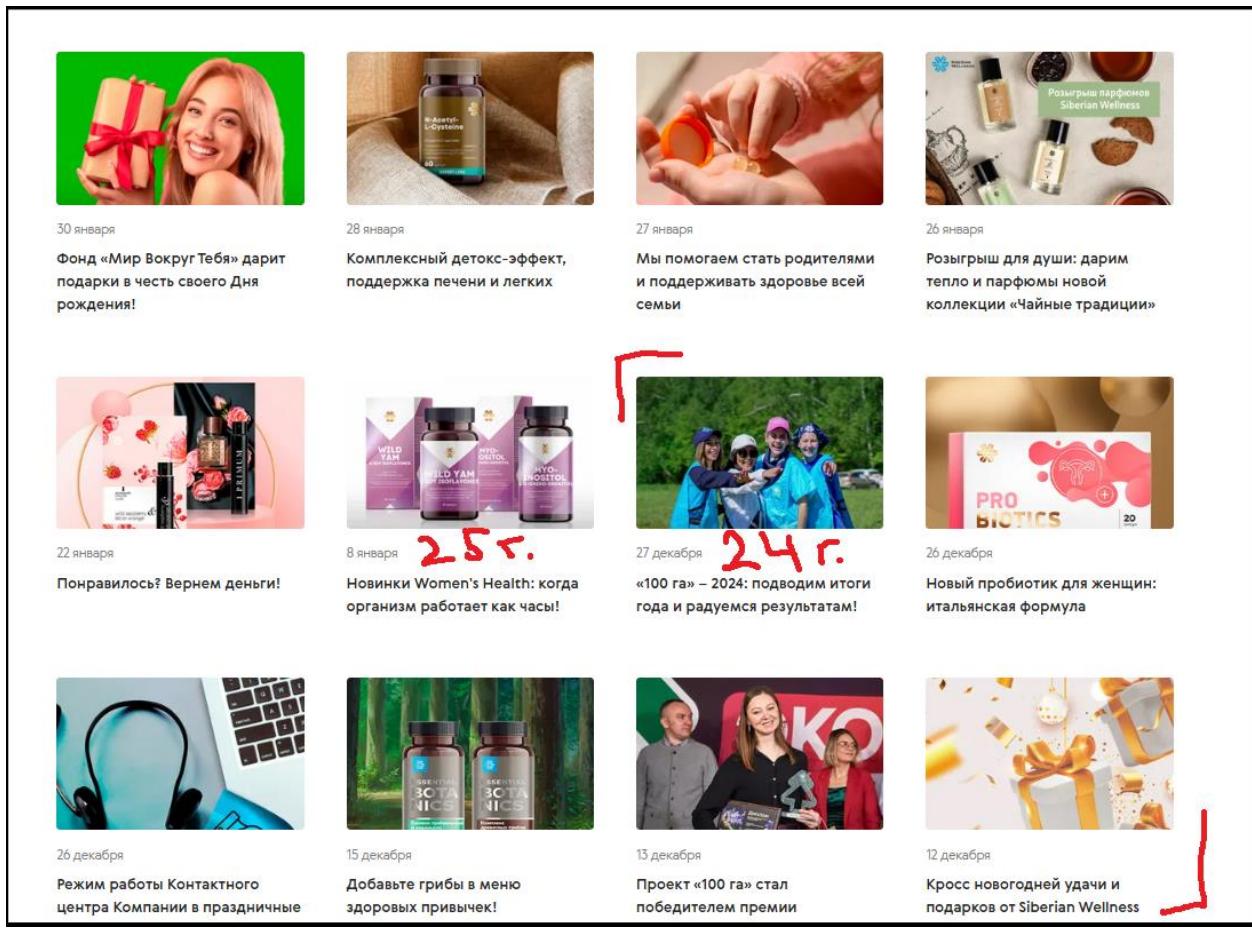


Рисунок 43 <https://siberianhealth.com/ru/news> ;

Обе конструкции предотвращают автоматическую подстановку будущего года системой.

Шаблоны и другие возможности XPath:

Точный поиск дат (первая группа захвата \$1):

*все даты указаны для примера (подставить можно любую):

"**18 марта 2024**" replace("//time, '.*?(\d{1,2} [а-яА-Я]+ \d{4}).*', '\$1')

"**18 марта 2024 17:20**" replace("//time, '.*?(\d{1,2} [а-яА-Я]+ \d{4} \d{2}(:\d{2}){1,2}).*', '\$1')

"**11:27, 11 ноября 2024**" replace("//time, '.*?(\d{1,2}:\d{2}, \d{2} [а-яА-Я]+ \d{4}).*', '\$1')

"**1 мая 2024, 15:18**" replace("//time, '.*?(\d{1,2} [а-яА-Я]+ \d{4}, \d{2}:\d{2}).*', '\$1')

"**1 мая 2024 в 15:18**" replace("//time, '.*?(\d{1,2} [а-яА-Я]+ \d{4} в \d{2}:\d{2}).*', '\$1')

"**16 мая, 2023 11:32**" replace("//time, '.*?(\d{1,2} [а-яА-Я]+,? \d{4} \d{2}:\d{2}).*', '\$1')

"**8 января, 2025**" replace("//div[@class='postmetadata'], '.*?(\d{1,2} [а-яА-Я]+,? \d{4}).*', '\$1')

"**11 ноября**" - replace("//time, '.*?(\d{1,2} [а-яА-Я]+).*', '\$1')

"**10 сен 2020**" - replace("//time, '.*?(\d{1,2} [а-яА-Я]{3} \d{4}).*', '\$1')

"**5 мар. 2015**" - replace("//time, '.*?(\d{1,2} [а-яА-Я]{3}\. \d{4}).*', '\$1')

"**Мар 3, 2025**" - replace("//div[@class='entry-meta'], '.*?([А-Яа-я]{3} \d{1,2}, \d{4}).*', '\$1')

***если даты на английском, то достаточно поменять а-яА-Я на а-zA-Z, либо можно сразу прописывать а-яА-Яa-zA-Z**

"**30.10.2023**" replace("//time, '.*?(\d{1,2}\.\d{2}\.\d{4}).*', '\$1')

"**8.12.2023**" replace("//time, '.*?(\d{1,2}\.\d{2}\.\d{4}).*', '\$1')

"**18:05 13.05.2025**" replace("//time, '.*?(\d{2}:\d{2}\.\d{2}\.\d{4}).*', '\$1')

"**08.10.2024 12:10**" replace("//time, '.*?(\d{1,2}\.\d{2}\.\d{4} \d{2}:\d{2}).*', '\$1')

"08.10.2024, 12:10" replace("//time, '.*?(\d{1,2}\.\d{2}\.\d{4}, \d{2}:\d{2}).*', '\$1')
"08.10.2024 12:10:08" replace("//time, '.*?(\d{1,2}\.\d{2}\.\d{4} \d{2}:\d{2}:\d{2}).*', '\$1')
"08.10.2024, 12:10:08" replace("//time, '.*?(\d{1,2}\.\d{2}\.\d{4}, \d{2}:\d{2}:\d{2}).*', '\$1')
"28.10.2024 в 06:29" replace("//time, '.*?(\d{1,2}\.\d{2}\.\d{4} в \d{2}:\d{2}).*', '\$1')
"05.11.2024 - 12:27:17" replace("//time, '.*?(\d{2}\.\d{2}\.\d{4} - \d{2}:\d{2}:\d{2}).*', '\$1')
"09/12/2024" replace("//time, '.*?(\d{1,2}/\d{1,2}/\d{4}).*', '\$1')
"04/07/2025 - 16:26" replace("//span[@class='submitted-by']/strong, '.*?(\d{2}/\d{2}/\d{4} - \d{2}:\d{2}).*', '\$1')
Если вместо пробелов управляющие символы используем Unicode: \u00A0
понедельник\u00A011:54, \u00A0декабря\u00A02024
"18 марта 2024" - replace("//time, '.*?(\d{1,2}[\s\u00A0][а-яА-Я]+[\s\u00A0]\d{4}).*', '\$1')
"11:27, 11 ноября 2024" - replace("//time, '.*?(\d{1,2}:\d{2}, [\s\u00A0]\d{1,2}[\s\u00A0][а-яА-Я]+[\s\u00A0]\d{4}).*', '\$1')

Первые 3 слова из строки ("15 января 2024 <любой текст>"):

```
string-join(tokenize(normalize-space("//div), "\s+")[position() <= 3], " ")
```

Последние 3 слова из строки ("<любой текст> 18 февраля 2024")

```
for $x in normalize-space("//div) return string-join(subsequence(tokenize($x, '\s+'), (count(tokenize($x, '\s+')) - 2)), ' ')
```

Вырезаем любые слова, любого регистра (большие, маленькие буковки), дополнять список можно через палочку "|" означающую "ИЛИ":

```
//div[matches(normalize-space(), '^(\ фото | видео)$', 'i')]
```

Извлекаем одним правилом названия классов, которые, "посредине", немного отличаются между собой: //a[@class="hotNewsSm-title"] и //a[@class="hotNews-title"];

```
//a[matches(normalize-space(@class), '^hotNews.*-title$')]
```

Поиск строки состоящей ровно из 3 слов разделенных пробелом:

```
//time[matches(., '^\s*\s*\s*$')]
```

Выбираем/заменяем все элементы, у которых текст содержит только цифровое значение:

```
//*[number() = number()] - целые числа и с плавающей запятой  
//*[not(translate(., "0123456789", ""))] - целые числа (используем not() для обратного значения)  
//*[matches(., "^\d+$")] - целые числа  
//*[matches(., "^\d+(\.\d+)?$")] - целые числа и с плавающей запятой  
//*[matches(., '\s*\d+\s*$')] - целые числа игнорируя пробелы (по типу normalize-space())  
replace("//div, ['^\d'], "") - целые числа  
replace("//h1, ['^\w\d'], "") - достать только цифры
```

Поиск/замена элементов, которые содержат/не содержат определенный текст/пустые (русский, английский, цифры, а также регистр):

```
//h2[a]/text()[not(translate(*, "абвгдеёжзийклмнопрстуфхцчшщъыъэ়য়", ""))] - выбирает все элементы на странице, у которых текст не содержит русские буквы в нижнем регистре.
```

`//*[matches(., "[а-яА-Я]")]` - содержит русские буквы в нижнем и верхнем регистрах, однако использует функцию matches(), которая позволяет использовать регулярные выражения для поиска.

`//*[not(matches(., "[а-яА-Яа-зА-З0-9]"))]` - не содержит цифры, русские и английские буквы во всех регистрах.

`//a[@class="post-url post-title"] [not(matches(., "[ଆଜିତକାଳେରେଖାରୁଦ୍ଧାରୀଙ୍କବଶକସହ])]` - язык ория не читается системой и добавим их алфавит в исключение поиска ссылок

`//*[matches(name(), '^h[1-6]$')]` - найти все теги h1 - h6

`replace("//h1, '[^ଅ-ୟା-ୟ]', ' ')` - буквы рус. Алфавита

`matches("//div, '[\w\s]')` – спецсимволы

`/p[normalize-space(replace(., ' ', ' ')) = 'Фотографии:']` – проверка с учетом неразрыв пробелов

Поиск на пустоту/пробелы:

`//*[not(normalize-space(text()))]` - не содержит текст в определенном теге

`//*[not(normalize-space(.))]` - действует аналогично предыдущему примеру, но отличается тем, что проверяет всю глубину вложенности, а не определенный тег

`//p[string(.) = " "]` - строка содержит только пробел

`//p[string(.) = ""]` – пустая строка

`//*[matches(., '[\s\u00A0]')]` – любой пробел (хотя бы один)

`replace(., '\u00A0', ' ')` - замена неразрывного пробела на обычный

`//*[not(replace(., '[\s\u00A0]', ''))]` – выбирает элементы которые полностью состоят из пробелов (в т.ч. неразрывных)

`//h1[matches(., '?{4,}')]` – если в строке ≥ 4 символов "?" подряд

Все элементы у которых есть любые атрибуты, кроме id `//*[boolean(@*[name() != 'id'])]`

Стили

При генерации html-разметки через XPath можно применять базовые css-стили для форматирования текста. Поддерживаются следующие свойства:

Шрифты: шрифты (font-family), размеры (font-size), начертание (font-weight, font-style)

Текст: жирный (bold), курсив (italic)

Центрирование: (text-align: center)

Дополнительное оформление (подчёркивание, зачёркивание)

Пример:

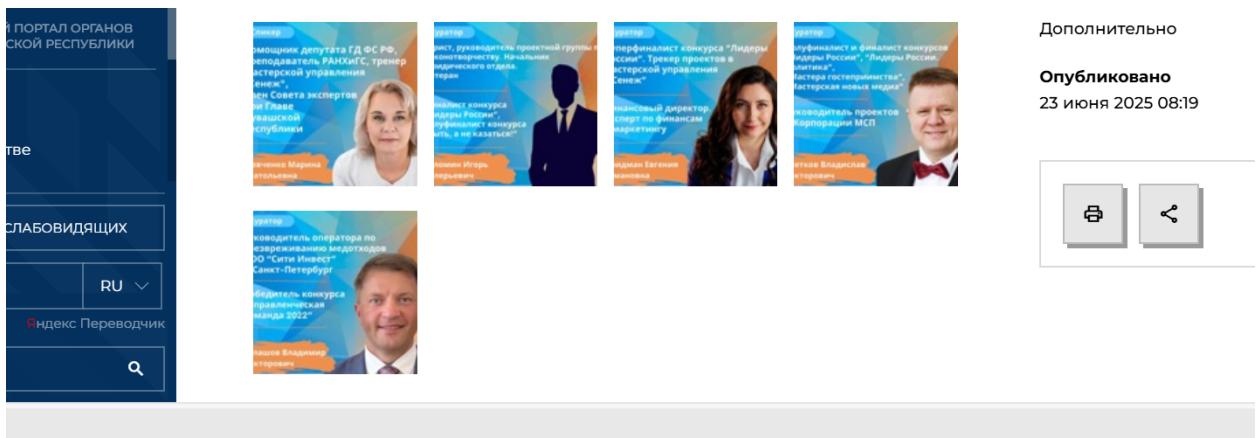
```
concat(  
'<p style="',  
'font-family: Calibri, sans-serif;',  
'font-weight: bold;',  
'font-style: italic;',  
'font-size: 2em;',  
'">'
```

```
(//div[@class="page-detail"]//b)[1],  
'</p>'  
)
```

Если картинка расположена в ссылке <a>

[Пресс-центр | Министерство образования Чувашской Республики](#)

```
if(not(normalize-space((//div[@class="news_text"])[1])))  
then(//*[contains(@class,"news_item_title")]]//h1, for $x in  
//div[@class="swip_wr1"]/a/@href return concat('<img src=""', $x, '">'))  
else((//div[@class="news_text"])[1])
```



```
<div class="swip_wr">  
  <div class="swip_wr1">  
    <a title href="https://fs.cap.ru/file/17a0eab74a28445cae009eedfa3d7bf7" alt data-bgimage="https://fs.cap.ru/file/17a0eab74a28445cae009eedfa3d7bf7" fancybox="gallery" target="_blank" style="background-image: url("https://fs.cap.ru/file/9a4d565ac13044538379cfffa1ed3519");" data-lg-id="c7c94aaa-4cc13263a6b9"></a>  
  </div>  
  <div class="swip_wr">  
    <div class="swip_wr1" == $0>  
      <a title href="https://fs.cap.ru/file/c47e09200ffcc44cd87e3e1fb536687b4" alt data-bgimage="https://fs.cap.ru/file/c47e09200ffcc44cd87e3e1fb536687b4" fancybox="gallery" target="_blank" style="background-image: url("https://fs.cap.ru/file/e689832f9b744540b3a758abcfcd4194");" data-lg-id="4f42f9d0-c92c94d9a058"></a>  
    </div>  
  </div>  
</div>
```

Также на стили влияют некоторые теги: <s>, <mark>, <code>, ...

В строке не менее 2 символов точки:

```
//h1/following-sibling::div[string-length(normalize-space()) = 16 and matches(., '.*\..*\.', 's')]
```

```
//p[string-length(.) - string-length(translate(., '#', "")) = 10]
```

Если url закодированы по типу ...%50%25... for \$x in //a[@class="newline__item"] return replace(\$x/@href, '_\d{1,3}%25_', '')

```
if ((//div[@class="article-details__date"])[1][not(ancestor::div[@style="display: none"])])[matches(., "вчера|сегодня", "i")] then (if ((//div[@class="article-details__date"])[1][not(ancestor::div[@style="display: none"])])[matches(., "вчера", "i")]) then (concat(concat(substring(string(current-date()), 1, 8), string(number(day-from-date(current-date()) - 1))), substring-before((//div[@class="article-details__date"])[1][not(ancestor::div[@style="display: none"])]), ","))) else (current-date()) else (substring-before((//div[@class="article-details__date"])[1][not(ancestor::div[@style="display: none"])]), ".")
```

Чтобы читались все атрибуты

Нужно превратить все атрибуты в строку. Например так:

```
//h1[matches(string-join(@*, ' '), 'title|heading', 'i')]
```

Здесь:

`@*` → берём все атрибуты,

`string-join(..., ' ')` → склеиваем их значения через пробел,

`matches(..., 'title|heading', 'i')` проверяет по всей склейке.

Работает по типу `distinct-values()` убирает дубли `((//a[./h3])[not(@href = preceding::a[./h3]/@href)]/@href`

Если в узле только неразрывный пробел: `//p[normalize-space(translate(., codepoints-to-string(160), " ")) = ""]`

