QUERYING A RESTFUL API

Stateless: with every request, you send along a token/authentication of who you are

token = "super_secret_token"
r = requests.get("https://github.com/user",
    params={"access_token": token})
print( r.content )

{"login":"Mohammad Nayeem Teli","id":
10536112,"avatar_url":"ht...

• PUT/POST/DELETE can edit your repositories, etc.
• Try it out: https://github.com/settings/tokens/new
AUTHENTICATION AND OAUTH

Old and busted:

```
r = requests.get("https://api.github.com/user", auth=('nayeemmz', "cmsc320summer"))
```

New hotness:

- What if I wanted to grant an app access to, e.g., my Facebook account without giving that app my password?
- OAuth: grants access tokens that give (possibly incomplete) access to a user or app without exposing a password
“... I WILL RETURN INFORMATION IN A STRUCTURED FORMAT.”

So we’ve queried a server using a well-formed GET request via the `requests` Python module. What comes back?

General structured data:
- **Comma-Separated Value (CSV) files & strings**
- **Javascript Object Notation (JSON) files & strings**
- **HTML, XHTML, XML files & strings**

Domain-specific structured data:
- **Shapefiles**: geospatial vector data (OpenStreetMap)
- **RVT files**: architectural planning (Autodesk Revit)
- You can make up your own! **Always document it.**
CSV FILES IN PYTHON

Any CSV reader worth anything can parse files with any delimiter, not just a comma (e.g., “TSV” for tab-separated)

1,28-May,Introduction,—,"pdf, pptx",Teli,  
2,29-May,Scraping Data with Python,A naconda's Test Drive,,Teli,  
3,30-May,"Vectors, Matrices, and Dataframes",Introduction to pandas,,Teli,  
4,31-May,Jupyter notebook lab,,”Hanyu, Jue”,

Don’t write your own CSV or JSON parser

```python
import csv
with open("schedule.csv", "rb") as f:
    reader = csv.reader(f, delimiter="","", quotechar='"')
    for row in reader:
        print(row)
```
JSON FILES & STRINGS

JSON is a method for **serializing objects**: 
- Convert an object into a string (done in Java in 131/132?)
- **Deserialization** converts a string back to an object

Easy for humans to read (and sanity check, edit)
Defined by three universal data structures

- **object**
  - string
  - value

- **array**
  - value
  - value

- **value**
  - string
  - number
  - object

- Python dictionary, Java Map, hash table, etc …
- Python list, Java array, vector, etc …
- Python string, float, int, boolean, JSON object, JSON array, …

Images from: http://www.json.org/
JSON IN PYTHON

Some built-in types: "Strings", 1.0, True, False, None
Lists: [“Goodbye”, “Cruel”, “World”]
Dictionaries: {“hello”: “bonjour”, “goodbye”, “au revoir”}
Dictionaries within lists within dictionaries within lists:
[1, 2, {“Help”: [
  “I’m”, {“trapped”: “in”},
  “CMSC641”
]}}
GET https://api.twitter.com/1.1/friends/list.json?
cursor=-1&screen_name=twitterapi&skip_status=true&include_user_entities=false

{
  "previous_cursor": 0,
  "previous_cursor_str": "0",
  "next_cursor": 1333504313713126852,
  "users": [{
    "profile_sidebar_fill_color": "252429",
    "profile_sidebar_border_color": "181A1E",
    "profile_background_tile": false,
    "name": "John Doe",
    "profile_image_url": "http://a0.twimg.com/profile_images/2838630046/4b82e286a659fae310012520f4f756bb_normal.png",
    "created_at": "Thu Jan 18 00:10:45 +0000 2007", ...
}]}
Repeat: don’t write your own CSV or JSON parser

- [https://news.ycombinator.com/item?id=7796268](https://news.ycombinator.com/item?id=7796268)
- [rsdy.github.io/posts/dont_write_your_json_parser_plz.html](https://rsdy.github.io/posts/dont_write_your_json_parser_plz.html)

Python comes with a fine JSON parser

```python
import json

r = requests.get( "https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=JohnDoe", auth=auth )

data = json.loads(r.content)

json.load(some_file)  # loads JSON from a file
json.dump(json_obj, some_file)  # writes JSON to file
json.dumps(json_obj)  # returns JSON string
```
Still hugely popular online, but JSON has essentially replaced XML for:

- Asynchronous browser ←→ server calls
- Many (most?) newer web APIs

**XML is a hierarchical markup language:**

```xml
<tag attribute="value1">
  <subtag>
    Some cool words or values go here!
  </subtag>
  <openclosetag attribute="value2" />
</tag>
```

You probably won’t see much XML, but you will see plenty of HTML, its substantially less well-behaved cousin …
DOCUMENT OBJECT MODEL (DOM)
SCRAPING HTML IN PYTHON

HTML – the specification – is fairly pure
HTML – what you find on the web – is horrifying
We’ll use BeautifulSoup:
• conda install -c asmeurer beautiful-soup=4.3.2

```python
import requests
from bs4 import BeautifulSoup

r = requests.get( "https://cs.umd.edu/class/summer2019/cmsc320/"
)

root = BeautifulSoup( r.content )
root.findAll("a")  # links for CMSC320
```
Totally not hypothetical situation:
• You really want to learn about data science, so you choose to download all of last semester’s CMSC320 lecture slides to wallpaper your room …
• … but you now have carpal tunnel syndrome from clicking refresh on Piazza last night, and can no longer click on the PDF and PPTX links.

Hopeless? No! Earlier, you built a scraper to do this!

```python
lnks = root.findAll("a")  # links for CMSC320
```

Sort of. You only want PDF and PPTX files, not links to other websites or files.
REGULAR EXPRESSIONS

Given a list of URLs (strings), how do I find only those strings that end in *.pdf or *.pptx?

• Regular expressions!
• (Actually Python strings come with a built-in endswith function.)

```
"this_is_a_filename.pdf".endswith((".pdf", ".pptx"))
```

What about .pDf or .pPTx, still legal extensions for PDF/PPTX?

• Regular expressions!
• (Or cheat the system again: built-in string lower function.)

```
"tHiS_IS_a_FileNAme.pDF".lower().endswith(
   (".pdf", ".pptx"))
```
EVERYBODY STAND BACK.

I KNOW REGULAR EXPRESSIONS.

IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—

I GOT 99 PROBLEMS,

SO I USED REGULAR EXPRESSIONS.

NOW I HAVE 100 PROBLEMS.
REGULAR EXPRESSIONS

Used to **search** for specific elements, or groups of elements, that match a pattern

```python
import re

# Find the index of the 1st occurrence of "cmsc320"
match = re.search(r"cmsc320", text)
print( match.start() )

# Does start of text match "cmsc320"?
match = re.match(r"cmsc320", text)

# Iterate over all matches for "cmsc320" in text
for match in re.finditer(r"cmsc320", text):
    print( match.start() )

# Return all matches of "cmsc320" in the text
match = re.findall(r"cmsc320", text)
```
Can match sets of characters, or multiple and more elaborate sets and sequences of characters:

- Match the character ‘a’: a
- Match the character ‘a’, ‘b’, or ‘c’: \[abc\]
- Match any character except ‘a’, ‘b’, or ‘c’: \[^abc\]
- Match any digit: \d (= [0123456789] or [0–9])
- Match any alphanumeric: \w (= [a–zA–Z0–9_])
- Match any whitespace: \s (= [\t\n\r\f\v])
- Match any character: .

Special characters must be escaped: .^$*+?\{\}"[]\|()
A few common modifiers (available in Python and most other high-level languages; +, \{n\}, \{n,\} may not):

- Match character ‘a’ exactly once: a
- Match character ‘a’ zero or once: a?
- Match character ‘a’ zero or more times: a*
- Match character ‘a’ one or more times: a+
- Match character ‘a’ exactly n times: a\{n\}
- Match character ‘a’ at least n times: a\{n,\}

Example: match all instances of “University of <somewhere>” where <somewhere> is an alphanumeric string with at least 3 characters:

- \s*University\sfo\s\w\{3,\}
If you’re going to reuse the same regex many times, or if you aren’t but things are going slowly for some reason, try compiling the regular expression.

- https://blog.codinghorror.com/to-compile-or-not-to-compile/

```
# Compile the regular expression “cmsc320”
regex = re.compile(r“cmsc320”)

# Use it repeatedly to search for matches in text
regex.match( text )  # does start of text match?
regex.search( text )  # find the first match or None
regex.findall( text )  # find all matches
```
import re
import requests
from bs4 import BeautifulSoup

try:
    from urllib.parse import urlparse
except ImportError:
    from urlparse import urlparse

import the modules

# HTTP GET request sent to the URL url
r = requests.get( url )

# Use BeautifulSoup to parse the GET response
root = BeautifulSoup( r.content )
lnks = root.find_all("a")

Get some HTML via HTTP
DOWNLOADING A BUNCH OF FILES

# Cycle through the href for each anchor, checking # to see if it's a PDF/PPTX link or not
for lnk in lnks:
    href = lnk['href']

    # If it's a PDF/PPTX link, queue a download
    if href.lower().endswith(('pdf', 'pptx')):
        urld = urlparse.urljoin(url, href)
        rd = requests.get(urld, stream=True)

        # Write the downloaded PDF to a file
        outfile = "outbase"+"i"+.pdf"
        with open(outfile, 'wb') as f:
            f.write(rd.content)
NEXT:

NUMPY, SCIPY, AND DATAFRAMES

\[ y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it} \]