Web Automation with Python

Travis B. Hartwell <nafai@travishartwell.net>

Utah Python User Group

2011-04-14

Why Automate?

"The three chief virtues of a programmer are: Laziness, Impatience and Hubris."

—Larry Wall

Web Automation

To automate things on the web, this is my preferred order:

1. Web service APIs using urllib, with higher-level abstractions on top

Web Automation

To automate things on the web, this is my preferred order:

1. Web service APIs using urllib, with higher-level abstractions on top
2. Simple web scraping with urllib and BeautifulSoup

Web Automation

To automate things on the web, this is my preferred order:

1. Web service APIs using urllib, with higher-level abstractions on top
2. Simple web scraping with urllib and BeautifulSoup
3. Automation with Mechanize, using BeautifulSoup for parsing

Web Automation

To automate things on the web, this is my preferred order:

1. Web service APIs using urllib, with higher-level abstractions on top
2. Simple web scraping with urllib and BeautifulSoup
3. Automation with Mechanize, using BeautifulSoup for parsing
4. Automating a web browser with Selenium

Web Service APIS

Many major services on the web today offer APIs for users and developers to access:

> *If there's an app there's an API, so we've made ours available to everyone at* `http://jobs.github.com/api`.*

These APIs are often simple enough that all you need is urllib to make call to them.

*`https://github.com/blog/836-github-jobs-update`

Example: Getting My Latest Twitter Status

Here's a simple script for getting my latest Twitter status:

```
import json
import urllib

URL_BASE = "http://twitter.com/statuses/"
STATUS_QUERY = "user_timeline/%s.json?count=1"
URL = URL_BASE + STATUS_QUERY % "travisbhartwell"

status_json = urllib.urlopen(URL).read()
status = json.loads(status_json)
print status[0][u'text']
```

Higher level wrappers around urllib

Sometimes you can make a convenient wrapper around urllib calls.
Examples:

- twitter

Higher level wrappers around urllib

Sometimes you can make a convenient wrapper around urllib calls.
Examples:

- twitter
- twipiclib

Higher level wrappers around urllib

Sometimes you can make a convenient wrapper around urllib calls.
Examples:

- twitter
- twipiclib
- facebook

Higher level wrappers around urllib

Sometimes you can make a convenient wrapper around urllib calls.
Examples:

- twitter
- twipiclib
- facebook
- gdata

Example: Uploading a picture to twitpic and posting

```
from twitpic import twitpic2
import twitter

twitpic = twitpic2.TwitPicOAuthClient(
    consumer_key = consumer_key,
    consumer_secret = consumer_secret,
    access_token = access_token,
    service_key = twitpic_api_key,
)

params = {"media": "presentation-1.png",
          "message": "Slide 2"}
response = twitpic.create('upload', params)
```

Example: Uploading a picture to twitpic and posting (cont).

```
api = twitter.Api(consumer_key=consumer_key,
                  consumer_secret=consumer_secret,
                  access_token_key=token,
                  access_token_secret=token_secret)

message = "%s: %s" % (response["text"]
status = api.PostUpdate(message,
                        response["url"]))
```

Simple Web Scraping with urllib and BeautifulSoup

The idea:
Use urllib or urllib2 from the standard library to retrieve content from the web and then parse it using a parser such as Beautiful Soup.
Useful when you just need to grab information or download links from a page and not have to manipulate forms or worry about Javascript.

Example: Downloading Media from an Archives Page

I recently used this technique to download the videos from the recent LDS General Conference. This technique could easily be applied to similar pages, like the PyCon Talk Archives
First, to start parsing, simply:

```
page = urllib.urlopen(URL)
doc = BeautifulSoup(page)
```

Example (cont.): Link Types to Search For

On viewing the source, I discovered that the download links were of this form (url shortened for convenience):

```
<a
   href="http://host/dir/talk.mp4"
   class="video-360p"
   title="mp4">
   mp4
</a>
```

Beautiful Soup makes getting links of this type easy:

```
tags = doc.findAll("a",
                   attrs={"class": "video-360p"})
```

That gives you a list of matching links that you can then act upon.

Example (cont.): Further Link Filtering

I didn't want to download all of the links from that page. For example, to exclude the musical numbers. In examining the source, I found that all of the links for those were in a table with the class "music":

```
if tag.findParents(attrs={"class": "music"}):
        continue
```

Example (cont.) Downloading the links

Finally, download the links:

```
href = tag.attrMap["href"]
urllib.urlretrieve(href)
```