

- [Home](#)
- [About Me](#)
- [About This Blog](#)



Building Web APIs with Flask

Flask is the ideal Python framework for building REST APIs that are flexible, easy to maintain and efficient. I have created a training video with O'Reilly in which I show all my techniques:



Visit <http://flaskbook.com> for information about this and other Flask training products I have created!

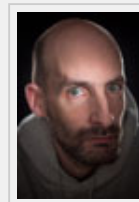
About Miguel

Welcome to my blog!

I'm a software engineer, photographer and filmmaker in Portland, Oregon, USA.

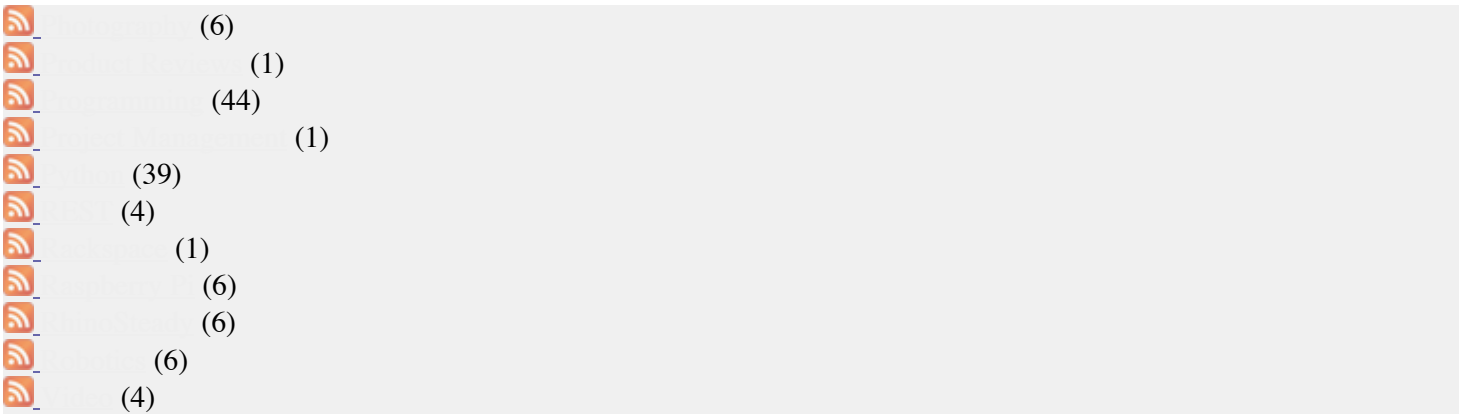
You can also find me on [Facebook](#), [Google+](#), [LinkedIn](#), [Github](#) and [Twitter](#).

Thank you for visiting!



Categories

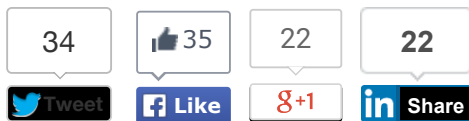




Easy Web Scraping with Python

April 21 2014

Posted by [Miguel Grinberg](#) under [Programming](#), [Python](#).



A little over a year ago I wrote an article on [web scraping using Node.js](#). Today I'm revisiting the topic, but this time I'm going to use Python, so that the techniques offered by these two languages can be compared and contrasted.

The Problem

As I'm sure you know, I attended PyCon in Montréal earlier this month. The video recordings of all the talks and tutorials have already been released on YouTube, with an index available at [pyvideo.org](#).

I thought it would be useful to know what are the most watched videos of the conference, so we are going to write a scraping script that will obtain the list of available videos from pyvideo.org and then get viewer statistics from each of the videos directly from their YouTube page. Sounds interesting? Let's get started!

The Tools

There are two basic tasks that are used to scrape web sites:

1. Load a web page to a string.
2. Parse HTML from a web page to locate the interesting bits.

Python offers two excellent tools for the above tasks. I will use the awesome [requests](#) to load web

pages, and [BeautifulSoup](#) to do the parsing.

We can put these two packages in a virtual environment:

```
$ mkdir pycon-scraper
$ virtualenv venv
$ source venv/bin/activate
(venv) $ pip install requests beautifulsoup4
```

If you are using Microsoft Windows, note that the virtual environment activation command above is different, you should use `venv\Scripts\activate`.

Basic Scraping Technique

The first thing to do when writing a scraping script is to manually inspect the page(s) to scrape to determine how the data can be located.

To begin with, we are going to look at the list of PyCon videos at <http://pyvideo.org/category/50/pycon-us-2014>. Inspecting the HTML source of this page we find that the structure of the video list is more or less as follows:

```
<div id="video-summary-content">
  <div class="video-summary">    <!-- first video -->
    <div class="thumbnail-data">...</div>
    <div class="video-summary-data">
      <div>
        <strong><a href="#link to video page#">#title#</a></strong>
      </div>
    </div>
  <div class="video-summary">    <!-- second video -->
    ...
  </div>
  ...
</div>
```

So the first task is to load this page, and extract the links to the individual pages, since the links to the YouTube videos are in these pages.

Loading a web page using requests is extremely simple:

```
import requests
response = requests.get('http://pyvideo.org/category/50/pycon-us-2014')
```

That's it! After this function returns the HTML of the page is available in `response.text`.

The next task is to extract the links to the individual video pages. With BeautifulSoup this can be done using [CSS selector](#) syntax, which you may be familiar if you work on the client-side.

To obtain the links we will use a selector that captures the `<a>` elements inside each `<div>` with class `video-summary-data`. Since there are several `<a>` elements for each video we will filter them to include only those that point to a URL that begins with `/video`, which is unique to the individual video pages. The CSS selector that implements the above criteria is `div.video-summary-data a[href^=/video]`. The following snippet of code uses this selector with BeautifulSoup to obtain the `<a>` elements that point to video pages:

```
import bs4
soup = bs4.BeautifulSoup(response.text)
links = soup.select('div.video-summary-data a[href^=/video]')
```

Since we are really interested in the link itself and not in the `<a>` element that contains it, we can improve the above with a list comprehension:

```
links = [a.attrs.get('href') for a in soup.select('div.video-summary-data a[href^=/video]')]
```

And now we have a list of all the links to the individual pages for each session!

The following script shows a cleaned up version of all the techniques we have learned so far:

```
import requests
import bs4

root_url = 'http://pyvideo.org'
index_url = root_url + '/category/50/pycon-us-2014'

def get_video_page_urls():
    response = requests.get(index_url)
    soup = bs4.BeautifulSoup(response.text)
    return [a.attrs.get('href') for a in soup.select('div.video-summary-data a[href^=/video]')]

print(get_video_page_urls())
```

If you run the above script you will get a long list of URLs as a result. Now we need to parse each of these to get more information about each PyCon session.

Scraping Linked Pages

The next step is to load each of the pages in our URL list. If you want to see how these pages look, here is an example: <http://pyvideo.org/video/2668/writing-restful-web-services-with-flask>. Yes,

that's me, that is one of my sessions!

From these pages we can scrape the session title, which appears at the top. We can also obtain the names of the speakers and the YouTube link from the sidebar that appears on the right side below the embedded video. The code that gets these elements is shown below:

```
def get_video_data(video_page_url):
    video_data = {}
    response = requests.get(root_url + video_page_url)
    soup = bs4.BeautifulSoup(response.text)
    video_data['title'] = soup.select('div#videobox h3')[0].get_text()
    video_data['speakers'] = [a.get_text() for a in soup.select('div#sidebar a[href^=/speaker]')]
    video_data['youtube_url'] = soup.select('div#sidebar a[href^=http://www.youtube.com]')[0].get_text()
```

A few things to note about this function:

- The URLs returned from the scraping of the index page are relative, so the `root_url` needs to be prepended.
- The session title is obtained from the `<h3>` element inside the `<div>` with id `videobox`. Note that `[0]` is needed because the `select()` call returns a list, even if there is only one match.
- The speaker names and YouTube links are obtained in a similar way to the links in the index page.

Now all that remains is to scrape the views count from the YouTube page for each video. This is actually very simple to write as a continuation of the above function. In fact, it is so simple that while we are at it, we can also scrape the likes and dislikes counts:

```
def get_video_data(video_page_url):
    # ...
    response = requests.get(video_data['youtube_url'])
    soup = bs4.BeautifulSoup(response.text)
    video_data['views'] = int(re.sub('[^0-9]', '',
                                   soup.select('.watch-view-count')[0].get_text().split()[0]))
    video_data['likes'] = int(re.sub('[^0-9]', '',
                                   soup.select('.likes-count')[0].get_text().split()[0]))
    video_data['dislikes'] = int(re.sub('[^0-9]', '',
                                   soup.select('.dislikes-count')[0].get_text().split()[0]))
    return video_data
```

The `soup.select()` calls above capture the stats for the video using selectors for the specific id names used in the YouTube page. But the text of the elements need to be processed a bit before it can be converted to a number. Consider an example views count, which YouTube would show as `"1,344 views"`. To remove the text after the number the contents are split at whitespace and only the first part is used. This first part is then filtered with a regular expression that removes any characters that are not digits, since the numbers can have commas in them. The resulting string is finally converted to an integer and stored.

To complete the scraping the following function invokes all the previously shown code:

```
def show_video_stats():
    video_page_urls = get_video_page_urls()
    for video_page_url in video_page_urls:
        print get_video_data(video_page_url)
```

Parallel Processing

The script up to this point works great, but with over a hundred videos it can take a while to run. In reality we aren't doing so much work, what takes most of the time is to download all those pages, and during that time the script is blocked. It would be much more efficient if the script could run several of these download operations simultaneously, right?

Back when I wrote the scraping article using Node.js the parallelism came for free with the asynchronous nature of JavaScript. With Python this can be done as well, but it needs to be specified explicitly. For this example I'm going to start a pool of eight worker processes that can work concurrently. This is surprisingly simple:

```
from multiprocessing import Pool

def show_video_stats(options):
    pool = Pool(8)
    video_page_urls = get_video_page_urls()
    results = pool.map(get_video_data, video_page_urls)
```

The `multiprocessing.Pool` class starts eight worker processes that wait to be given jobs to run. Why eight? It's twice the number of cores I have on my computer. While experimenting with different sizes for the pool I've found this to be the sweet spot. Less than eight make the script run slower, more than eight do not make it go faster.

The `pool.map()` call is similar to the regular `map()` call in that it invokes the function given as the first argument once for each of the elements in the iterable given as the second argument. The big difference is that it sends all these to run by the processes owned by the pool, so in this example eight tasks will run concurrently.

The time savings are considerable. On my computer the first version of the script completes in 75 seconds, while the pool version does the same work in 16 seconds!

The Complete Scraping Script

The final version of my scraping script does a few more things after the data has been obtained.

I've added a `--sort` command line option to specify a sorting criteria, which can be by views, likes or dislikes. The script will sort the list of results in descending order by the specified field. Another option, `--max` takes a number of results to show, in case you just want to see a few entries from the top. Finally, I have added a `--csv` option which prints the data in CSV format instead of table aligned, to make it easy to export the data to a spreadsheet.

The complete script is available for download at this location:

<https://gist.github.com/miguelgrinberg/5f52ceb565264b1e969a>.

Below is an example output with the 25 most viewed sessions at the time I'm writing this:

```
(venv) $ python pycon-scraper.py --sort views --max 25 --workers 8
Views  +1  -1 Title (Speakers)
3002  27   0 Keynote - Guido Van Rossum (Guido Van Rossum)
2564  21   0 Computer science fundamentals for self-taught programmers (Justin Abrahms)
2369  17   0 Ansible - Python-Powered Radically Simple IT Automation (Michael Dehaan)
2165  27   6 Analyzing Rap Lyrics with Python (Julie Lavoie)
2158  24   3 Exploring Machine Learning with Scikit-learn (Jake Vanderplas, Olivier Grisel)
2065  13   0 Fast Python, Slow Python (Alex Gaynor)
2024  24   0 Getting Started with Django, a crash course (Kenneth Love)
1986  47   0 It's Dangerous to Go Alone: Battling the Invisible Monsters in Tech (Julie Pagano)
1843  24   0 Discovering Python (David Beazley)
1672  22   0 All Your Ducks In A Row: Data Structures in the Standard Library and Beyond
1558  17   1 Keynote - Fernando Pérez (Fernando Pérez)
1449   6   0 Descriptors and Metaclasses - Understanding and Using Python's More Advanced Features
1402  12   0 Flask by Example (Miguel Grinberg)
1342   6   0 Python Epiphanies (Stuart Williams)
1219   5   0 0 to 00111100 with web2py (G. Clifford Williams)
1169  18   0 Cheap Helicopters In My Living Room (Ned Jackson Lovely)
1146  11   0 IPython in depth: high productivity interactive and parallel python (Fernando Perez)
1127   5   0 2D/3D graphics with Python on mobile platforms (Niko Skrypnik)
1081   8   0 Generators: The Final Frontier (David Beazley)
1067  12   0 Designing Poetic APIs (Erik Rose)
1064   6   0 Keynote - John Perry Barlow (John Perry Barlow)
1029  10   0 What Is Async, How Does It Work, And When Should I Use It? (A. Jesse Jiryu Davis)
 981  11   0 The Sorry State of SSL (Hynek Schlawack)
 961  12   2 Farewell and Welcome Home: Python in Two Genders (Naomi Ceder)
 958   6   0 Getting Started Testing (Ned Batchelder)
```

Conclusion

I hope you have found this article useful as an introduction to web scraping with Python. I have been pleasantly surprised with the use of Python, the tools are robust and powerful, and the fact that

the asynchronous optimizations can be left for the end is great compared to JavaScript, where there is no way to avoid working asynchronously from the start.

Miguel



40 comments



#1 [Weihong Guan](#) said 11 months ago

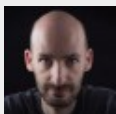
Great job. I have been working with your blog for a few month. Look forwards to the publication of your new book.



#2 said 11 months ago

Great article, Miguel!

A question: in you flask tutorial series, you opted to not use activate with virtualenv. Have you reconsidered, or is that a flask accommodation?



#3 Miguel Grinberg said 11 months ago

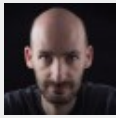
@Fraser: Yes, I have softened my views on activated virtualenvs since I wrote that first mega-tutorial article. Even for Flask, these days I'm using more the activate/deactivate scripts.



#4 Sheila said 11 months ago

This is neat -- I was wondering if you would like to use the json call to pyvideo to get a list of videos, and if so, would you like to request enhancements for what to include? It wouldn't be a good example of scraping, but maybe you could do another post about learning to call apis?

yours,
a maintainer of pyvideo :)



#5 Miguel Grinberg said 11 months ago

Sheila: I was already alerted of the fact that there is JSON info for videos by your colleague Will. I will add an appendix at the end of this article showing how to optimize my script using this data and only scraping the youtube pages. Thanks!



#6 said 11 months ago

Did you hear about the Scrapy[1]?

1 - <http://scrapy.org>



#7 Martin Betz said 10 months ago

Thanks for this tutorial. Of course we all know Scrapy, but for a case like this one with limited complexity, Scrapy seems to be a bit of an overkill. BeautifulSoup appears more appropriate!



#8 Meysam said 9 months ago

Hello Miguel, Thanks for this great tutorial.

I just run your script and get this error:

Traceback (most recent call last):

File "pycon-scraper.py", line 67, in <module>

show_video_stats(parse_args())

File "pycon-scraper.py", line 47, in show_video_stats

results = sorted(pool.map(get_video_data, video_page_urls), key=lambda video: video[options.sort],

File "/usr/lib/python2.7/multiprocessing/pool.py", line 250, in map

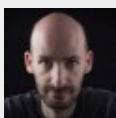
return self.map_async(func, iterable, chunksize).get()

File "/usr/lib/python2.7/multiprocessing/pool.py", line 554, in get

raise self._value

IndexError: list index out of range

what is the problem?



#9 Miguel Grinberg said 9 months ago

@Meysam: this highlights one of the problems with web scraping, which is that scripts break when the web

sites change their pages. I have an updated version of the script here:
<https://gist.github.com/miguelgrinberg/5f52ceb565264b1e969a>



#10 jeffrey said 8 months ago

Good tutorial so far, the one bit that doesn't make sense to me yet is this:

```
"video_data['views'] = int(re.sub('[^0-9]', "",  
                                soup.select('.watch-view-count')[0].get_text().split()[0]))"
```

"This first part is then filtered with a regular expression that removes any characters that are not digits, since the numbers can have commas in them."

From reading about this function `re.sub`, it basically does a find and replace? So wouldn't it find all strings that start with 0-9 and replace with an empty string? I thought you would need to search for commas, and replace them with ". Can you elaborate?



#11 Tao said 8 months ago

I tried to use the same code you uploaded here, they work fine with the static website, I just wonder if there is any way to retrieve the content from a dynamic web site, for example website using ajax? Thanks



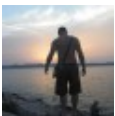
#12 Miguel Grinberg said 8 months ago

@jeffrey: the regular expression finds anything that is NOT a digit. The ^ before 0-9 inverts the selection criteria.



#13 Miguel Grinberg said 8 months ago

@Tao: for a website that runs Javascript code you need to use a more elaborated client. Try Selenium.



#14 johnnycakes79 said 8 months ago

Great Tutorial. Thanks. It nicely covers 2 things I've been hacking away at this weekend... well 3 in fact - downloading SciPy 2014 videos (automagically), scraping websites and parallel python... I've grabbed your Gist, merged it with my own massaged fork of Nick Facano's pytube and voila... it works - kinda! thanks again: https://github.com/johnnycakes79/pyvideo_scraper



#15 David said 8 months ago

My new home for the next few months :)



#16 David said 8 months ago

Great article. So what is your opinion of Python vs. Node.js specific to scraping?



#17 Miguel Grinberg said 8 months ago

@David: I prefer to make things asynchronous explicitly in Python to have the asynchronous behavior forced on me by Node. And I don't mind I don't like asynchronous code, for some things it's great, but for this type of work not so much in my opinion.



#18 said 8 months ago

Selenium is also good for scraping....great thing is it is web based application to use for scraping.....



#19 Fara said 7 months ago

How to click on link in python and moving to next page



#20 Miguel Grinberg said 7 months ago

@Fara: the concept of clicking a link does not exist when you are using scraping tools like the ones I present in this article. The equivalent would be to find the link that you are interested in, and then send a request to that link. I do that several times in the example presented above.



#21 Alfredo Conetta said 7 months ago

Just starting an MSc thesis in GIS and think that this web scrapping could be used to enable extraction of geographic context from news stories. The above info was interesting and seems to be quite straightforward.



#22 Rd Hilman Hermarian said 4 months ago

I found this article very helpful. Thanks Miguel.



#23 Xiaobo said 4 months ago

This is cool! I wonder if you ever encounter website with CAPTCHA, and were you able to scrape that?



#24 Anubhav Aggarwal said 4 months ago

Hi,

I am relatively new to python, but I have a coding background and I understand the logic. I am unable to find documentation on "soup.select". My problem is that the tag that has the href attribute that I want also defines a class. Not able to access the attribute.

```
<div class="company">
```

```
    <a class="cd dot-company" target="_blank"
```

```
href="http://qiin.en.alibaba.com/contactinfo.html" rel="nofollow" data-domdot="id:2634,sid:200947260"
```

```
data-dot="c_pid=200947260&c_type=info&cid=2634"><span class="ico-tcd"></span>Contact Details</a>
```

```
</div>
```



#25 Miguel Grinberg said 4 months ago

@Xiaobo: A website that presents a CAPTCHA is expecting humans, not robots. Trying to circumvent the CAPTCHA prompt is possible, but it is hard, unethical, and not worth the trouble in my opinion.

- [««](#)
- [«](#)
- [»»](#)
- [»](#)

Leave a Comment

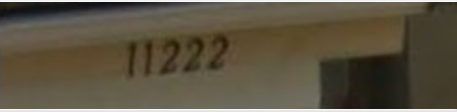
Name

URL


Email

Comment

Captcha



[Privacy & Terms](#)



Submit

This page was generated in None seconds.

© 2014 by Miguel Grinberg. All rights reserved. [Questions?](#)