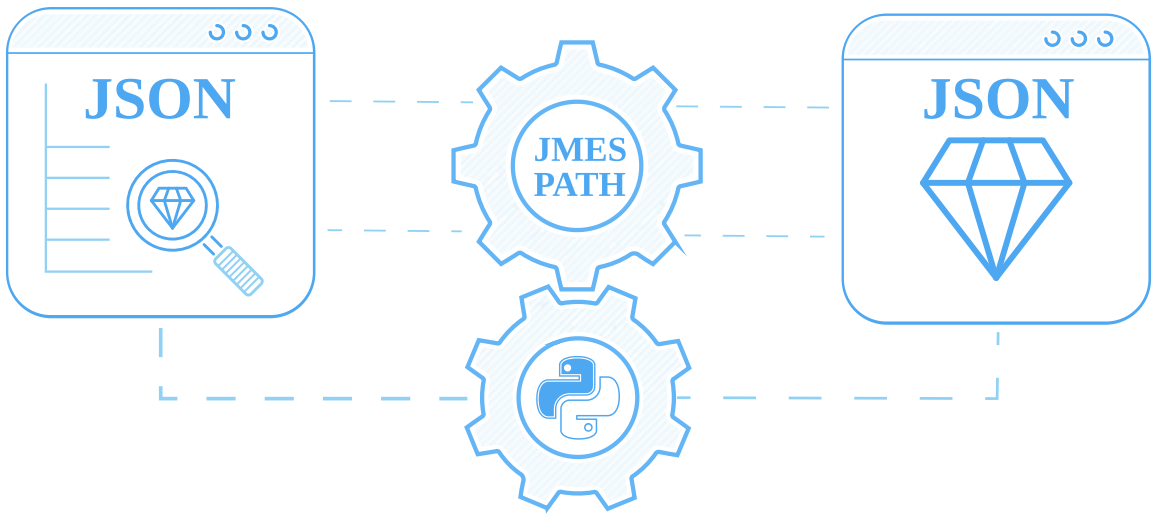
Quick Intro to Parsing JSON with JMESPath in Python

by Bernardas Ališauskas May 10, 2024 #Data Parsing #Python



JMESPath is a popular JSON query language used for parsing JSON datasets. It has gained popularity in web scraping as JSON is becoming the most popular data structure in this medium.

Many popular web scraping targets contain hidden JSON data that can be extracted directly. Unfortunately, these datasets are huge and contain loads of useless data. This makes JSON parsing an important part of the modern web scraping process.

In this Jmespath tutorial, we'll take a quick overview of this path language in web scraping and Python. We'll cover setup, the most used features and do a quick real-life example project by scraping Realtor.com.

JOIN THE NEWSLETTER

Get monthly web scraping insights 👆

Learn at ScrapFly Academy

What is JMESPath?

JMESPath is a path language for parsing JSON datasets. In short, it allows writing path rules for selecting specific data fields in JSON.

When web scraping, JMESPath is similar to XPath or CSS selectors we use to parse HTML - but for JSON. This makes JMESPath a brilliant addition to our web scraping toolset as HTML and JSON are the most common data formats in this niche.

JMESPath Setup

JMESPath is implemented in many different languages:

Language	Implementation
Python	jmespath.py
PHP	jmespath.php
Javascript	jmespath.js
Ruby	jmespath.rb
Lua	jmespath.lua
Go	go-jmespath
java	jmespath-java
Rust	jmespath.rs
DotNet	jmespath.net

In this tutorial, we'll be using Python though other languages should be very similar.

To install jmespath in Python we can use `pip install` terminal command:

```
$ pip install jmespath
```

Jmespath Usage Tutorial

You're probably familiar with dictionary/hashtable dot-based path selectors like `data.address.zipcode` - this dot notation is the foundation of JMESPath but it can do much more!

Just like with Python's lists we can also **slice and index** jmespath arrays:

```
import jmespath
data = {
    "people": [
        {"address": ["123 Main St", "California" ,"US"]},
    ]
}
jmespath.search("people[0].address[:2]", data)
['123 Main St', 'California']
```

Further, we can apply projections that apply rules for **each list element**. This is being done through the `[]` syntax:

```
data = {
  "people": [
    {"address": ["123 Main St", "California" ,"US"]},
    {"address": ["345 Alt St", "Alaska" ,"US"]},
  ]
}
jmespath.search("people[].address[:2]", data)
[
  ['123 Main St', 'California'],
  ['345 Alt St', 'Alaska']
]
```

Just like with lists we can also apply similar projections to objects (dictionaries). For this `*` is used:

```
data = {
  "people": {
    "foo": {"email": "foo@email.com"},
    "bar": {"email": "bar@email.com"},
  }
}
jmespath.search("people.*.email", data)
[
  "foo@email.com",
  "bar@email.com",
]
```

The most interesting feature of JMESPath for web scraping has to be **data reshaping**. Using the `.[]` and `.{}` syntax we can completely reshape lists and objects:

```
data = {
  "people": [
    {
      "name": "foo",
      "age": 33,
      "addresses": [
        "123 Main St", "California", "US"
      ],
      "primary_email": "foo@email.com",
      "secondary_email": "bar@email.com",
    }
  ]
}
jmespath.search("""
  people[].{
    first_name: name,
    age_in_years: age,
    address: addresses[0],
    state: addresses[1],
    country: addresses[2],
    emails: [primary_email, secondary_email]
  }
""", data)
[
  {
    'address': '123 Main St',
    'age_in_years': 33,
    'country': 'US',
    'emails': ['foo@email.com', 'bar@email.com'],
    'first_name': 'foo',
    'state': 'California'
  }
]
```

As you can see, using JMESPath we can easily parse complex datasets into something more digestible which is especially useful when web scraping JSON datasets.

Web Scraper Example

Let's explore a real-life JMESPath python example by taking a look at how it would be used in web scraping.

In this example project, we'll be scraping real estate property data on realtor.com which is a popular US portal for renting and selling properties.

We'll be using a few Python packages:

- httpx - HTTP client library which will let us communicate with Realtor.com's servers
- parsel - HTML parsing library which will help us to parse our web scraped HTML files.

And of course `jmespath` for parsing JSON. All of these can be installed using `pip install` command:

```
$ pip install jmespath httpx parsel
```

Realtor.com is using hidden web data to render its property pages which means instead of parsing HTML we can find the whole JSON dataset hidden in the HTML code.

Let's take a look at any random example property like this one

If we take a look at the page source we can see the JSON data set hidden in a `<script>` tag:

We can see entire property dataset hidden in a script element

We can extract it using HTML parser though it's huge and contains a bunch of gibberish computer data. So we can parse out the useful bits using JMESPath:

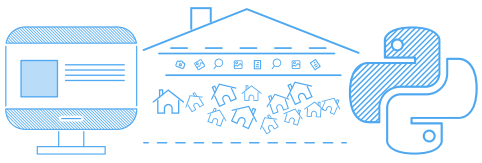
```python
import json
import httpx
import jmespath
from parsel import Selector

# establish HTTP client and to prevent being instantly banned lets set some browser-like headers
session = httpx.Client(
    headers={
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36",
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
        "Accept-Language": "en-US,en;q=0.9",
        "Accept-Encoding": "gzip, deflate, br",
    },
)

# 1. Scrape the page and parse hidden web data
response = session.get(
    "https://www.realtor.com/realestateandhomes-detail/335-30th-Ave_San-Francisco_CA_94121_M17833-49194"
)
assert response.status_code == 200, "response is banned - try ScrapFly? 😉"
selector = Selector(text=response.text)
# find <script id="__NEXT_DATA__"> node and select it's text:
data = selector.css("script#__NEXT_DATA__::text").get()
# load JSON as python dictionary and select property value:
data = json.loads(data)["props"]["pageProps"]["initialProps"]["propertyData"]

# 2. Reduce web dataset to important data fields:
result = jmespath.search(
    """{
    id: listing_id,
    url: href,
    status: status,
    price: list_price,
    price_per_sqft: price_per_sqft,
    date: list_date,
    details: description,
    features: details[].text[],
    photos: photos[].{url: href, tags: tags[].label}
}
""", data)
print(result)
```

▶ **Example Output**

Using JMESpath we managed to reduce thousands of lines of JSON to essential data fields in just a few lines of Python code and a single JMESPath query - pretty awesome!

How to Scrape Realtor.com - Real Estate Property Data

For a full tutorial on scraping Realtor.com see our complete scrape guide which covers, scraping, parsing and how to avoid blocking



FAQ

To wrap this up JMESPath tutorial let's take a look at some frequently asked questions:

What's the difference between [] and [*] in JMESPath?

The `[]` flattens all results while `[*]` keeps the structure as it is in the orignal dataset. See this example in Python:

```
data = {
  "employees": [
    {
      "people": [
        {"address": ["123 Main St", "California", "US"]},
        {"address": ["456 Sec St", "Nevada", "US"]},
      ],
    },
    {
      "people": [
        {"address": ["789 Main St", "Washington", "US"]},
        {"address": ["a12 Sec St", "Alaska", "US"]},
      ],
    },
  ]
}

jmespath.search("employees[*].people[*].address", data)
[
  # fromt he first group:
  [['123 Main St', 'California', 'US'], ['456 Sec St', 'Nevada', 'US']],
  # from the second group:
  [['789 Main St', 'Washington', 'US'], ['a12 Sec St', 'Alaska', 'US']]
]

jmespath.search("employees[].people[].address", data)
[
  # all groups merged:
  ['123 Main St', 'California', 'US'],
  ['456 Sec St', 'Nevada', 'US'],
  ['789 Main St', 'Washington', 'US'],
  ['a12 Sec St', 'Alaska', 'US']
]
```

Can JMESPath be used on HTML?

No, for that refer to very similar HTML path languages like CSS Selectors and Xpath Selectors.

What are some alternatives to JMESPath?

Some other popular query language for JSON are JsonPath, JQ and pyquery

Jmespath in Web Scraping Summary

In this tutorial on JMESPath we did a quick overview of what this path language is capable of when it comes to parsing JSON.

We've covered JMESPath's multiple filters and selectors through Python examples using jmespath python library.

Quick Intro to Parsing JSON with JSONPath in Python

For an alternative to JMESPath see our intro to JSONPath which is another popular format for parsing JSON datasets in Python

Finally, to wrap everything up we've taken a look at how JMESPath is used in web scraping through a real-life scraper.

Check out ScrapFly Python SDK

Try ScrapFly for FREE!

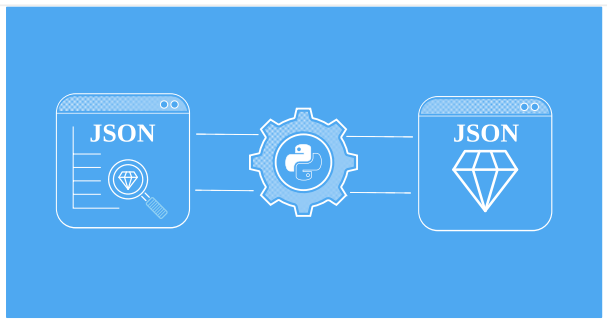
Related Questions

- 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 PYTHON

Related Posts

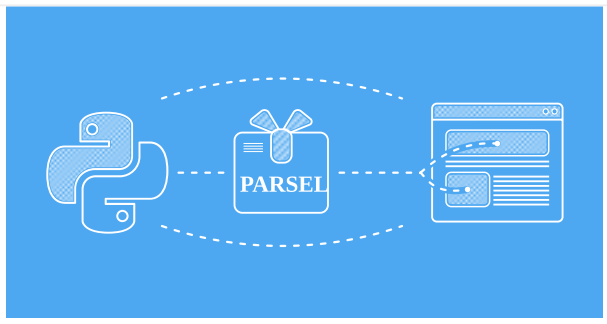


Jan 03, 2025

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

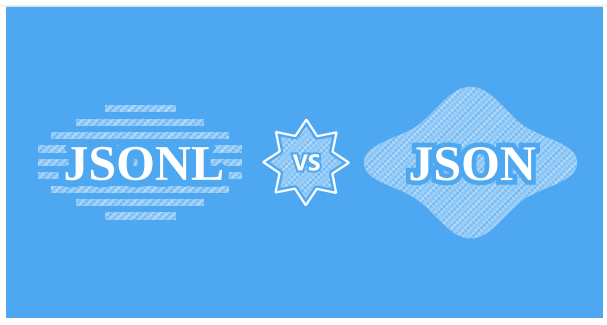


Dec 25, 2024

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

© 2025 Scrapfly - The Best Web Scraping API For Developers