Show and Tell: ASVZ Bot

BY ZENO KOLLER - SIPS BEAUTIFUL SOUP

Automation: One of the areas where technology can still provide some value to mankind, as opposed to attention-stealing apps. To be sure, I'm not talking trading bots. Rather, less distraction and more free time by programming!

Here's the situation: There's a certain ASVZ activity that I wanted to sign up for. Every year, the free spots are gone within days. So If you're not on the top of your game, checking the ASVZ site every couple of days, you're not gonna make it - I failed to sign up twice so far. This would also be my last year where I'm still in the student price tier, so I decided to build a system that notifies me when the event is online. Python to the rescue!

This was back in January, where there was another version of the ASVZ page. The first step would be to assess how to extract content from the page. I did not find any RSS feeds or any built-in notification features on the page. The site seemed to be built with Django and use AJAX to dynamically load lessons and courses. I examined the query-response format of the AJAX calls in the network tab of the browser's developer tools. The format was unfamiliar to me - at least it wasn't JSON. If the rendered HTML has any regularity to it, it would be faster to build a scraper. The developer tools revealed that indeed it did. So my notification system would look as follows:

For a given sport id and event type (lesson, course, camp):

1. Scrape the current offerings from the ASVZ page.
2. If the result is nonempty, notify me via email.

Note that this worked because there were no events online yet. I knew that once there were any, *mine* would be there. Thus, the system doesn't need to store state across multiple executions (to find out whether anything changed).

Before I describe the two steps in more detail, let me state the obvious: Of course, emailing the responsible people (I know one of them), would have taken much less effort. But we're here to learn something, aren't we?

Scraping on my scraper bike[1]

I don't often scrape web pages, but when I do, I use Python and BeautifulSoup[2]. The API is straightforward: You specify the attributes for some parts of the HTML tree you're interested in. Here is an example:

```
page = urlopen(f'{BASE_URL}?sport={requested_sport}&type={requested_type}')
soup = BeautifulSoup(page, 'html.parser')
activity_rows = soup.find('div', attrs={'class': 'some-container'}).
findAll('div', attrs={
'class': 'activity-row'})  # classes have been changed
```

To keep everything neat and tidy, I wrote a method that parses activities from HTML, which I mapped to `content_rows` to get the list of activities

```
activities = [Activity.from_html(row) for row in content_rows]
```

Not if I notify

The next step is to send me a message in case `len(activities) > 0`. I'm sure there are many ways to get a notification. I did not want to spend much time researching. One option would be to use Python's built-in `smptlib` for relaying a message via a throwaway email account. I used the `mail` utility instead and called it a day:

```
os.system(f'echo "{notification}" |  mail -s "{subject}" {recipient}')  #
Format strings rule
```

Configuring `mail` is left as an exercise to the reader.

Industrie Vier Null

Now comes the automation part[3]. If I had built a trading bot instead, I could afford a server that runs the script. I didn't, so here I am, running it locally my MacBook like a pauper[4]. How to automatically run a script every n minutes? Linux users would tell you to use `cron`, which has been deprecated on MacOS in favor of `launchd` a few years ago. You can still use `cron`, but using `launchd` trades off a ghastly XML syntax with the ability to run jobs while the laptop is „sleeping". Neat! For my mental health, I won't even start explaining how to define jobs for `launchd`[5]. Here are three tips that are likely more helpful, anyway:

1. If you use a Python script and have dependencies in virtualenv, make sure to use the absolute path to the virtualenv's Python binary.
2. The XML file allows you to define a log file path for standard output and error (for debugging).
3. From Apple's documentation: „If your daemon shuts down too quickly after being launched, launchd may think it has crashed. Daemons that continue this behavior may be suspended and not launched again when future requests arrive. To avoid this behavior, do not shut down for at least 10 seconds after launch.". Let me `time.sleep(10)` on this.

I checked that my script is working by using a sport which yield a list of results and was happy.

Aftermath

Two weeks after writing the script, ASVZ published a new version of their website, which broke my script. At first I was a bit sad, but I noticed that now, there is a JSON API. Thanks, ASVZ! All I needed to fix it was exchaning the `BeautifulSoup` dependency for `requests`, adapting the fetch part a bit and simplifying the activity parsing: ➔

```
try
    response = requests.get(API_URL.format(sport_id=SPORTS[requested_sport],
type_id=ACTIVITY_TYPES[requested_type]))
except requests.exceptions.RequestException as e:
    print(f'Well, fuck: {e}')
    sys.exit(1)
activities = [Activity.from_dict(result) for result in json.get('results',
[])]
```

The script ran happily for a few weeks, without any results. Then, for some reason, I reinstalled the virtualenv the script also uses, but forgot to install requests. In suspicion, I still checked the website manually every so often. Which was also how I saw that ASVZ published the activities. The happy end: I got in. You could learn two lessons from this:

1. Use separate virtualenvs and keep track of requirements for every project separately.
2. Or, if you're lazy, be suspicious.

Good luck with your automation attempts!

Footnotes

[1] Please listen to this while reading this part: https://www.youtube.com/watch?v=hQGLNPJ9VCE

[2] BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/

[3] One could argue that running code on a computer is already automation, but that's not what I mean here.

[4] Using a cheaper laptop would also allow for a server..

[5] Launchd sample code: https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingLaunchdJobs.html