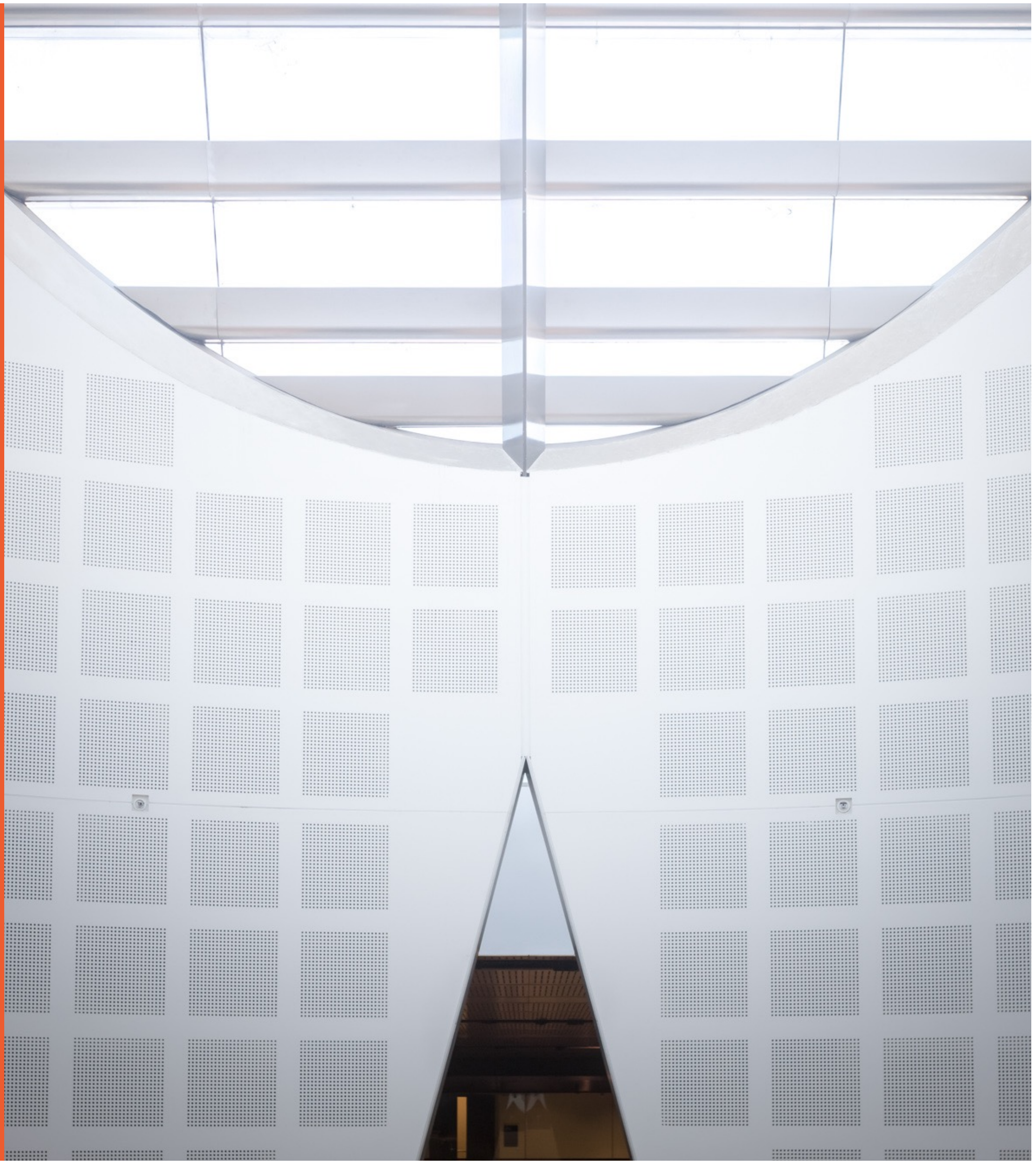
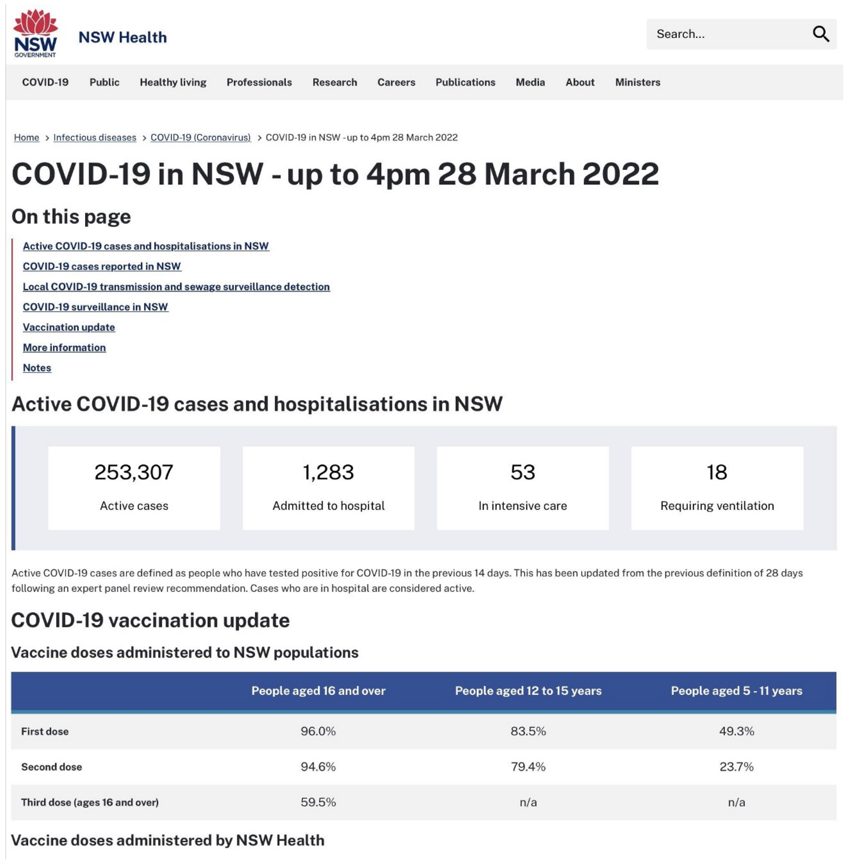
COMP5339: Data Engineering
Week 4: Web Scraping and Web API

Presented by

Uwe Roehm
School of Computer Science



Motivating Example: How can we extract this data?



NSW Health

Search...

COVID-19 Public Healthy living Professionals Research Careers Publications Media About Ministers

Home > Infectious diseases > COVID-19 (Coronavirus) > COVID-19 in NSW -up to 4pm 28 March 2022

COVID-19 in NSW - up to 4pm 28 March 2022

On this page

- Active COVID-19 cases and hospitalisations in NSW
- COVID-19 cases reported in NSW
- Local COVID-19 transmission and sewage surveillance detection
- COVID-19 surveillance in NSW
- Vaccination update
- More information
- Notes

Active COVID-19 cases and hospitalisations in NSW

253,307	1,283	53	18
Active cases	Admitted to hospital	In intensive care	Requiring ventilation

Active COVID-19 cases are defined as people who have tested positive for COVID-19 in the previous 14 days. This has been updated from the previous definition of 28 days following an expert panel review recommendation. Cases who are in hospital are considered active.

COVID-19 vaccination update

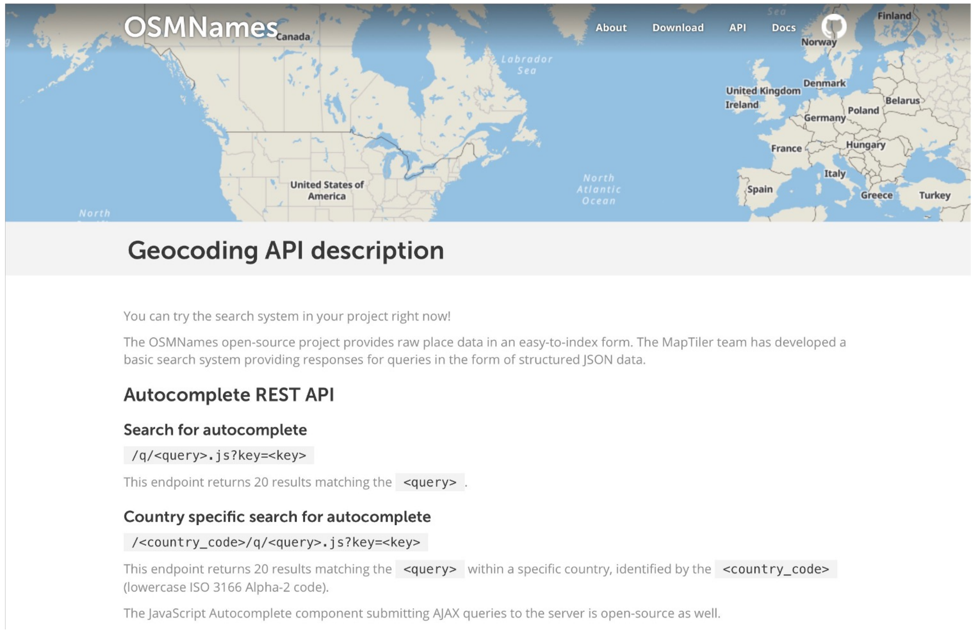
Vaccine doses administered to NSW populations

	People aged 16 and over	People aged 12 to 15 years	People aged 5 - 11 years
First dose	96.0%	83.5%	49.3%
Second dose	94.6%	79.4%	23.7%
Third dose (ages 16 and over)	59.5%	n/a	n/a

Vaccine doses administered by NSW Health

Web APIs

- Many websites or web services provide programmable APIs which allow you to explicitly request data.
- Public interfaces
  - Eg. Google Maps or OpenStreetMap APIs
- Official APIs requiring registration
  - Eg. https://opendata.transport.nsw.gov.au
  - https://data.gov.au – registry of open data
- Closed partner interfaces (B2B)
  - Eg https://www.airbnb.com.au/partner
- Unofficial 3rd party APIs…



Geocoding API description

You can try the search system in your project right now!

The OSMNames open-source project provides raw place data in an easy-to-index form. The MapTiler team has developed a basic search system providing responses for queries in the form of structured JSON data.

Autocomplete REST API

Search for autocomplete

```
/q/<query>.js?key=<key>
```

This endpoint returns 20 results matching the `<query>`.

Country specific search for autocomplete

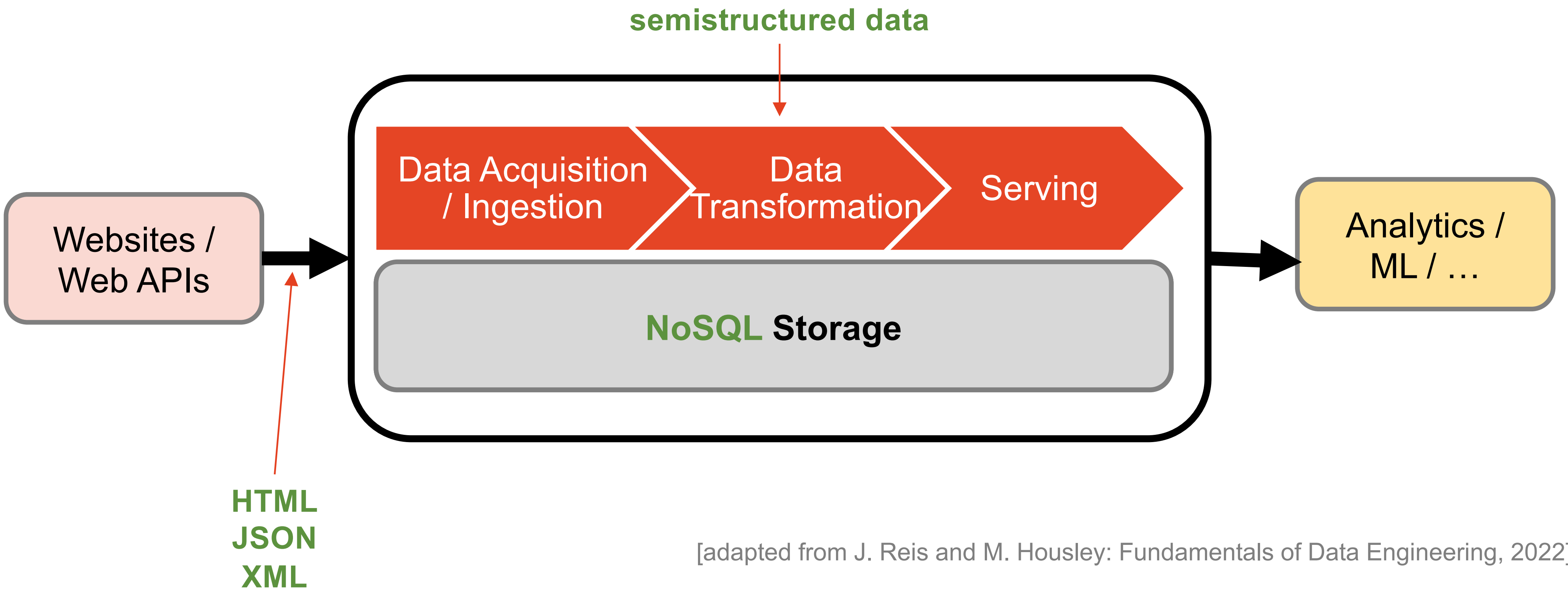
```
/<country_code>/q/<query>.js?key=<key>
```

This endpoint returns 20 results matching the `<query>` within a specific country, identified by the `<country_code>` (lowercase ISO 3166 Alpha-2 code).

The JavaScript Autocomplete component submitting AJAX queries to the server is open-source as well.

Source: https://osmnames.org/api/

Data Engineering Lifecycle – Web Data



[adapted from J. Reis and M. Housley: Fundamentals of Data Engineering, 2022]

Web Scraping

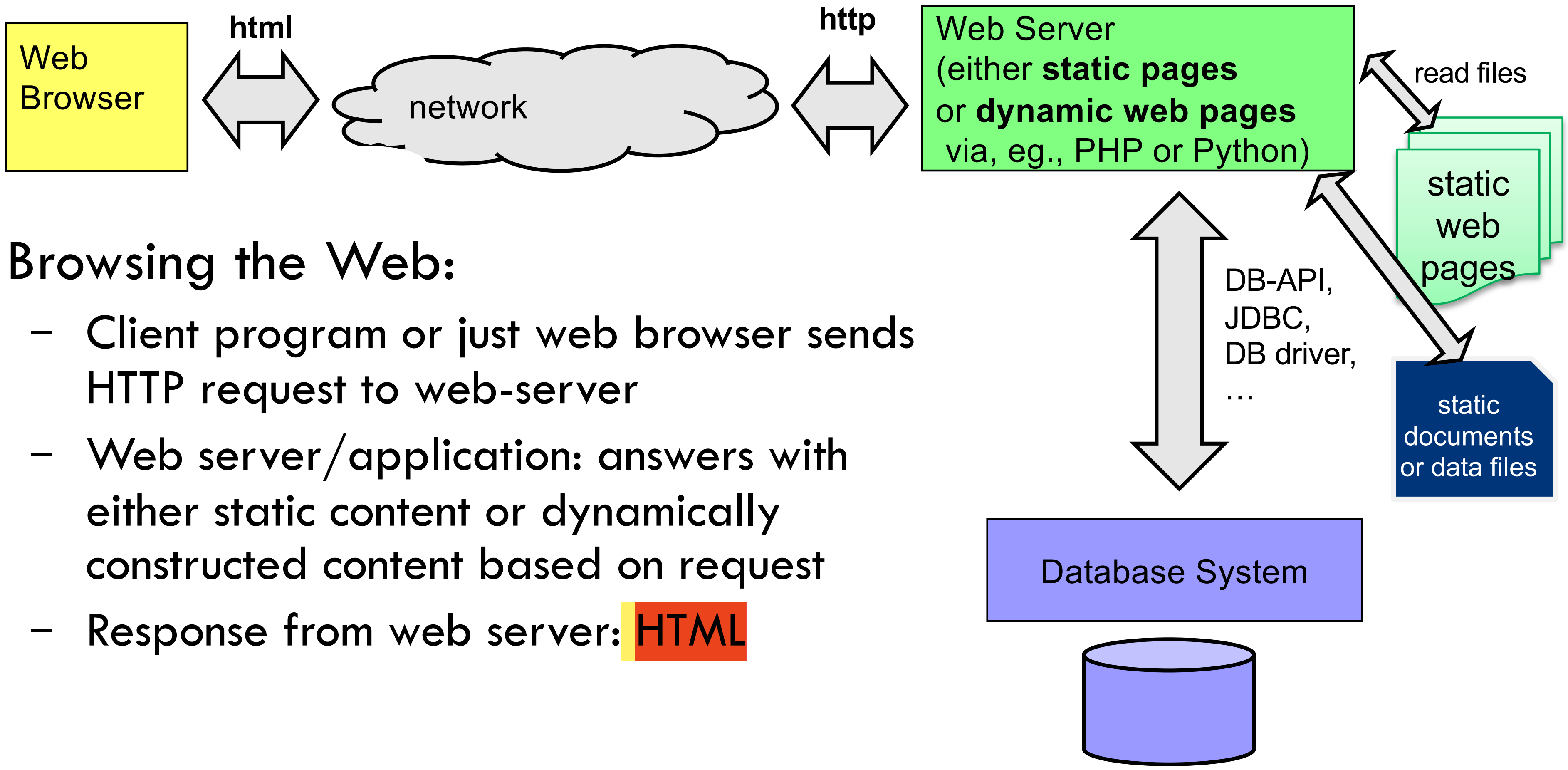
THE UNIVERSITY OF SYDNEY

Acquiring Data from Websites

- Two approaches:
  - either download files using web requests
  - or scrape (embedded) data from web pages
- In both cases, we start with an HTTP request, but then interpret the result differently
- For example in Python, there are several support libraries to access the web
  - Pandas library supports reading structured data directly from a URL too
  - Example Code:

```
import pandas as pd
url="https://data.gov.au/.../airline_portcountry.csv"
content = pd.read_csv(url)
```

Overview of Web Requests



Web Browser
html
network
http
Web Server (either **static pages** or **dynamic web pages** via, eg., PHP or Python)
read files
static web pages
DB-API, JDBC, DB driver, …
static documents or data files
Database System

- Browsing the Web:
  - Client program or just web browser sends HTTP request to web-server
  - Web server/application: answers with either static content or dynamically constructed content based on request
  - Response from web server: HTML

Web Scraping – General Approach

- **Reconnaissance**
  - Identify source, and check its structure and content
  - Check terms-of-service and robots.txt; if in doubt, contact owner
- **Webpage Retrieval**
  - Download one or multiple pages from source
  - Typically in a script or program that generates new URLs based on website structure and its URL format
- **Data Extraction** from webpage
  - Content parsing, raw data extraction
- **Data Cleaning** and transformation into the required format
- **Data Storage** / Analysis / combining with other data sets

Scraping the Web

- How to get information from web pages that have data embedded?
- Web pages are written in **HTML** which is a semi-structured data format with some similarity to XML

```
<html>
  <head>
    <title>Data Engineering</title>
  </head>
  <body>
    <h1><span class="uoscode">COMP5339</span> - Data Engineering</h1>
    <div class="lecturer">Lijun Chang</div>
    <p class="description">COMP5339 is about …</p>
  </body>
</html>
```

Scraping the Web (cont'd)

- HTML is not always well-formed, let alone annotated or semantically marked up
- Many HTML parsers are too strict for real-world usage
  - Would stop parsing incorrectly-written web pages without giving us a chance to extract data
- Web browsers are very forgiving when interpreting HTML pages
- There are several 3rd party support libraries or tools to help us parse poorly structured HTML

Which Tools?

Lots of tools and programming frameworks available:

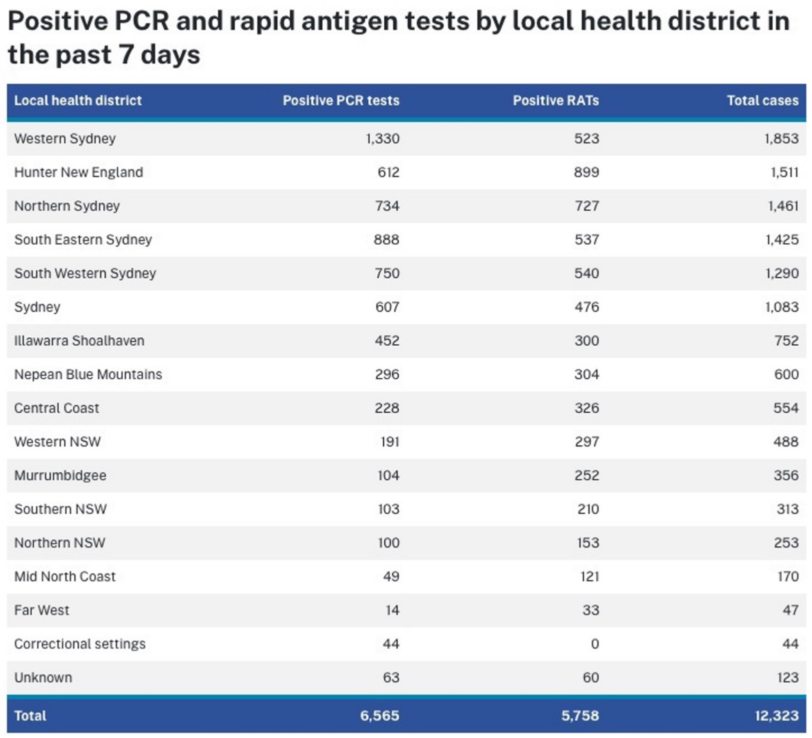
- Unix command line tool
  - curl, grep, awk, …
- 3rd Party tools
  - eg. Google spreadsheets (ImportHTML() function)
  - Many commercial solutions with nice 'click' interfaces and visualisations
    - Example: Import.io, many more…
  - WebCrawlers-as-a-Service (eg. Scrapinghub)
- Programming libraries
  - Eg. Pandas, **BeautifulSoup** library for Python; or frameworks like Scrapy

Example: NSW COVID-19 Data

- https://www.health.nsw.gov.au/news/Pages/20220329_00.aspx

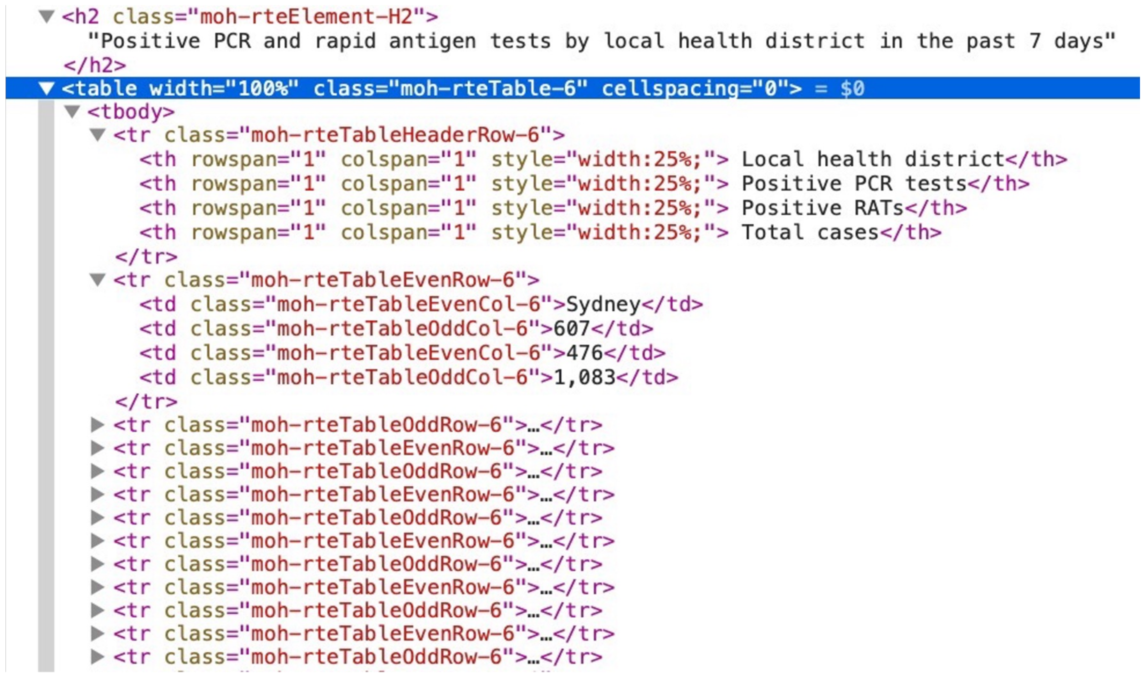
- Inspect the logic and structure of the website
  - Inspect webpage structure
    - Reasonable HTML including some annotation and classes to identify data parts easily
    - Note any URL patterns

Example: Inspecting HTML Code of Websites



Positive PCR and rapid antigen tests by local health district in the past 7 days

Local health district	Positive PCR tests	Positive RATs	Total cases
Western Sydney	1,330	523	1,853
Hunter New England	612	899	1,511
Northern Sydney	734	727	1,461
South Eastern Sydney	888	537	1,425
South Western Sydney	750	540	1,290
Sydney	607	476	1,083
Illawarra Shoalhaven	452	300	752
Nepean Blue Mountains	296	304	600
Central Coast	228	326	554
Western NSW	191	297	488
Murrumbidgee	104	252	356
Southern NSW	103	210	313
Northern NSW	100	153	253
Mid North Coast	49	121	170
Far West	14	33	47
Correctional settings	44	0	44
Unknown	63	60	123
Total	6,565	5,758	12,323



```
▼ <h2 class="moh-rteElement-H2">
    "Positive PCR and rapid antigen tests by local health district in the past 7 days"
  </h2>
▼ <table width="100%" class="moh-rteTable-6" cellspacing="0"> = $0
  ▼ <tbody>
    ▼ <tr class="moh-rteTableHeaderRow-6">
        <th rowspan="1" colspan="1" style="width:25%;"> Local health district</th>
        <th rowspan="1" colspan="1" style="width:25%;"> Positive PCR tests</th>
        <th rowspan="1" colspan="1" style="width:25%;"> Positive RATs</th>
        <th rowspan="1" colspan="1" style="width:25%;"> Total cases</th>
      </tr>
    ▼ <tr class="moh-rteTableEvenRow-6">
        <td class="moh-rteTableEvenCol-6">Sydney</td>
        <td class="moh-rteTableOddCol-6">607</td>
        <td class="moh-rteTableEvenCol-6">476</td>
        <td class="moh-rteTableOddCol-6">1,083</td>
      </tr>
    ▶ <tr class="moh-rteTableOddRow-6">…</tr>
    ▶ <tr class="moh-rteTableEvenRow-6">…</tr>
    ▶ <tr class="moh-rteTableOddRow-6">…</tr>
    ▶ <tr class="moh-rteTableEvenRow-6">…</tr>
    ▶ <tr class="moh-rteTableOddRow-6">…</tr>
    ▶ <tr class="moh-rteTableEvenRow-6">…</tr>
    ▶ <tr class="moh-rteTableOddRow-6">…</tr>
    ▶ <tr class="moh-rteTableEvenRow-6">…</tr>
    ▶ <tr class="moh-rteTableOddRow-6">…</tr>
    ▶ <tr class="moh-rteTableEvenRow-6">…</tr>
    ▶ <tr class="moh-rteTableOddRow-6">…</tr>
```

For single webpages, Google spreadsheet can help

- ImportHTML
  - URL
  - “list” or “table”
  - Index of which list or table to import from webpage

- Example (in Google Spreadsheet):

```
ImportHTML("https://www.health.nsw.gov.au/news/Pages/20220329_00.aspx",
"table", 1)
```

Some Tips

- URL format is important to take note of
  - Look at any parameters (e.g. flight data from Airport OnTime dataset: http://transtats.bts.gov/PREZIP/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_**2021_1**.zip - the year and month are parameterized and can be looped over)
  - Find any patterns with links when accessing data (i.e. do they do it monthly, yearly, bi-weekly etc.)
  - Access tokens (i.e. do they pass an API key or?)
- Web page structure is useful to note.
  - Use the page inspector to narrow down what you're looking for.
  - Complex tokenising can get messy (we might have to tokenise child nodes of the elements)
    - but tools like Beautiful Soup for Python can help

Summary

- Data ingestion from websites
  - Option 1: download data files or documents using HTTP requests
  - Option 2: scrape (embedded) data from web pages
- Web Scraping:
  - Structured approach including Reconnaissance, Web Retrieval, Data Extraction, …
  - Various tools available, well scriptable nowadays
- Being a good net citizen: respect terms-of-use, copyright, robots.txt

Web Scraping with Python

THE UNIVERSITY OF SYDNEY

Web Page Retrieval in Python

- Two forms of requests: **GET** (optional with parameters)
  or **POST** (with params)
- Make simple web requests in Python
  - Python `requests` library:
    standard webpage: `requests.get(`*`URL`*`)`
    webpage with parameters: `requests.get(`*`URL`*`, params=dict(`*`key=value,…`*`))`
    web form (POST request): `requests.post(`*`URL`*`, params=dict(`*`key=value,…`*`))`
- Example:

```
import requests
from bs4 import BeautifulSoup
response = requests.get("http://www.example.com")
print(response.status_code) # inspect response code of server
content = BeautifulSoup(response.text, 'html5lib')
```

Web Page Retrieval: URLs

- URL – Uniform Resource Locator
  - "address" format on the web
  - Example: https://www.health.nsw.gov.au/news/Pages/20220329_00.aspx
  - General Format
    - **protocol://site/*path_to_resource***
    - Typical protocols: http, https, ftp
    - `site` can be a domain name or IP address, followed by :port number
      - my.domain.com:2100
    - `path_to_resource` is usually a sequence of names separated by slashes
      - often contains encoded values: `service?name=fred&address=...`

Web Site Crawling in Python (multiple page scraping)

- **Scrapy**
  - Extensive Python framework to implement a web ‘spider’ – a program that follows multiple links along the web
  - Can extend this spider class with own functionality which extracts parts of the visited pages while the spider follows further links
  - https://docs.scrapy.org/en/latest/intro/overview.html

- **Selenium**
  - A programmable web browser for which a Python binding exists which allows to actually send requests as if a user would have clicked on links or used a page (including running javascripts)
  - Typically used for automatic testing of websites
  - But can also be used for ‘crawling’ a complex interactive website
  - https://selenium-python.readthedocs.io

HTML – Hypertext Markup Language

- Webpages are written in HTML
  - Textual markup language that defines **structure**, **content**, and **design** of a page as well as active elements (scripts, forms, etc.)
  - Typically several additional files linked:
    - CSS - cascading style sheets
    - Scripts, Images, videos etc.
- Markup via open & closing **tags** in (e.g. <title>…</title>)
  - Pre-defined in HTML standards (http://www.w3.org)
- Interpreted by web browsers for display
  - HTML is designed to be interpreted by programs

HTML Example

```
<!DOCTYPE html>
<html>
 <head>
  <title>Literature List…</title>
 </head>
 <body>
   <h1>References</h1>
   <p>The following are some interesting links on web scraping:</p>
    <div id="biblist">
    <ul>
      <li> "Data Science From Scratch", Chapter 23 </li>
      <li> <a href="http://blog.danwin.com/examples-of-web-scraping-in-python-3-x-for-data-journalists/">Web
      Scraping for Data Journalists</a>   </li>
       …
    </ul>
    </div> …
</body>
</html>
```

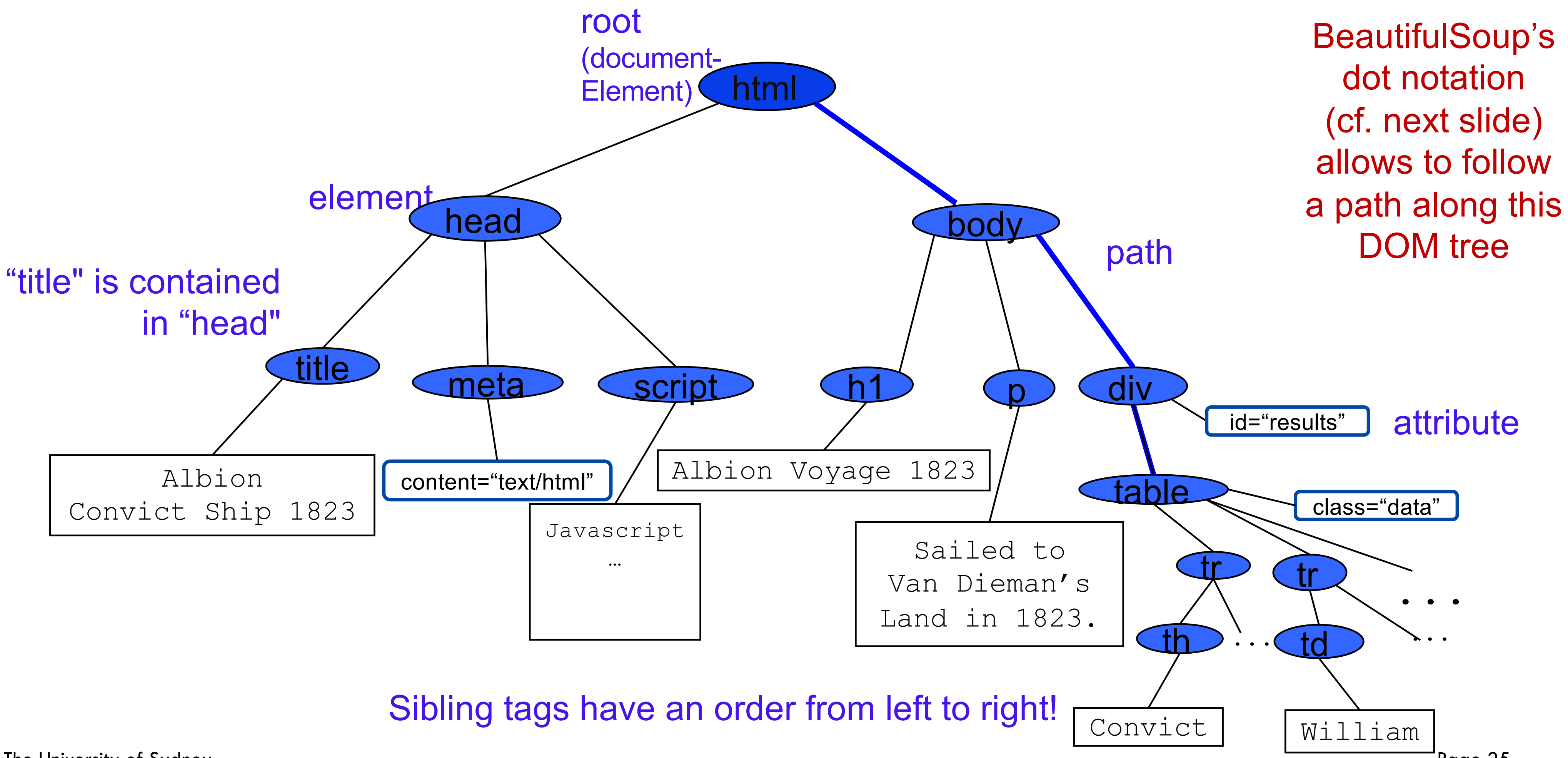
General Structure of a Web Page

- Head
  - **title**, style sheets, scripts, meta-data
- Body
  - headings, text, lists, tables, images, forms etc.
- Wide variety of quality of web pages
  - Some pages are automatically generated from CMS => not really human-readable
  - Some are heavy on design elements, others are more “structured”
- Web Page inspector of Web Browser
  - Good for Reconnaissance Phase

How to Select Content in a Webpage?

- Four options:
  - Text patterns
    - simple, but not really great for complex patterns as we rely on some own parsing…
  - DOM navigation
    - Document object model
  - CSS selectors
    - based on the tag types, class specifications and IDs elements
    - easy to specify, but depends on CSS classes and IDs being well used
  - XPath expressions
    - powerful language that allows to navigate along document tree and select all nodes or even sub-trees which match the path expr.
    - can contain filter predicates, e.g. on values of XML/HTML attributes

HTML Document Model (DOM): Element-Tree



BeautifulSoup's dot notation (cf. next slide) allows to follow a path along this DOM tree

Sibling tags have an order from left to right!

Content Extraction with BeautifulSoup

Example for DOM-based navigation and data extraction:

```python
from bs4 import BeautifulSoup
page_content = BeautifulSoup(webpage_source, 'html5lib')

# Example 1: print the title element of the page content
print("Example 1: get a specific HTML element - such as the page title")
print(page_content.title)

# Example 2: print just the text of the title element using the 'text' operator
print("\nExample 2: get HTML element's text content")
print(page_content.title.string)

# Example 3: navigate along a tag sequence path and print content
print("\nExample 3: navigate along an element path to some content inside the page")
print(page_content.body.div['id'])

# Example 4: find ALL hyperlinks on the page (anywhere on the page, just print URL)
print("\nExample 4: find_all() URLs of hyperlinks on this webpage")
for link in page_content.find_all("a"):
    if (link.has_attr('href')):
        print(link.get('href'));
```

Content Extraction into Pandas DataFrame

```
import pandas as pd
import requests
from bs4 import BeautifulSoup

response = requests.get("http://www.example.com")
print(response.status_code) # inspect response code of server
content = BeautifulSoup(response.text, 'html5lib’)

table = content.find_all(‘table’)[0]
df = pd.read_html(str(table))[0]          # only works with HTML tables
countries = df[“COUNTRY”]
print(countries)
```

CSS Selectors

- HTML elements can have multiple CSS class attributes associated for display, as well as an ID attribute for identification
  - Example: <table class=”data” id=“42”>
- CSS Selectors (the most important ones):
  - Selecting an element e with a specific class: *e.class*
    - E.g. <table class=“data”> => table.data
  - Selecting an element e by ID: e#*id*
    - E.g. <div id=“results”> => div#results or just #results
  - Selecting by position within a parent element
    - e:first-child e:last-child e:nth-child(*n*)
  - Selecting instances out of multiple occurrences
    - e:nth-of-type(*n*)
  - …

Using CSS Selectors with BeautifulSoup

- BeautifulSoup provides several functions that support CSS selectors:
  `find()`   `find_all()`   `select()`
- Examples: (assuming `page_content` is a parsed webpage)
  - Find an element by type:
    ```
    elements = page_content.find_all("h3")
    for e in elements: …
    ```
  - Find an element by id:
    ```
    element = page_content.find(id="ship")
    ```
  - Find table elements with a specific CSS class:
    ```
    element = page_content.find_all("table", "data")
    ```
  - Look for tags matching general (complex) CSS selector:
    ```
    elements = page_content.select("#ship.data")
    for e in elements: …
    ```

Scraping HTML with Pandas

```
import pandas as pd

# pandas can also read directly from a URL – but only tables!
dfs = pd.read_html('https://www.health.nsw.gov.au/news/Pages/20220329_00.aspx')
# scrapes tables as a list(!) of DataFrames
dfs[0].tail()

# plot as bar chart
%matplotlib inline
import matplotlib.pyplot as plt
f = plt.figure()
plt.title("Covid case sources in NSW")
dfs[0].plot.bar(x="LHD", y="Total cases", ax=f.gca())
```

Summary

- Web scraping / data retrieval is well scriptable
  - Comprehensive support for requesting and interpreting web data in Python

- HTML – Hypertext Markup Language
  - Document model (DOM) is tree-structured

- Identifying relevant information
  - URL structure for navigating dynamic web pages
  - DOM–navigation or CSS selectors to identify relevant HTML elements

Web API Programming

THE UNIVERSITY OF
SYDNEY

Getting Data via Service-APIs (Web Services)

- Many website or web service provide programmable APIs which allow you to explicitly request data
- Typical result formats:
  - JSON

```
{ "uoscode" : "COMP5310",
  "title" : "Principles of Data Science",
  "lecturers" : [ "Ben Hachey", "Uwe Roehm" ],
  "description" : "COMP5310 is about …" }
```

  - XML

```
<?xml version="1.0">
  <uoscode>COMP5310</uoscode>
  <title>Principles of Data Science</title>
  <lecturers>
     <name>Ben Hachey</name>
     <name>Uwe Roehm</name>
  </lecturers>
  <description>COMP5310 is about …</description>
```

Web Service Standards

- RESTful APIs (also: RESTful web services)
  - REST: Representational State Transfer
  - Stateless operations performed on a simple, uniform API using standard HTTP requests (eg. GET or PUT)
  - requests made to a resource's URL responded with HTML, XML, or JSON
- XML Web Services
  - XML-message based (both ways: request and response)
  - Based on the “Simple Object Access Protocol” (SOAP), but actually the much more complex (but also more powerful) standard
- If we refer to web service in this course, we mean REST APIs

JSON Overview

- JSON: JavaScript Object Notation
  - A text-based, semi-structured format for data interchange
  - Basically: nested key-value pairs with types Number, String, Boolean & Array
  - Originates from object serialisation in JavaScript
  - Promoted as low-overhead (read: less verbose) format vs. XML
  - Have widespread support nowadays in systems

https://tools.ietf.org/html/rfc7159

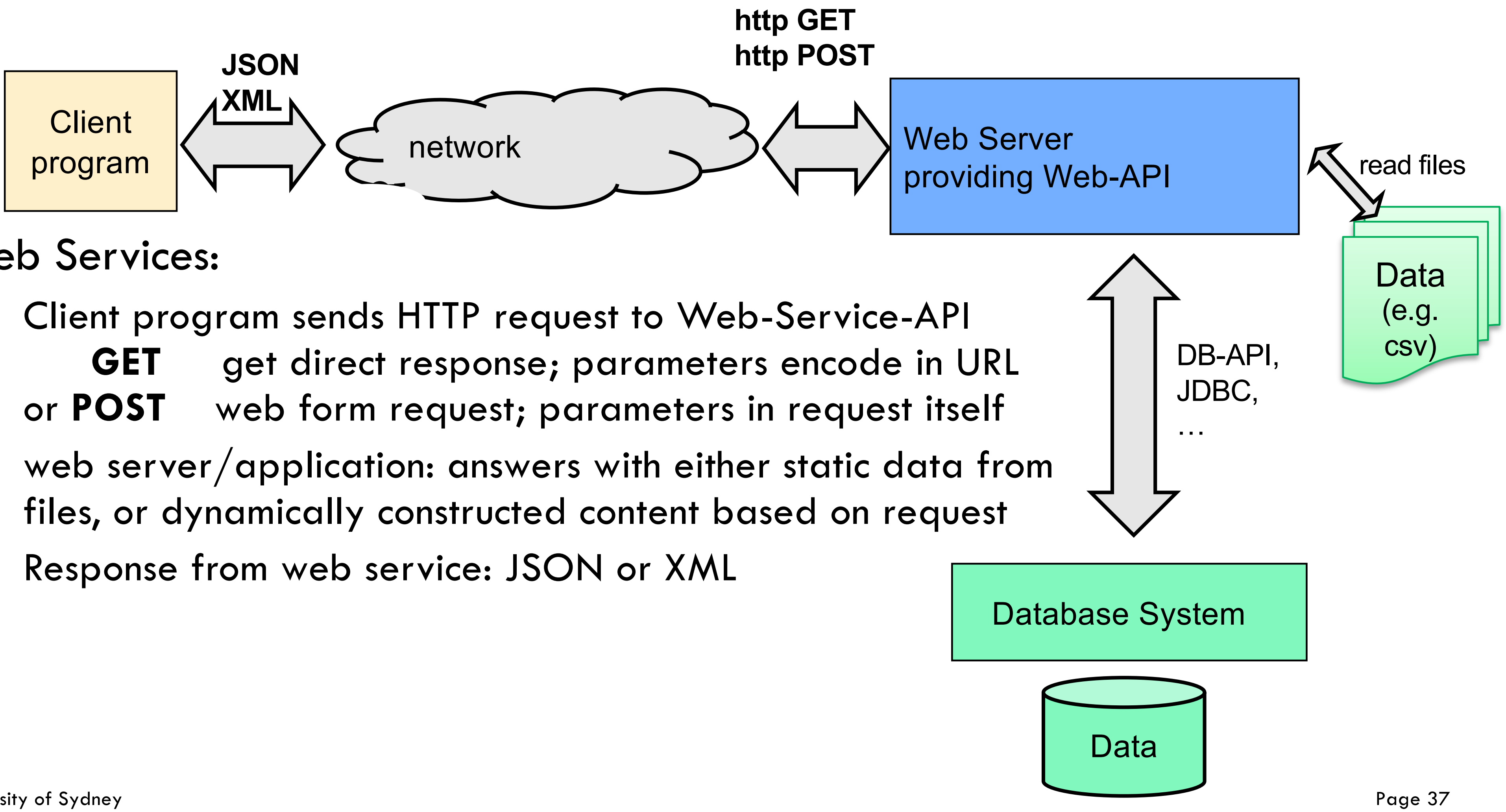
JSON Overview

Unordered **sets** of name-value pairs with nested structure.

JSON element	Description
{	Begins a JSON object, which might contain zero or more values/objects.
[	Begins an array of JSON objects or values.
"name":value, "name":value	One or more name-value pairs, separated by commas. Value can be an atomic value, a nested JSON object or an array.
]	Ends an array.
}	Ends a JSON object.

Supported Data Types: (quoted) Strings, Numbers, Boolean, **null**
(plus objects and arrays)

Overview of Web Services



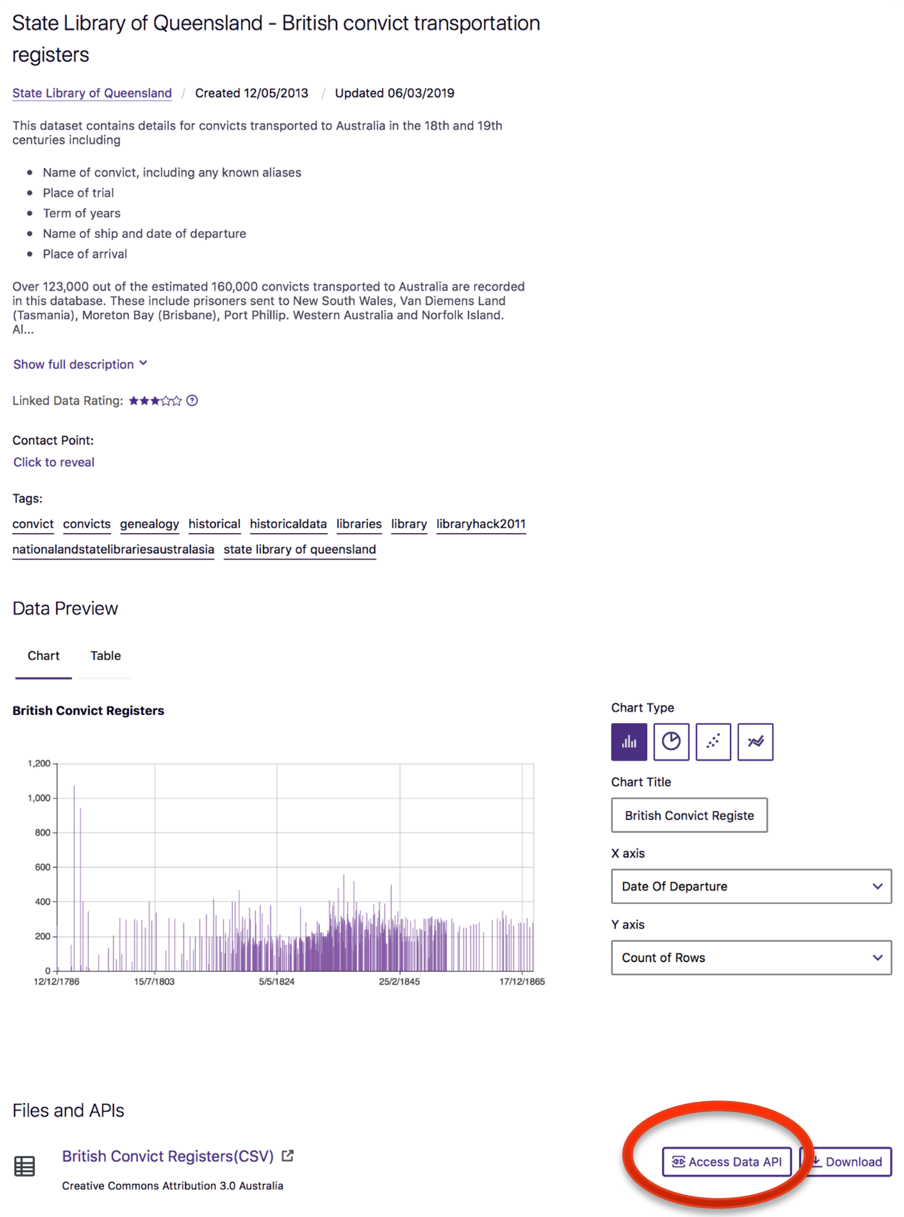
- Web Services:
  - Client program sends HTTP request to Web-Service-API
    GET get direct response; parameters encode in URL
    or **POST** web form request; parameters in request itself
  - web server/application: answers with either static data from files, or dynamically constructed content based on request
  - Response from web service: JSON or XML

Example: British Convict Transport Registers from SLQ

```
import requests
import json

url='https://www.data.qld.gov.au/api/3/action/datastore_
search?resource_id=dbcfa4a6-3ec7-4264-bcee-43b21a470d34’
Limit = '5'
Query = 'Asia’
response = requests.get(url+"&limit="+limit+"&q="+query)
search_result = response.json()

for r in search_result["result"]["records"]:
    print(r["Vessel"],"\t", r["Date of Departure"],
                        "\t", r["Place of Arrival"])
```



State Library of Queensland - British convict transportation registers

State Library of Queensland / Created 12/05/2013 / Updated 06/03/2019

This dataset contains details for convicts transported to Australia in the 18th and 19th centuries including

- Name of convict, including any known aliases
- Place of trial
- Term of years
- Name of ship and date of departure
- Place of arrival

Over 123,000 out of the estimated 160,000 convicts transported to Australia are recorded in this database. These include prisoners sent to New South Wales, Van Diemens Land (Tasmania), Moreton Bay (Brisbane), Port Phillip. Western Australia and Norfolk Island. Al...

Show full description

Linked Data Rating: ★★★☆☆

Contact Point:
Click to reveal

Tags:
convict convicts genealogy historical historicaldata libraries library libraryhack2011 nationalandstatelibrariesaustralasia state library of queensland

Data Preview

Chart Table

British Convict Registers

Chart Type

Chart Title: British Convict Registe

X axis: Date Of Departure

Y axis: Count of Rows

Files and APIs

British Convict Registers(CSV)
Creative Commons Attribution 3.0 Australia

Web-APIs / JSON and Pandas

- Pandas can transform a JSON string into a DataFrame (or Series)

```
import pandas as pd
import requests
response = requests.get(base_url + "&q=Eliza")
df = pd.read_json(response.text)
```

- You might need to select the appropriate part first:

```
import pandas as pd
df = pd.json_normalize(response.json()['result']['records'])
df[["Vessel","Date of Departure","Place of Arrival"]]
```

Working with Web APIs

- Further Python examples that look up some public information of github repos:

```
import requests
endpoint = "https://api.github.com/users/postgres/repos"
response = requests.get(endpoint)
repos    = response.json()

last_3_repositories = sorted(repos,
                             key=lambda r: r["created_at"],
                             reverse=True)[:3]

print("Most recent 3 repositories:")
for repo in last_3_repositories:
    print(repo["name"], ":", repo["language"])
```

```
# the same using Pandas
import requests
import pandas as pd
endpoint = "https://api.github.com/users/postgres/repos"
response = requests.get(endpoint)

df = pd.read_json(response.text)
df.sort_values(by='created_at', ascending=False, inplace=True)
df[['name','language','created_at']].head(3)
```

Example: OpenStreetMap API

[https://nominatim.org/release-docs/develop/api/Search/]

- Some APIs also allow additional parameters to be given
- Simple Python program that looks up a given address
  - Note: many APIs require some sort of authentication such as an API key

```
import requests, json
base_url = ‘https://nominatim.openstreetmap.org/search’
my_params= {'q': '1 Cleveland Street,Darlington,Australia','format':'json’}
headers  = {‘User-Agent’: ‘COMP5339’}
response = requests.get(base_url, params=my_params, headers=headers)

print(json.dumps(response.json(), indent=4, sort_keys=False))

results  = response.json()
geo_lat  = results[0][‘lat’]
geo_lon  = results[0][‘lon']
print(geo_lat, geo_lon)
```

Summary

- Web APIs allow dynamic, loosely coupled systems
- Useful for
  - data ingestion,
  - Part of the data transformation, and
  - serving data downstream
- Meant for automatic, programmable use
  - no overhead to extract useful data from presentation or style information
- Semi-structured data as response – JSON or XML