LECTURE 6 MINING WEB CONTENT III

LEK HSIANG HUI

OUTLINE

Web Application Design

Access APIs using Python Packages

Access APIs without using Python Packages

Scraping using an actual browser/headless browser

RECAP: TECHNIQUES FOR WEB SCRAPING

The following are some of the techniques for doing web scraping:

- Extracting content from HTML source
- Extracting content using a HTML parser
- **Web Scraping using APIs**
- Scraping using an actual browser/headless browser

WEB APPLICATION DESIGN

CLIENT/SERVER

The 2 main entities in a web environment are:

- **Client**
  - Web Browser, Mobile App, Wearables, etc
- **Server**
  - Web Server

WEB DEVELOPMENT

Even though Client/Server is still being used, the architecture of websites/web applications has evolved quite a bit

- Traditionally, users access the services through a web browser
- Now, users need not access the services through a web browser and the page/view can be dynamically generated on the client-side

TRADITIONAL WEB DEVELOPMENT

Traditional web development

- User requests a page from the server
- Server figures out what the user wants and generates the page dynamically into a HTML document before sending to user
- Client (web browser) displays the HTML document

MODERN WEB DEVELOPMENT

Modern web development

- Modern web development usually involves the use of Application Programming Interface (API)

Web API provides a mechanism for clients (browser, mobile app, etc) **to consume services or extend the capability of the site**

- Example:
  Consume services: get the first 20 records, add a Weibo post, etc
  Extend capability: new mobile app to access social media, etc

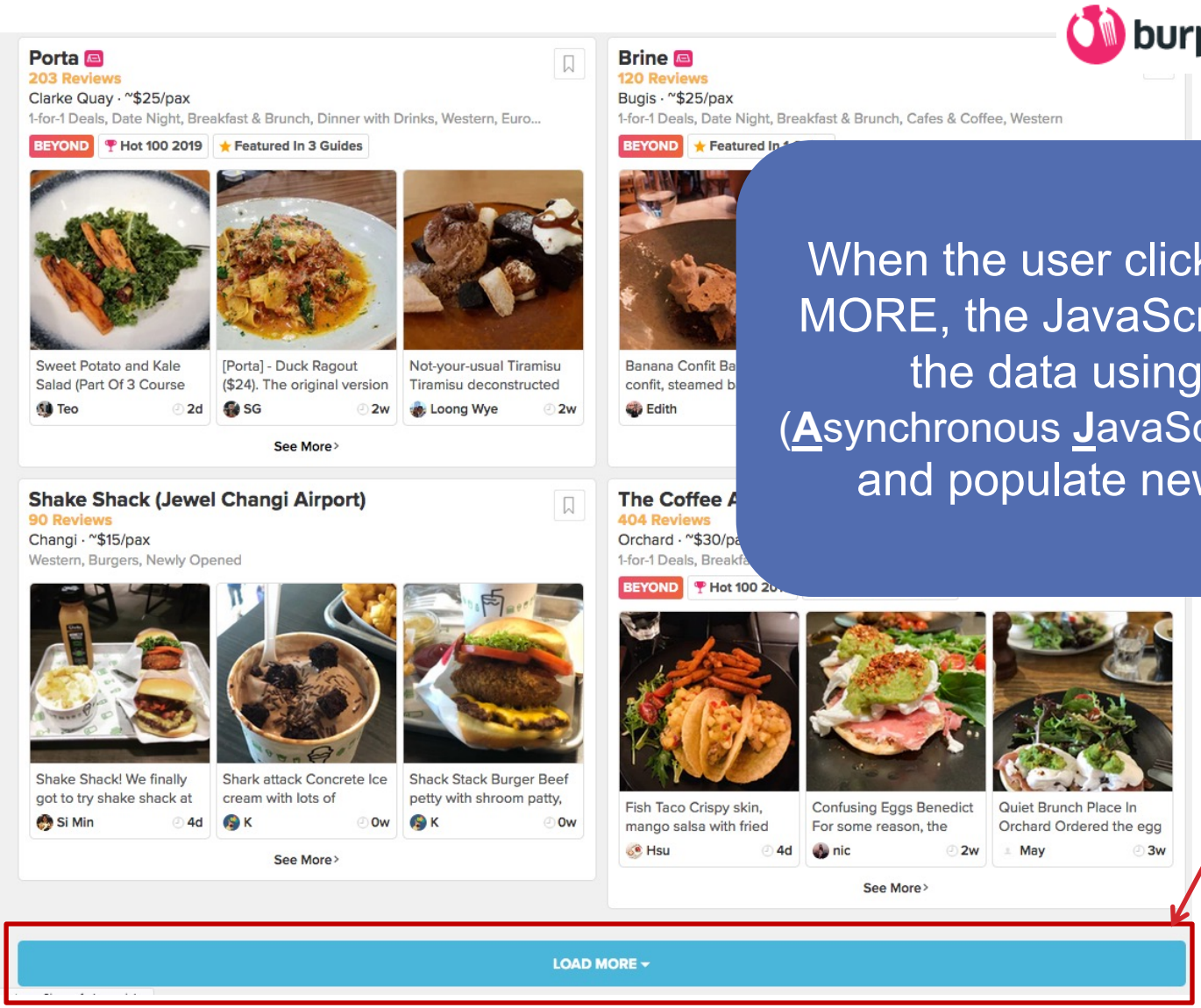
MODERN WEB DEVELOPMENT

Most web API uses a data exchange format for communication rather than sending HTML (resulting in lesser data transfer – faster)

- Data exchange formats: XML, JSON (more common, to be discussed later)

Some APIs are created and used only by the creator (private API), where others are meant to be used for anyone (public API)

PRIVATE API USAGE



When the user clicks on LOAD MORE, the JavaScript will fetch the data using **AJAX** (**A**synchronous **J**avaScript **A**nd **X**ML) and populate new records

PUBLIC API USAGE

nominatim.openstreetmap.org/sear × | Result for school of computing × | +

https://nominatim.openstreetmap.org/search?q=school of computing Singapore&format=json&addressdetails=1 133%

JSON | Raw Data | Headers

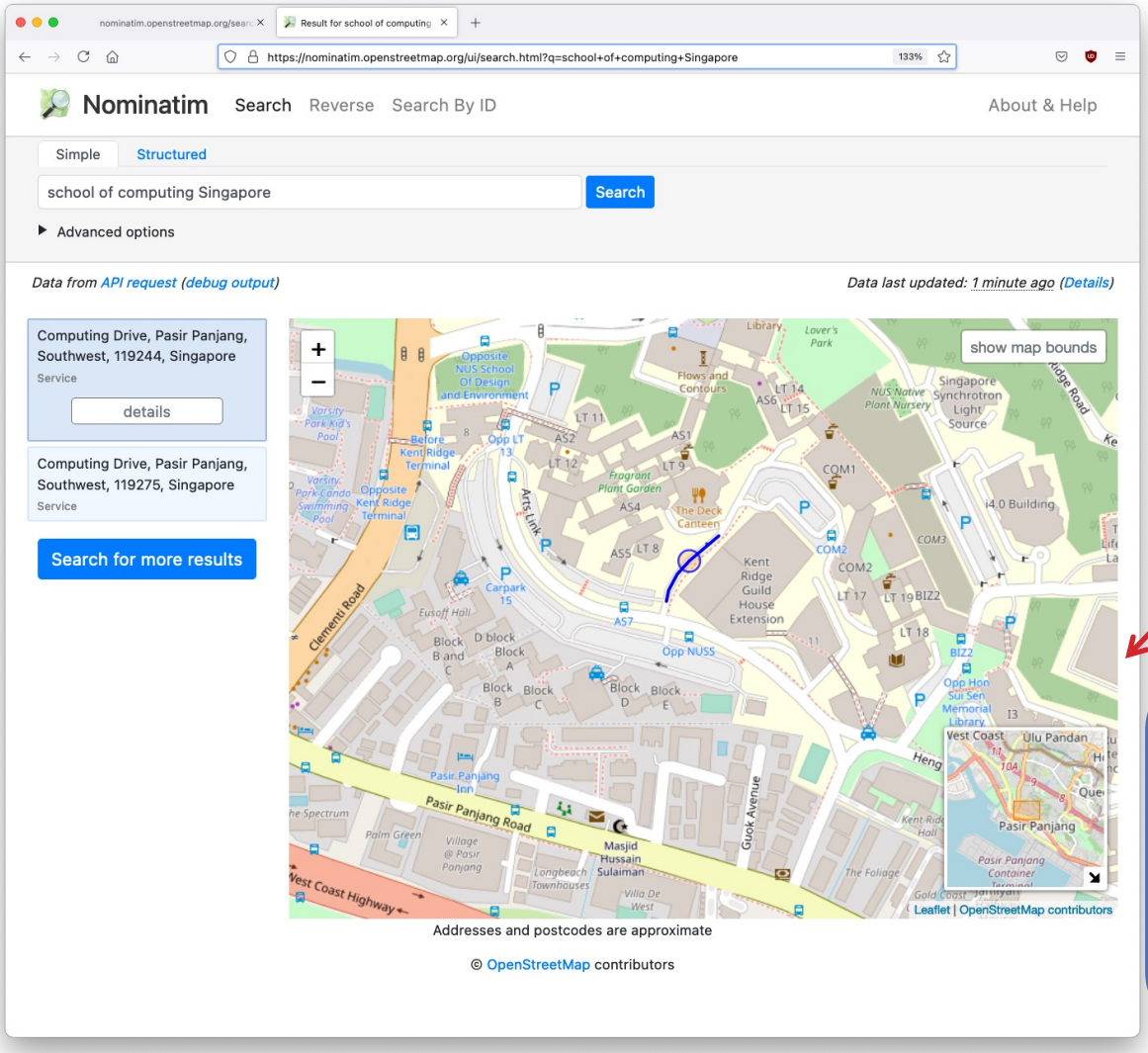
Save | Copy | Collapse All | Expand All | Filter JSON

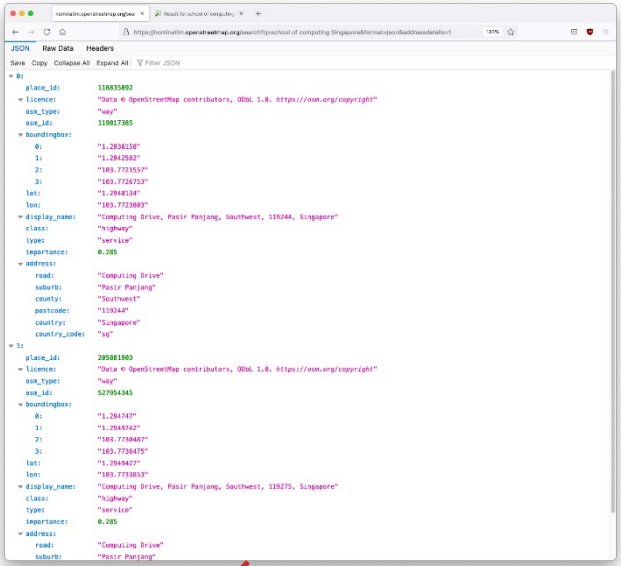

```
▼ 0:
    place_id:        118835092
  ▼ licence:         "Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright"
    osm_type:        "way"
    osm_id:          119817385
  ▼ boundingbox:
      0:             "1.2936158"
      1:             "1.2942582"
      2:             "103.7721557"
      3:             "103.7726753"
    lat:             "1.2940134"
    lon:             "103.7723803"
  ▼ display_name:    "Computing Drive, Pasir Panjang, Southwest, 119244, Singapore"
    class:           "highway"
    type:            "service"
    importance:      0.285
  ▼ address:
      road:          "Computing Drive"
      suburb:        "Pasir Panjang"
      county:        "Southwest"
      postcode:      "119244"
      country:       "Singapore"
      country_code:  "sg"
▼ 1:
    place_id:        205881903
  ▼ licence:         "Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright"
    osm_type:        "way"
    osm_id:          527954345
  ▼ boundingbox:
      0:             "1.294747"
      1:             "1.2949742"
      2:             "103.7730487"
      3:             "103.7736475"
    lat:             "1.2949427"
    lon:             "103.7733853"
  ▼ display_name:    "Computing Drive, Pasir Panjang, Southwest, 119275, Singapore"
    class:           "highway"
    type:            "service"
    importance:      0.285
  ▼ address:
      road:          "Computing Drive"
      suburb:        "Pasir Panjang"
```


https://nominatim.openstreetmap.org/search?q=13%20computing%20drive,%20Singapore&format=json&addressdetails=1

URL to access the OpenStreetMap API

PUBLIC API USAGE





Public API can be used programmatically, and at the same time to power websites

EXTERNAL API USAGE

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON


```
▼ 0:
    place_id:          118835092
  ▼ licence:           "Data © OpenStreetMap contributors, ODbL 1.0. https
    osm_type:          "way"
    osm_id:            119817385
  ▼ boundingbox:
      0:               "1.2936158"
      1:               "1.2942582"
      2:               "103.7721557"
      3:               "103.7726753"
    lat:               "1.2940134"
    lon:               "103.7723803"
  ▼ display_name:      "Computing Drive, Pasir Panjang, Southwest, 119244, Singapore"
    class:             "highway"
    type:              "service"
    importance:        0.285
  ▼ address:
      road:            "Computing Drive"
      suburb:          "Pasir Panjang"
      county:          "Southwest"
      postcode:        "119244"
      country:         "Singapore"
      country_code:    "sg"
```


Example of the **JSON** data exchange format

It allows any data to be represented in a text format together with its structure

ACCESS APIS IN PYTHON PACKAGES

ACCESSING API USING PYTHON PACKAGES

Before you try to work with an API manually, could check whether there is a package to access the API

We will use the OSMPythonTools package as an example

- OSMPythonTools allow us to access the OpenStreetMap services
- https://wiki.openstreetmap.org/wiki/OSMPythonTools

ACCESSING API MANUALLY

Not all site has a package that integrates with the API

- Furthermore, each time when an API changes, it might potentially break the packages that using with the API

Better to learn how to access APIs manually without packages

- Need to learn how to work with JSON output

HANDS-ON: WEB SCRAPING USING API

Download and access:
Webscraping using API.ipynb

jupyter Webscraping using API Complete Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

OpenStreetMap

Using OpenStreetMap Python Package (OSMPythonTools)

Install using `pip install OSMPythonTools` (in the Anaconda Prompt)

More documentation examples at https://wiki.openstreetmap.org/wiki/OSMPythonTools

Sourcecode : https://github.com/mocnik-science/osm-python-tools

In [1]:
```
from collections import OrderedDict
from OSMPythonTools.nominatim import Nominatim
from OSMPythonTools.overpass import overpassQueryBuilder, Overpass
from OSMPythonTools.data import Data, dictRangeYears, ALL
import requests
import urllib.parse
```

In [2]:
```
nominatim = Nominatim()
place = nominatim.query("13 computing drive, Singapore")
```

In [3]:
```
place.toJSON()
```

Out[3]:
```
[{'place_id': 118835092,
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright',
  'osm_type': 'way',
  'osm_id': 119817385,
  'boundingbox': ['1.2936158', '1.2942582', '103.7721557', '103.7726753'],
  'lat': '1.2940134',
  'lon': '103.7723803',
```

CONCLUDING NOTES ON API

Should use API as far as possible instead of other approaches

- Lesser data payload → faster

****But take note that most API adopts rate limiting**

- Should not violate the rate limits if not you might be banned!
- Protip: If the service you are accessing requires you to authenticate, use another dummy account

SCRAPING USING AN ACTUAL BROWSER

RECAP: TECHNIQUES FOR WEB SCRAPING

The following are some of the techniques for doing web scraping:

- Extracting content from HTML source
- Extracting content using a HTML parser
- Web Scraping using APIs
- **Scraping using an actual browser/headless browser**

If the site loads contents dynamically and does not offer an API, you will have to use the last technique

SCRAPING USING ACTUAL BROWSER/HEADLESS BROWSER

This is the most powerful technique

- *"Anything that you see on the browser can be scraped"*

Not all site offers an API and sometimes the page is rendered dynamically using JavaScript, by looking at the HTML source code, you will not be able to extract the data

- They probably did not intend you to do web scraping on the site!

Headless browser = browser without a GUI

DISCLAIMER

It is a gray area whether you are allowed to do web scraping

- Most site would not allow you to do that
- But everyone is doing it anyways

Some things are outrightly illegal such as selling data that you do not own, etc

This section is meant for educational purposes, you are responsible for your own actions ☺

Suppose we want to scrap data off the Baidu search engine

The page is loaded based on what the user enters in the search box

SCRAPING USING ACTUAL BROWSER/HEADLESS BROWSER

Idea:

- Load an instance of a browser
- Navigate the browser instance to the starting page
- Programmatically perform certain actions on the page
- The page is rendered accordingly on the browser instance
- Perform web scraping on the browser instance

To do this in Python, we can make use of the selenium package

Download and access:
Webscraping using Selenium.ipynb

HANDS-ON: WEB SCRAPING USING BROWSER

jupyter Webscraping using Selenium (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Code

Selenium

Install using:

`pip install selenium`

`pip install chromedriver-binary==91.0.4472.101.0` (need to make sure that the Chrome driver matches the Chrome that's installed in your machine)

Check https://pypi.org/project/chromedriver-binary/#history

in the Anaconda Prompt

In [1]:
```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.chrome.options import Options
import chromedriver_binary
from selenium.webdriver.common.keys import Keys
import pandas as pd
```

In [2]:
```
driver = webdriver.Chrome()
driver.get("https://www.baidu.com/")
```

In [3]:
```
#find the search box
searchbox = driver.find_element_by_css_selector("#kw")
searchbox.send_keys("predictive analytics")
searchbox.send_keys(Keys.RETURN)
```

WEB SCRAPING USING BROWSER INSTANCE

Advantages:

- Pretty much can handle any website
- Simulate as if it were someone doing actual web-surfing but we can now programmatically grab contents from the page

Disadvantages:

- Slow (would want to open up an actual browser, load css, images, etc)
    - Can try disable loading of images

CONCLUDING NOTES ON USING SELENIUM

There are other technologies that runs much faster

- PhantomJS, Puppeteer (based on NodeJS)
- In my opinion, web scraper using NodeJS is much more natural (since NodeJS is Javascript (JS) and logic on the web is powered by JS)

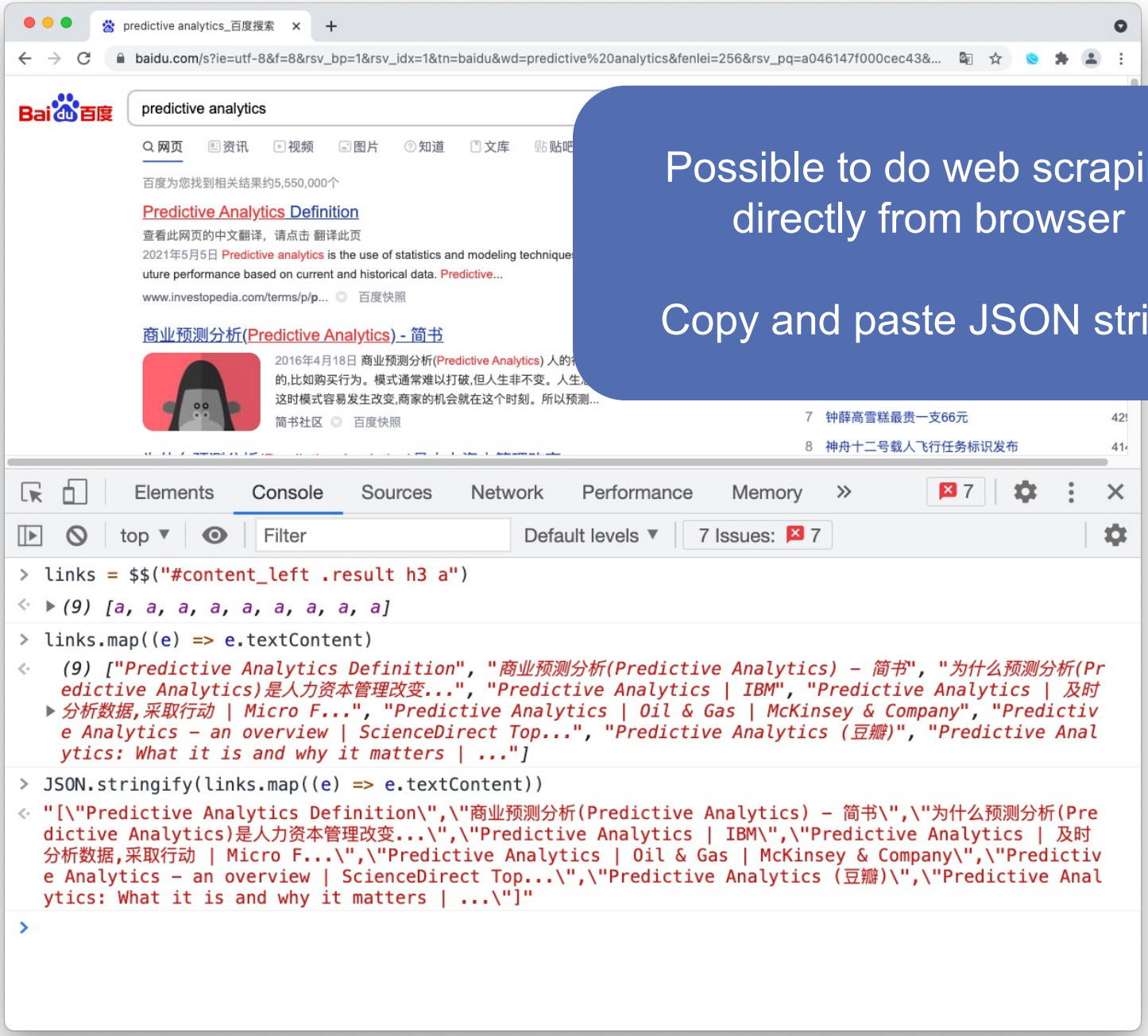
Don't adopt Selenium as the first choice

- Generally, this approach of web scraping is slow (even if using NodeJS)
- Public API > Private API > HTML Parsing > Selenium

CONCLUDING NOTES ON USING SELENIUM

Practically speaking…

- Don’t have to code every action, some actions which are difficult to code, you could manually do it on the browser
- Being good at Javascript/CSS selectors is probably more important than knowing how to code in Selenium
- Possible to do web scraping directly on the browser!



Possible to do web scraping directly from browser

Copy and paste JSON string

SUMMARY

Web Application Design

Access APIs in Python

- Using packages and accessing using URL

Scraping using an actual browser/headless browser

WHAT'S NEXT?

Recommender System