Parsing HTML with CSS Selectors

by Bernardas458Ališauskas Apr 11, 2024 #Data Parsing #Css Selectors

When it comes to parsing web scraped HTML content, there are multiple techniques to select the data we want. For simple text parsing - various text parsing techniques like regular expression can be used. However, HTML is designed to be a machine-readable text structure - we can take advantage of this fact and use special path languages like CSS selectors to extract data in a much more efficient and reliable way!

In this article, we'll be taking a deep look at this unique path language and how can we use it to extract needed details from modern, complex HTML documents.

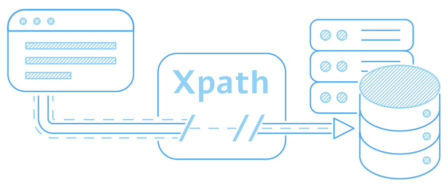
What Are CSS Selectors?

If ever done some web development, you are probably familiar with CSS selectors for applying styles to HTML websites - we can use the same tool for parsing HTML data! Cascading Style Sheets protocol offers a unique path language for selecting HTML nodes to apply style to - these are called CSS Selectors. While this path language is designed to find nodes for styling, we can also use it to find nodes for parsing and navigation in our web scrapers.

CSS selectors tend to be brief but powerful enough for most web-scraping related parsing. Let's take a quick look at common pros and cons, especially *compared to Xpath selectors*:

Parsing HTML with Xpath

For more on xpath selectors see our in depth introduction article which covers xpath syntax, usage and various tips and tricks.



Pros:

- Brief and simple to read - which in turn means easier to maintain.
- Made for the web. Since CSS selectors are used to apply styles for content, that means our web scraper should be able to select elements just as our browser can. This, also means this path language has a huge community
- Part of web standard - meaning it's built-in web browsers and many other web tools which makes CSS selectors very accessible!

Cons:

- Only capable of selecting nodes, where's xpath can select node attributes too. However, many CSS selector clients expand functionality with their own extra syntax. For example, for selecting node attributes or inner text extra pseudo classes like `::attr`, `::text` or simple client methods like `.text()` or `.attribute()` are being added as extra functionality.
- Not very extendible. Unlike XPATH, CSS selector clients tend to offer less opportunity for extensions.
- Not as powerful as XPATH. We'll explore this more later but generally, CSS selectors might not be able to traverse HTML trees as freely as XPATH selectors.

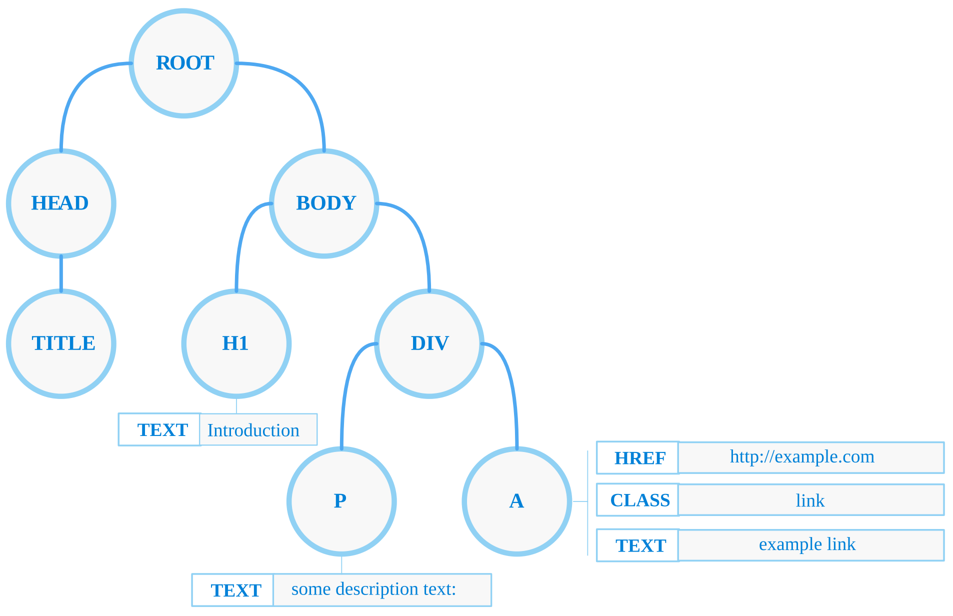
HTML Overview

HTML (HyperText Markup Language) is designed to be easily machine-readable and parsable. In other words, HTML follows a tree-like structure of nodes and their attributes, which we can easily navigate programmatically.

Let's start off with a small example page and illustrate its structure:

```
<head>
  <title>
  Document Title
  </title>
</head>
<body>
  <h1>Introduction</h1>
  <div>
    <p>some description text: </p>
    <a class="link" href="https://example.com">example link</a>
  </div>
</body>
```

In this basic example of a simple web page, we can see that the document already resembles a data tree. Let's go a bit further and illustrate this:



HTML tree is made of nodes which can contain attributes such as classes, IDs and text itself.

Here we can wrap our heads around it a bit more easily: it's a tree of nodes, and each node can also have properties attached to them like keyword attributes (like `class` and `href`) and natural attributes such as text.

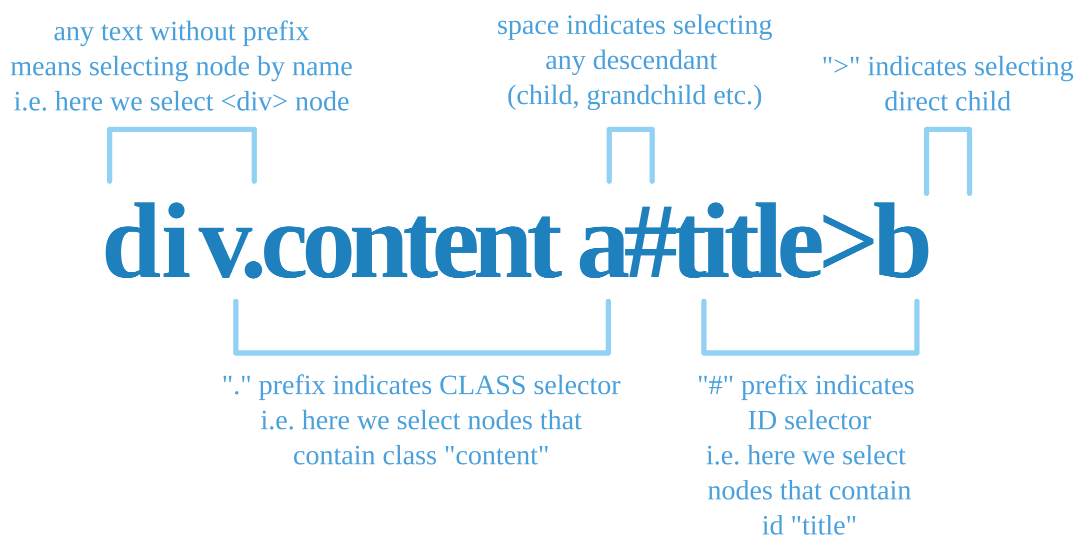
Now that we're familiar with HTML let's familiarize ourselves with CSS selector syntax and let's use it to parse some data!

HTML Syntax Overview

Css selectors are often referred to as *selectors* and a single selector indicates a path to a particular HTML node.

To test our CSS selectors, we'll be using an embedded selector playground

The average CSS selector in web scraping often looks something like this:


any text without prefix means selecting node by name i.e. here we select <div> node
space indicates selecting any descendant (child, grandchild etc.)
">" indicates selecting direct child
div.content a#title>b
"." prefix indicates CLASS selector i.e. here we select nodes that contain class "content"
"#" prefix indicates ID selector i.e. here we select nodes that contain id "title"

The most common CSS selector features are class and descendant selectors

In this example, our selector will select all `<b>` nodes that are children of an `<a>` node with an ID of "title" which is under a `<div>` node with a CLASS of "content":

```
<div class="content">
  <a id="title" href="https://twitter.com/@scrapfly_dev">follow us on twit
    <b>@scrapfly_dev</b></a>
</div>
```

div.content a#title>b xpath css

```
<b>@scrapfly_dev</b>
```

As you can see, CSS selectors is just a chain of various expressions joined either by a space or ">" character. Let's see the most commonly used expressions in web-scraping:

expression	description
`node`	selects any descendant (child, grandchild etc.) that matches node name
`>node`	selects a direct child that matches node name
`~node`	selects sibling that matches node name
`+node`	selects only adjacent siblings that matches node name
`.class`	class constraint - select only nodes that contain this class
`#id`	ID constraint - select only nodes that contain this ID
`[attribute=]`	attribute match constraint, e.g. `span[data=foo]` will select all span node with data="foo" attribute
`[attribute*=]`	attribute contains constraint, e.g. `span[data*=foo]` will select all span node with "foo" value in the `data` attribute, like: `<span data="foobar gaz">`
`[... i]`	attribute constrain match case insensitivity indicator, e.g. `span[data=foo i]` will match both `<span data="Foo">` and `<span data="FOO">` etc.
`,`	allows grouping of multiple selectors joining all results, e.g. `h1, h2` will select both h1 and h2 nodes

This is the core syntax available in most CSS selector clients, which should provide us with enough flexibility to parse most of HTML trees we might encounter on the modern web. Let's take a look at some examples!

Interactive CSS Selector Cheatsheet

To remember all of this syntax see our interactive cheatsheet which explains all of the CSS syntax used in web scraping.

Basic CSS Navigation

The most important feature of CSS selectors is node selection by name and descendant chaining. For example, using `>` character we can chain multiple node selectors:

```
<div>
```

```
  <p class="socials">
    Follow us on
    <a href="https://twitter.com/@scrapfly_dev">Twitter!</a>
  </p>
</div>
```

div>p>a xpath css

```
<a href="https://twitter.com/@scrapfly_dev">Twitter!</a>
```

This allows us to define strict selection paths. However, using direct child selector (`>`) can make our selectors too strict for highly dynamic HTML files found in modern websites. In other words, what if the `<a>` node gets wrapped in some other styling node? That would break our selector.

Instead, we should use a mixture of space and `>` selectors to find the sweet spot of stability and accuracy:

```
<div>
  <p class="primary socials content">
    Follow us on
  <ul>
    <li>
      <a href="https://twitter.com/@scrapfly_dev">Twitter!</a>
    </li>
  </ul>
  <a href="#">advertisement</a>
  </p>
</div>
```

p.socials li>a xpath css

```
<a href="https://twitter.com/@scrapfly_dev">Twitter!</a>
```

Here instead of defining a direct path we root our selector to `<div>` node that contains class `socials` (using `.socials` class constraint) and from there we can assume that any link in an unordered list is a social link.
Relying less on HTML structure and more on the context allows creating selectors that break less often on HTML structure changes.

Ideally, when designing our selectors we want to find the sweet spot between structure and context which will result in no false positives and something that doesn't break on small HTML tree changes:

```
<html>
<p class="primary socials content">
  Follow us on
<ul>
  <li>
    <a href="https://twitter.com/@scrapfly_dev">Twitter!</a>
  </li>
  <li>
    <a href="https://linkedin.com/@scrapfly_dev">Linkedin!</a>
  </li>
</ul>
```

```
<a href="#">advertisement</a>
</p>

</html>
```

p.socials li>a[href*="linkedin"]

xpath css

```
<a href="https://linkedin.com/@scrapfly_dev">Linkedin!</a>
```

Here, we're using attribute contains constraint to restrict extraction only of links that contain `"linkedin"` in their urls.

In this section, we've discovered what makes a good CSS selector in web scrapers: we want something robust that doesn't select false positives and something not too strict that might miss some results.
Further, let's take a look at some more complex parsing scenarios and how can we solved them using CSS selectors alone

Navigating Complex Structures

Unfortunately, modern websites can have very complex and dynamic HTML trees that difficult to navigate reliably. Let's take a look at few common examples of complex structures and how can we solve them with CSS selectors.

Firstly, it's nice to remember that we don't have to cram everything into a single CSS selector, and we can safely join multiple selectors using the `,` syntax:

```
<div>
  // American English website might contain:
  <p>Our favorite product is
    <b>product 1</b>
  </p>
  // while British website might contain:
  <p>Our favourite product is
    <b>product 2</b>
  </p>
</div>
```

p:contains('favorite')>b, p:contains('favourite')>b

xpath css

```
<b>product 1</b>
<b>product 2</b>
```

In this example, we use two selectors for 2 different spellings of the word "favorite". We're also using a special pseudo class `:contains` which allows us to check whether text value of the node contains some string.

Note that pseudo-class and pseudo-element availability varies by client. For example, `:contains` pseudo class is available in most web-scraping focused clients like parsel and javascript native ones jquery or sizzle

Another cool feature of selector joining is that all results come in ordered by their appearance, which means we can safely join selectors and retain the content structure:

```
<div>
  <p>For this recipe you'll need:</p>
  <a href="https://patreon.com">Support us on patreon</a>
  <b>600g of butter</b>
  <i>*margarine also works</i>
  <p>First, preheat the oven to 200C...</p>
  <p class="promo">For more recipes click the subscribe button</p>
</div>
```

```
b, i, p:not([class*=promo])
```

xpath css

```
<p>For this recipe you'll need:</p>
<b>600g of butter</b>
<i>*margarine also works</i>
<p>First, preheat the oven to 200C...</p>
```

In this example, we extract recipe text while avoiding promos and other non-recipe related texts. We're also using `:not` pseudo class which allows us to reverse our selector constraints, which is very useful for filtering out unwanted nodes.

Finally, last one of important CSS selector features for web scraping is result slicing. Often we want to select only matching nodes of specific indices:

```
<div>
  <p>advertisement paragraph</p>
  <p>first paragraph</p>
  <p>second paragraph</p>
  <p>advertisement paragraph</p>
</div>
```

```
p:nth-of-type(n + 2):nth-last-of-type(n + 2)
```

xpath css

```
<p>first paragraph</p>
<p>second paragraph</p>
```

In this example, we're using `:nth-of-type` and `:nth-last-of-type` to implement basic result slicing which allows us to filter out first and last nodes from our selection!

While Css selectors might appear a bit clunky compared to Xpath selectors (or other options) there's a surprising amount of power there that with some clever engineering can help us reliably extract data from HTML documents.

CSS Selector Clients

CSS selectors are primarily used in front-end web development, however there are few backend implementations that are used as clients for HTML parsing. Let's take a look at the most popular libraries that implement CSS selectors.

Python

To parse HTML in Python we have several options. However, many of them instead of executing natural CSS selectors convert them to XPath by using cssselect and run the XPath selectors through lxml XPath client. An example of such client would be parsel.

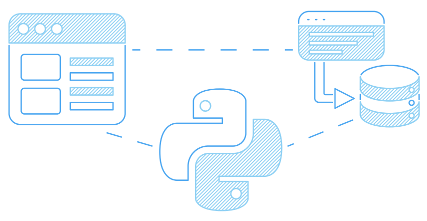
Other packages implement CSS selectors in varying capacity:

- selectolax - is a new modern, blazing fast CSS selector client.
- beautifulsoup - classic python client that supports CSS selectors as well as xpath and python object based navigation.

For Python, to parse HTML the most popular choices are beautifulsoup4 or parsel.

Hands on Python Web Scraping Tutorial and Example Project

For more on web scraping in Python check out our full introduction tutorial.

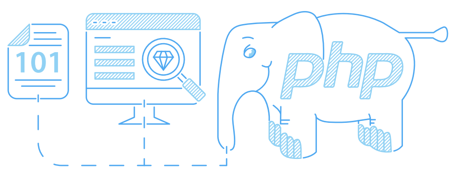


PHP

PHP like python also prefers Xpath selectors thus most CSS selector clients use css-selector component to convert CSS selectors to xpath selectors and execute them through either built-in DOMXPath or community favorite DOMCrawler

Web Scraping With PHP 101

For more on html parsing in php see our introduction article on web scraping with php which covers usage of `DOMCrawler` with both CSS and xpath selectors



Alternatively, when using browser emulation through browser emulation clients like Selenium php also gets access to browser's CSS selector capabilities:

```
// we can use findElements method of Selenium web driver to find elements by CSS
selectors
$webDriver->findElements(WebDriverBy::cssSelector("div.content a#title>b"));
```

Ruby

Ruby has several CSS capable clients, however most popular package is nokogiri which offers both CSS and xpath selectors and loads of parsing utility functions and extensions:

```
html_doc = Nokogiri::HTML('<html><body><div class="socials"><a
href="https://scrapfly.io/blog">Our blog</a></div></body></html>')
@doc.css("div.socials>a").attributes["href"]
```

Other Languages

Often xpath selectors are being favored over CSS selectors which makes CSS less accessible outside the mentioned few. That being said, since CSS selectors are very similar to XPATH selectors there's typically at least a community maintained translation layer available!

FAQ

Are CSS selectors better than XPATH?

Both path languages have their pros and cons. Generally, CSS selectors are briefer but less powerful than xpath. When web scraping it's best to mix both!

How to select element parent using CSS selectors?

CSS selectors do not support selection of node parents. Instead, XPath `..` selector can be used, e.g. `root/child/..` will select `root`

Can CSS selectors be extended like XPATH?

No, typically CSS selectors do not support native extensions. However many web scraping libraries can be patched with extra features as CSS selectors are typically converted to XPath selectors before runtime. That being said, it's better to fallback to XPath if CSS capabilities are inadequate.

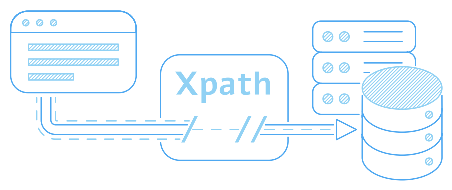
CSS Selector Summary

In this introduction article we covered the syntax of CSS selectors, explored basic navigation to solidify our knowledge and finally finished off by taking a look at more advanced usages to fully grasp what this little path language is capable off!

While CSS selectors are great ideally when web scraping it's best to take advantage of both CSS and XPATH selectors. Common idiom is to use CSS selectors for simple paths as they are short and easy to follow and for more complex selections fall back to xpath which is more verbose and powerful.

Parsing HTML with Xpath

For more on xpath selectors see our in depth introduction article which covers xpath syntax, usage and various tips and tricks.



For further CSS selector help we advise checking out #CSS-selectors tag on Stackoverflow which is very active and full of dedicated teachers!

Discover ScrapFly

Try ScrapFly for FREE!

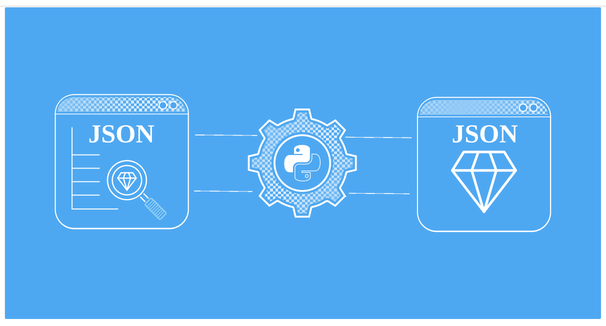
Related Questions

- How to scrape HTML table to Excel Spreadsheet (.xlsx)?
- How to select dictionary key recursively in Python?
- How to select all elements between two elements in XPath?
- How to parse dynamic CSS classes when web scraping?
- How to turn HTML to text in Python?
- How to use CSS selectors in NodeJS when web scraping?
- How to use XPath selectors in Python?
- How to select elements by text in XPath?
- How to select last element in XPath?
- What are some ways to parse JSON datasets in Python?
- What are devtools and how they're used in web scraping?
- How to select HTML elements by text using CSS Selectors?
- Scraper doesn't see the data I see in the browser - why?
- How to use XPath selectors in NodeJS when web scraping?
- How to select elements by class in XPath?

More >

DATA PARSING CSS SELECTORS

Related Posts

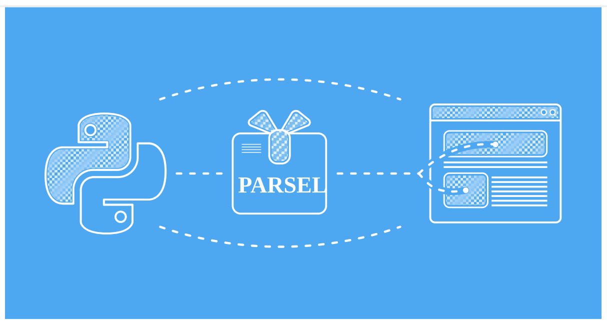


Jan 03, 2025

Ultimate Guide to JSON Parsing in Python

Learn JSON parsing in Python with this ultimate guide. Explore basic and advanced techniques using json, and tools like ijson and nested-lookup

DATA PARSING PYTHON

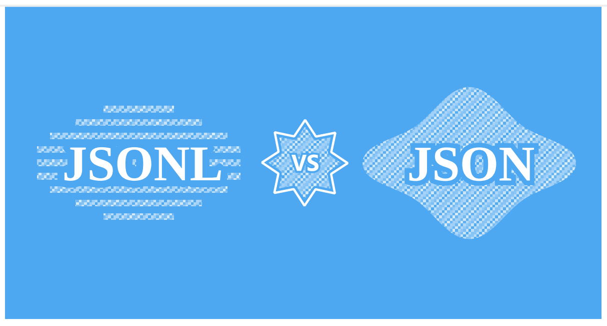


Dec 25, 2024

Guide to Parsel - the Best HTML Parsing in Python

Learn to extract data from websites with Parsel, a Python library for HTML parsing using CSS selectors and XPath.

DATA PARSING PARSEL



Dec 17, 2024

JSONL vs JSON

Learn the differences between JSON and JSONLines, their use cases, and efficiency. Why JSONLines excels in web scraping and real-time processing

DATA PARSING

Company

Careers
Terms of service
Privacy Policy
Data Processing Agreement
KYC Compliance
Status

Integrations

Zapier

Make
N8n
LlamaIndex
LangChain

Social

Tools

Convert cURL commands to Python code
JA3/TLS Fingerprint
HTTP2 Fingerprint
Xpath/CSS Selector Tester

Resources

API Documentation
Web Scraping Academy
Is Web Scraping Legal?
Web Scraping Tools
FAQ

Learn Web Scraping

Web Scraping with Python
Web Scraping with PHP
Web Scraping with Ruby
Web Scraping with R
Web Scraping with NodeJS
Web Scraping with Python Scrapy
How to Scrape without getting blocked tutorial
Web Scraping with Python and BeautifulSoup
Web Scraping with Nodejs and Puppeteer
How To Scrape Graphql
Best Proxies for Web Scraping
Top 5 Best Residential Proxies

Usage

What is Web Scraping used for?
Web Scraping for AI Training
Web Scraping for Compliance
Web Scraping for eCommerce
Web Scraping for Finance
Web Scraping for Fraud Detection
Web Scraping for Jobs
Web Scraping for Lead Generation
Web Scraping for News & Media
Web Scraping for Real Estate
Web Scraping for SERP & SEO
Web Scraping for Social Media
Web Scraping for Travel