ISA 414 – Managing Big Data

Lecture 10 – Data Collection

APIs (Part II)

Dr. Arthur Carvalho

arthur.carvalho@miamioh.edu

MIAMI UNIVERSITY

Lecture Objectives

- Continue to learn about APIs and how they are used during data collection
  - Understand the data exchange format called XML

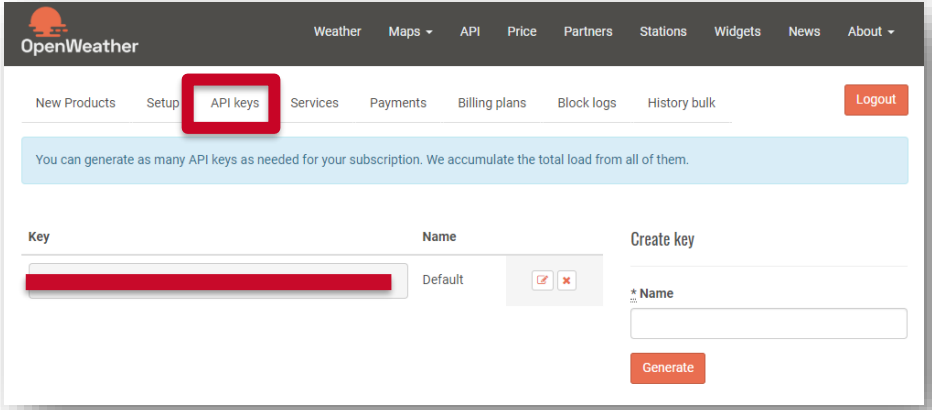
Lecture Instructions (Part I)

- Download the notebook "*Lecture 10.ipynb*" available on Canvas
  - Open the above file with VS Code

- Download the file *cdlib.xml* available on Canvas (optional)

Lecture Instructions (Part II)

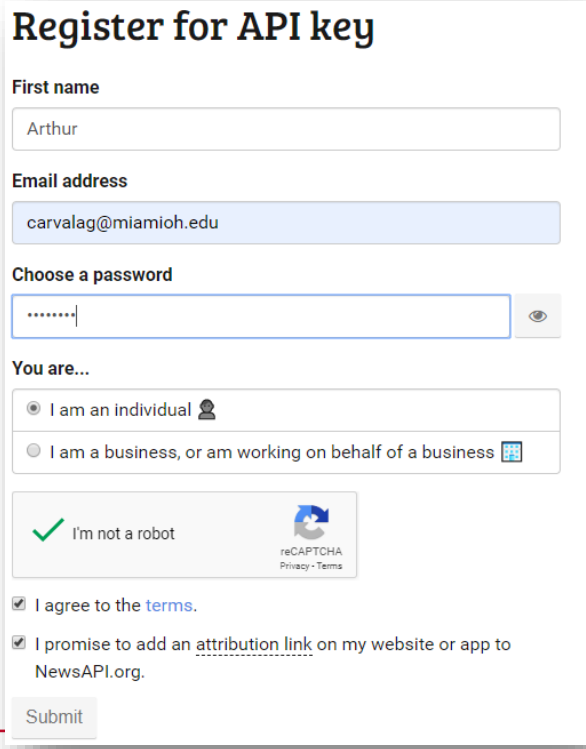
- Obtain an API key from OpenWeather
  1. Go to https://home.openweathermap.org/users/sign_up
     - Fill in the text fields
     - Click on “Create Account”
  2. Go to “API keys” or “My API Keys” under your username
     - Save your key



MIAMI UNIVERSITY

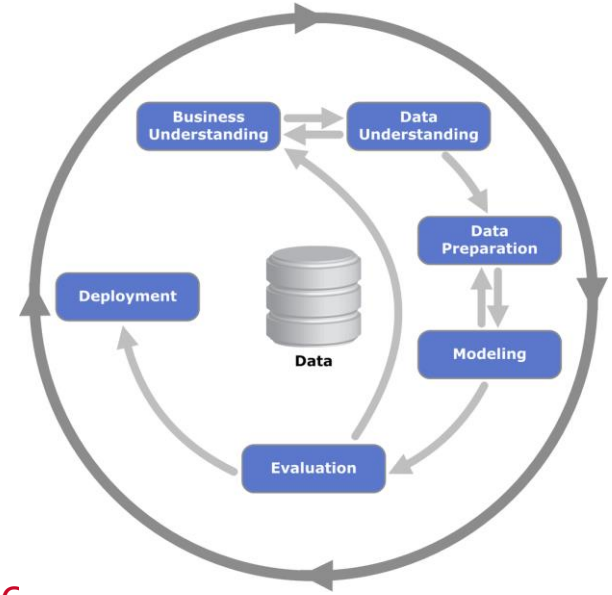
Lecture Instructions (Part III)

- Obtain an API key from News API
  - Go to https://newsapi.org
  - Click on “Get API key”
  - Fill in the forms and click on “Submit”
  - Save your API key



CRISP-DM

- From previous lecture
  - Big data (data analytics) project management



- We are currently focusing on data understanding/collection
  - To a less degree, on data preparation

MIAMI UNIVERSITY

Data Collection

- What have we learned up to know?
  - Scrap (unstructured) data from the web
    - Retrieve relevant data from HTML files
      - Regular expressions
  - Collect data via APIs
    - Request: REST + HTTP
    - Response: JSON

Data Collection

- What does come next?
  - Today: collect data via APIs
    - Request: REST + HTTP
    - Response: XML
  - Next lectures: collect data from databases
    - Querying a document-oriented database (MongoDB)
      - Unstructured data

Data Collection

- Data are assets
  - Not all relevant data are freely available on the web
  - Many companies make money by directly or indirectly selling data, *e.g.*, Twitter, Google, Facebook
    - How can we access these companies' data?
      - Via APIs
    - Which protocol must we follow?
      - REST (most popular), SOAP, …
    - What is the format of the provided data?
      - JSON, XML, …

XML

- eXtensible Markup Language
  - Markup language for data exchange
    - Similar to HTML, there are tags that indicate the beginning and end of each field in an XML file
    - But different than HTML, XML is not used for formatting texts
      - Instead, it is used to store data

Text formatting

HTML	XML
MIST7600 Data Management 3 credit hours	<course> <code>MIST7600</code> <title>Data Management</title> <credit>3</credit> </course>

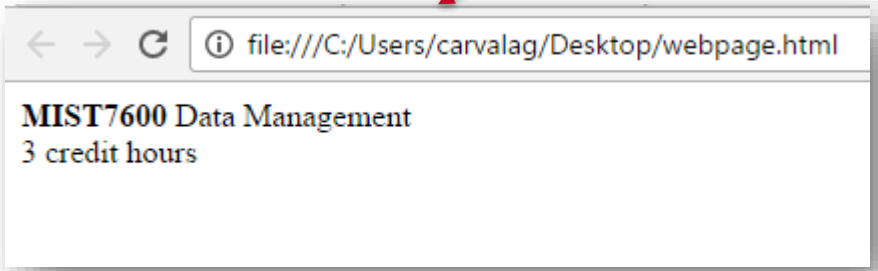
Data about a course

MIAMI UNIVERSITY

XML

HTML	XML
MIST7600 Data Management 3 credit hours	<course> <code>MIST7600</code> <title>Data Management</title> <credit>3</credit> </course>

equivalent to



equivalent to

Table COURSE

Code	Title	Credit
MIST7600	Data Management	3

MIAMI UNIVERSITY

XML

- Major rules
  - Elements must have both an opening and closing tag
    - Not always true with HTML (*e.g.*, <br>)
  - Elements must follow a strict tree-like hierarchy with only one root element
  - XML is case sensitive

XML

- Example: describing a library of CDs with XML
  - A library contains many CDs
  - Each CD might contain:
    - An ID
    - Title
    - Release year
    - Multiple tracks
  - Each track contains:
    - Track number
    - Track title
    - Track length

XML

- Example: describing a library of CDs with XML

Line 1: XML declaration
Line 2: root element called *cdlibrary*
Line 3: details of each CD in the library are in between the tags <cd> and </cd>
Line 8: details of each track are in between the tags <track> and </track>

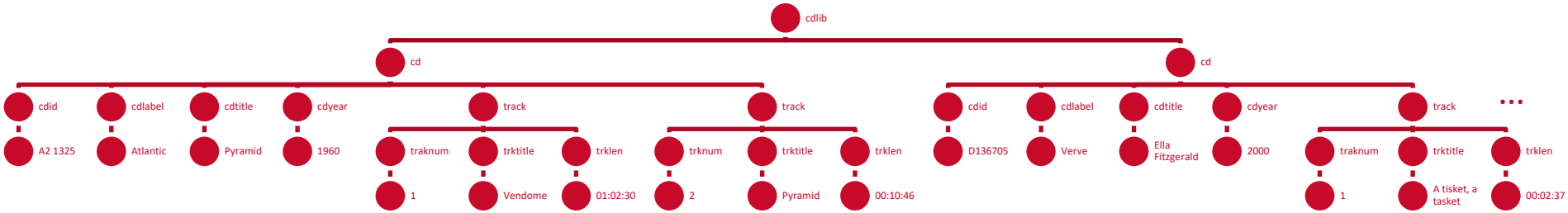
```
<?xml version = "1.0" encoding= "UTF-8"?>
<cdlibrary>
    <cd>
        <cdid>A2 1325</cdid>
        <cdlabel>Atlantic</cdlabel>
        <cdtitle>Pyramid</cdtitle>
        <cdyear>1960</cdyear>
        <track>
            <trknum>1</trknum>
            <trktitle>Vendome</trktitle>
            <trklen>00:02:30</trklen>
        </track>
        …
    </cd>
    …
</cdlibrary>
```

XML

- Example: describing a library of CDs with XML
  - Let’s take a closer look at the previous example
    - Open the file *cdlib.xml* (available on Canvas) with VS Code
  - This example clearly shows that XML files have a hierarchical structure


```xml
<?xml version="1.0" encoding="UTF-8"?>
- <cdlibrary>
  - <cd>
      <cdid>A2 1325</cdid>
      <cdlabel>Atlantic</cdlabel>
      <cdtitle>Pyramid</cdtitle>
      <cdyear>1960</cdyear>
    - <track>
        <trknum>1</trknum>
        <trktitle>Vendome</trktitle>
        <trklen>01:02:30</trklen>
      </track>
    - <track>
        <trknum>2</trknum>
        <trktitle>Pyramid</trktitle>
        <trklen>00:10:46</trklen>
      </track>
    </cd>
  - <cd>
      <cdid>D136705</cdid>
      <cdlabel>Verve</cdlabel>
      <cdtitle>Ella Fitzgerald</cdtitle>
      <cdyear>2000</cdyear>
    - <track>
        <trknum>1</trknum>
        <trktitle>A tisket, a tasket</trktitle>
        <trklen>00:02:37</trklen>
      </track>
    - <track>
        <trknum>2</trknum>
        <trktitle>Vote for Mr. Rhythm</trktitle>
        <trklen>00:02:25</trklen>
      </track>
    - <track>
        <trknum>3</trknum>
        <trktitle>Betcha nickel</trktitle>
        <trklen>00:02:52</trklen>
      </track>
    </cd>
  </cdlibrary>
```

XML



MIAMI UNIVERSITY

XML

- Let's analyze XML files inside Python
  - Background story
    - You work for a company that predicts the attendance of public events
      - Event organizers pay you for that information, and use it for adjusting ticket prices
    - A key predictor of event attendance is the weather
      - Your company decide to collect weather data from a reliable source and incorporate that in a predictive model
        - Open Weather: https://openweathermap.org/

XML

- Example: current weather in Cincinnati
  - Let's request an XML file with data about the current weather condition in Cincinnati and build an XML tree
    - API documentation: https://openweathermap.org/current
    - Request: REST HTTP; Response: XML

```
import requests
import xml.etree.ElementTree as ET

url = "http://api.openweathermap.org/data/2.5/weather?q=Cincinnati&mode=xml&appid=YOUR_API_KEY"
response = requests.get(url)

with open('weather.xml', 'wb') as file:
    file.write(response.content)
```

XML

- Example: current weather in Cincinnati
  - Let's get the root node and all its children from the tree

```
tree = ET.parse('weather.xml')
root = tree.getroot()
```


```
▼<current>
 ▼<city id="4508722" name="Cincinnati">
    <coord lon="-84.51" lat="39.1"/>
    <country>US</country>
    <timezone>-14400</timezone>
    <sun rise="2019-09-23T11:26:23" set="2019-09-23T23:34:36"/>
  </city>
  <temperature value="295.36" min="294.15" max="296.48" unit="kelvin"/>
  <humidity value="94" unit="%"/>
  <pressure value="1016" unit="hPa"/>
 ▼<wind>
    <speed value="3.1" unit="m/s" name="Light breeze"/>
    <gusts/>
    <direction value="230" code="SW" name="Southwest"/>
  </wind>
  <clouds value="90" name="overcast clouds"/>
  <visibility value="16093"/>
  <precipitation value="0.25" mode="rain" unit="1h"/>
  <weather number="500" value="light rain" icon="10d"/>
  <lastupdate value="2019-09-23T14:57:14"/>
</current>
```

XML

- Example: current weather in Cincinnati
  - Let's navigate through the XML file
    - Looking at the name and number of children of the root node

```
print(root.tag)
print(len(root))

for child in root:
    print(child.tag)
```


```xml
<current>
  <city id="4508722" name="Cincinnati">
    <coord lon="-84.51" lat="39.1"/>
    <country>US</country>
    <timezone>-14400</timezone>
    <sun rise="2019-09-23T11:26:23" set="2019-09-23T23:34:36"/>
  </city>
  <temperature value="295.36" min="294.15" max="296.48" unit="kelvin"/>
  <humidity value="94" unit="%"/>
  <pressure value="1016" unit="hPa"/>
  <wind>
    <speed value="3.1" unit="m/s" name="Light breeze"/>
    <gusts/>
    <direction value="230" code="SW" name="Southwest"/>
  </wind>
  <clouds value="90" name="overcast clouds"/>
  <visibility value="16093"/>
  <precipitation value="0.25" mode="rain" unit="1h"/>
  <weather number="500" value="light rain" icon="10d"/>
  <lastupdate value="2019-09-23T14:57:14"/>
</current>
```

XML

- Example: current weather in Cincinnati
  - Let's navigate through the XML file
    - Obtaining the 1st child of the root node

      city = root[0]

    - Retrieving the time zone value

      print(city[2].text)


```
▼<current>
  ▼<city id="4508722" name="Cincinnati">
    <coord lon="-84.51" lat="39.1"/>
    <country>US</country>
    <timezone>-14400</timezone>
    <sun rise="2019-09-23T11:26:23" set="2019-09-23T23:34:36"/>
  </city>
  <temperature value="295.36" min="294.15" max="296.48" unit="kelvin"/>
  <humidity value="94" unit="%"/>
  <pressure value="1016" unit="hPa"/>
  ▼<wind>
    <speed value="3.1" unit="m/s" name="Light breeze"/>
    <gusts/>
    <direction value="230" code="SW" name="Southwest"/>
  </wind>
  <clouds value="90" name="overcast clouds"/>
  <visibility value="16093"/>
  <precipitation value="0.25" mode="rain" unit="1h"/>
  <weather number="500" value="light rain" icon="10d"/>
  <lastupdate value="2019-09-23T14:57:14"/>
</current>
```

XML

- Example: current weather in Cincinnati
  - Let's navigate through the XML file
    - Retrieving the value associated with the key "value" for the temperature node

```
temperature = root[1]
print(temperature.get("value"))
```


```
▼<current>
 ▼<city id="4508722" name="Cincinnati">
    <coord lon="-84.51" lat="39.1"/>
    <country>US</country>
    <timezone>-14400</timezone>
    <sun rise="2019-09-23T11:26:23" set="2019-09-23T23:34:36"/>
  </city>
  <temperature value="295.36" min="294.15" max="296.48" unit="kelvin"/>
  <humidity value="94" unit="%"/>
  <pressure value="1016" unit="hPa"/>
 ▼<wind>
    <speed value="3.1" unit="m/s" name="Light breeze"/>
    <gusts/>
    <direction value="230" code="SW" name="Southwest"/>
  </wind>
  <clouds value="90" name="overcast clouds"/>
  <visibility value="16093"/>
  <precipitation value="0.25" mode="rain" unit="1h"/>
  <weather number="500" value="light rain" icon="10d"/>
  <lastupdate value="2019-09-23T14:57:14"/>
</current>
```

XML

- Note that sometimes it is difficult to navigate through JSON and XML files
  - One must know *ex-ante* the relevant fields and tree structure to navigate through the hierarchical trees
  - There are modules that help with this for very popular APIs, such as Twitter

API

- Many APIs/web services offer free, but limited services
  - Frequent queries and more complex data often require payments
  - Example: https://openweathermap.org/price (as of 07/19/2021)

Free	Startup	Developer	Professional	Enterprise
	40 USD / month	180 USD / month	470 USD / month	2,000 USD / month
Get API key	Subscribe	Subscribe	Subscribe	Subscribe
60 calls/minute 1,000,000 calls/month	600 calls/minute 10,000,000 calls/month	3,000 calls/minute 100,000,000 calls/month	30,000 calls/minute 1,000,000,000 calls/month	200,000 calls/minute 5,000,000,000 calls/month
Current Weather Minute Forecast 1 hour* Hourly Forecast 2 days* Daily Forecast 7 days* National Weather Alerts* Historical weather 5 days* Climatic Forecast 30 days Bulk Download	Current Weather Minute Forecast 1 hour** Hourly Forecast 2 days** Daily Forecast 16 days National Weather Alerts** Historical weather 5 days** Climatic Forecast 30 days Bulk Download	Current Weather Minute Forecast 1 hour Hourly Forecast 4 days Daily Forecast 16 days National Weather Alerts Historical weather 5 days Climatic Forecast 30 days Bulk Download	Current Weather Minute Forecast 1 hour Hourly Forecast 4 days Daily Forecast 16 days National Weather Alerts Historical weather 5 days Climatic Forecast 30 days Bulk Download	Current Weather Minute forecast 1 hour Hourly Forecast 4 days Daily Forecast 16 days National Weather Alerts Historical weather 5 days Climatic Forecast 30 days Bulk Download
Basic weather maps Historical maps Global Precipitation Map	Basic weather maps Historical maps Global Precipitation Map	Advanced weather maps Historical maps Global Precipitation Map	Advanced weather maps Historical maps Global Precipitation Map	Advanced weather maps Historical maps Global Precipitation Map
Road Risk API	Road Risk API	Road Risk API	Road Risk API	Road Risk API
Air Pollution API Geocoding API	Air Pollution API Geocoding API	Air Pollution API Geocoding API	Air Pollution API Geocoding API	Air Pollution API Geocoding API
Weather widgets	Weather widgets	Weather widgets	Weather widgets	Weather widgets
Uptime 95%	Uptime 95%	Uptime 99.5%	Uptime 99.5%	Uptime 99.9%

API

- How do companies charge an API user?
  - How do they know the usage by a certain user?
    - Via mandatory API keys
    - Now, you know why I did not share my API keys with you ☺
- How to circumvent API limitations?
  - Request multiple free API keys
    - Ethical? Legal?
      - Unlikely for business applications
      - Likely for noncommercial applications (*e.g.*, academic studies)
  - Put the “code to sleep” to avoid asking for data too often

API

- Business implications
  - Selling access to databases is becoming an incredibly common revenue stream
    - Think about all the “free-to-use” social networks out there
  - Two major steps an organization must follow:
    - Implement (code) the interface (API)
      - Tools: RESTify, Swagger, Python Flask, …
    - Define access policies (price, constraints, etc.)
      - Tools: IBM API Management Service, …

CASE STUDY: NEWS API

News API

- Monitoring news media
  - What is the media saying about a certain person and/or organization?
  - Why/when can this be important?
    - Election?
    - Marketing campaigns?
    - Proxy to evaluate managerial actions?
    - Crisis?

News API

- We will collect news articles using News API
  - Some of the API limitations (as of 07/19/2021)

Developer	Business	Enterprise
$0	$449	$849
totally free, start now	per month, billed monthly	per month, billed monthly
For all development and testing of personal or commercial projects.	For production and published commercial projects.	For enterprise projects that require exceptional resources and support.
Search all articles and get live top headlines	Search all articles and get live top headlines	
New articles available with 1 hour delay	New articles available in **real-time**	
Search articles up to a month old	Search articles up to **3 years** old	
CORS enabled for localhost	CORS enabled for **all origins**	
100 requests per day	250,000 requests per month included	250,000 requests per month included
No extra requests available	$44.90 per 25,000 extra requests	$44.90 per 25,000 extra requests
No uptime SLA	No uptime SLA	**99.95% uptime SLA**
Basic support	**Email support**	**Premium email and phone support**
Start with Developer	Start with Business	Start with Enterprise

Source: https://newsapi.org/pricing

News API

- News API returns data in JSON
  - Documentation: https://newsapi.org/docs

- Example: returning latest news about Bitcoin

```
import requests
response = requests.get("https://newsapi.org/v2/everything?q=Bitcoin&apiKey= add-your_key_here")
json_data = response.json()
```

Homework #5

- News API allows one to collect the latest 100 online articles on any topic
  - But our query currently returns only 20 articles
  - Goal: write a script that collects *only* the ***description*** of the latest 100 articles about Bitcoin
    - The result after running your script should be a list of strings whose length is 100
    - Hint: read the API documentation and find a key that allows you to collect more than 20 articles https://newsapi.org/docs/endpoints/everything
  - Report your code on Canvas before the deadline

Summary

- We learned more about APIs
  - Responses in XML
- We have not learned how to create APIs
  - Beyond the scope of this course
- Next lecture: collecting data from document-oriented databases