Web Scraping

Part #3: XML and Semi-Structured Data

Where do data scientists get their data?

A data scientist needs sources for data to do his or her work. While you don't need this level of detail on the AP test, here are is some information about the types of data available to data scientists:

- **Unstructured data:** Most data available on the web is unstructured data. Image files, sound files, video files, text files, and HTML files are all examples of unstructured data. These are some of the richest sources of data available, but they are also difficult to process (search, sort, classify, analyze, summarize etc). For example, while you were easily able to write algorithms in a previous chapter to apply image filters to the pixels of an image, it is very difficult to write functions to classify the objects featured in image files (people, trees, dogs, cats, mountains, etc). Similarly, while a web browser can effectively display an HTML file, it is not easy to write an algorithm to give a short summary of the content of the webpage. You can certainly extract useful data from an HTML file, but you just saw that it takes a great deal of effort using a module such as **Beautiful Soup**. In fact, the name of the module describes the World Wide Web: it is a beautiful soup of unstructured data. Data scientists often spend a great deal of time finding their data. To read more, here's a great blog post on the topic: https://www.dataquest.io/blog/web-scraping-beautifulsoup/ (https://www.dataquest.io/blog/web-scraping-beautifulsoup/)

- **Semi-structured data:** You already have experience with semi-structured data: a *.csv* file is an example of semi-structured data. This is data that supports automated processing of its contents, such as we saw with Pandas during the our chapter on Open Data. In other courses, you may learn about JSON (https://www.w3schools.com/js/js_json_intro.asp), and in this notebook you will learn about XML (https://www.w3schools.com/xml/xml_whatis.asp). As you will soon see, the beauty of XML is that you can work with data in an automated way.

- **Structured data:** This is data that is stored in a *database*. Organizations such as corporations, governments, and universities will have servers dedicated to their databases and database software. The data stored in a database is similar to what you have seen in *.csv* files, but has some additional structure. We will not work with databases (structured data) in this class, but they are a great source of information. But if you want to learn more, you may want look up the term *relational database* or *SQL*.

What is XML?

XML stands for eXtensible Markup Language. While HTML is meant to display webpages, XML is meant to store/transport/describe data. Humans can read and understand XML, and computers can also process XML in an automated way.

Task #1: Read the following pages from W3Schools, then answer the questions below:

- XML Introduction: https://www.w3schools.com/xml/xml_whatis.asp (https://www.w3schools.com/xml/xml_whatis.asp)
- XML Tutorial: https://www.w3schools.com/xml/default.asp (https://www.w3schools.com/xml/default.asp)

```
**Question \#1:** What does the "extensible" in "extensible markup language" mean?

Your Answer: Extensible in this case ,means the ability to change and degrade over time.
```

```
**Question \#2:** Suppose you are trying to write an algorithm to process data in an automated way. Why would you prefer for your algorithm to work with *extensible* data?

Your Answer: Extenisble data has a defined structure that you can use as building blocks to get the info needed.
```

Chicago Weather Data

Before you read any further, visit the page http://w1.weather.gov/xml/current_obs/KORD.xml (http://w1.weather.gov/xml/current_obs/KORD.xml) and look at the data on the page. This is a feed of current weather conditions at Chicago's O'Hare airport. You can use this XML document to create a webpage or app that always knows the most up-to-date weather in Chicago.

Let's take a look at the XML source for this feed:

In [2]:
```
from bs4 import BeautifulSoup       # Import BeaurtifulSoup
from urllib.request import urlopen  # Import urlopen

xml_page = urlopen("http://w1.weather.gov/xml/current_obs/KORD.xml")   # Opens wh
bs_obj = BeautifulSoup(xml_page, 'xml')  # Extracts the xml data

print(bs_obj.prettify())  # Makes it more easily readible or 'pretty'
```

```
 60
</suggested_pickup_period>
<location>
 Chicago, Chicago-O'Hare International Airport, IL
</location>
<station_id>
 KORD
</station_id>
<latitude>
 41.97972
</latitude>
<longitude>
 -87.90444
</longitude>
<observation_time>
 Last Updated on May 5 2020, 6:51 am CDT
</observation_time>
<observation_time_rfc822>
 Tue, 05 May 2020 06:51:00 -0500
```

If you want to extract the current temperature from this data, just run the code below:

In [3]:
```
current_temp = bs_obj.find('temp_f').getText()  # Grabs the text found in the ins
print(current_temp)
```

```
45.0
```

```
**Question \#3:** What will you probably notice about the output from the cell
directly above if you were to re-run the code in this notebook at a later date?

Your Answer: It will change what the tempreture is at the given date/time.
```

Question #4: Where was this temperature measurement taken? To answer this question, you must write code that uses the latitude and longitude data in the XML above to create a tuple. Give your answer as a tuple in the form (latitude,longitude)

Location Tuple:

In [17]:
```
# Your code here
current_long = bs_obj.find("longitude").getText()
current_lat = bs_obj.find("latitude").getText()
current_point = current_long, current_lat
print(current_point)
```

```
('-87.90444', '41.97972')
```

In [20]:
```
ref_point = (-87.90444,41.97972)
print(ref_point)
```

```
(-87.90444, 41.97972)
```

Task #2 Use the Google Maps API to create a marker map with the location of the coordinates you found in the previous question.

- HINT: Look back at Metadata Part 4 to see how we displayed GPS coordinates

In [26]:
```
# Import the gmaps python module and load in your API Key:
import gmaps
gmaps.configure(api_key="AIzaSyCLla6Q7krE9xNg6SnNMoGNIzjCLddE9EU")

location = []

from ipywidgets.embed import embed_minimal_html


location.append(ref_point)


marker = gmaps.marker_layer(location)

markermaps = gmaps.Map()
markermaps.add_layer(marker)

embed_minimal_html('MarkerMap.html', views=[markermaps])
print("*** Check your 'Webscraping Part 4' folder to find the new HTML file named
```

```
*** Check your 'Webscraping Part 4' folder to find the new HTML file named "Mar
kerMap3". ***
```

Question \#5: What does the following function, *tag_extractor(url, tag)*, do? Some structure is provided below to help you answer this question:

Your Answer:

* _What is the purpose of the function?_
    * Your answer: To find the current tempreture
* _What purpose does the parameter **url** serve in the function?_
    * Your answer: which url should I use
* _What purpose does the parameter **tag** serve in the function?_
    * Your answer: Which variaabke should I read
* _What information is being returned by the function?_
    * Your answer: Current tempreture

In [6]:
```
def tag_extractor(url, tag):
    from bs4 import BeautifulSoup
    from urllib.request import urlopen

    xml_page = urlopen(url)   #opens whatever page we are requesting
    bs_obj = BeautifulSoup(xml_page, 'xml')

    return bs_obj.find(tag).getText()

tag_extractor('http://w1.weather.gov/xml/current_obs/KORD.xml', 'temp_f')
```

Out[6]:
```
'45.0'
```

Question \#6: How can *tag_extractor(url, tag)* be considered an abstraction that helps to manage the complexity of a computer program?

Your Answer: Instead of hveing to run the above over and over, you just use this function, plug in the url and tagt, and you can save many cell space.

Task #3: Use *tag_extractor(url, tag)* to determine the date/time of the most recent temperature measurement.

In [29]:
```
# Your code here
tag_extractor('http://w1.weather.gov/xml/current_obs/KORD.xml', 'observation_time
```

Out[29]:
```
'Tue, 05 May 2020 09:51:00 -0500'
```

HTML as Output

Question \#7: What is the purpose of the function *html_output()*? What kind of data does html_output() produce as output?

Your Answer: It takes the current tempreture at ohare, and displays it in a website

In [32]:
```
# define the function:
def html_output():
    output_string = """
    <html>
    <head>
        <style>
            body {
                background-color: #BBBBBB;
                text-align: center;
            }
        </style>
    </head>

    <body>
    <h1>Chicago Weather</h1>
    <p> The current temperature in Chicago is
    """

    output_string += tag_extractor('http://w1.weather.gov/xml/current_obs/KORD.xr

    output_string += """
    degrees Fahrenheit.
    </p>
    <br>

    </body>
    </html>
    """

    html_file= open("output/O'Hare Temperature.html","w")
    html_file.write(output_string)
    html_file.close()

# now call the function:

html_output()
print("*** Look in the 'Output' folder of 'Web Scraping Part 3' to find the new H
```

```
*** Look in the 'Output' folder of 'Web Scraping Part 3' to find the new HTML f
ile. ***
```

Task #4: Create your own version of *html_output()* that includes the date/time of the most recent temperature measurement as well as two other measurements to your output *.html* document.

In [39]:
```
# Your code here
# define the function:
def html_output():
    output_string = """
    <html>
    <head>
        <style>
            body {
                background-color: #BBBBBB;
                text-align: center;
            }
        </style>
    </head>

    <body>
    <h1>Chicago Weather</h1>
    <p> The last update of tempreture readings was on
    """

    output_string += tag_extractor('http://w1.weather.gov/xml/current_obs/KORD.xr

    output_string += """

    and it meassure the weather was

    """

    output_string += tag_extractor('http://w1.weather.gov/xml/current_obs/KORD.xr

    output_string +="""

    and the wind direction is

    """
    output_string += tag_extractor('http://w1.weather.gov/xml/current_obs/KORD.xr
    """
    </p>
    <br>

    </body>
    </html>
    """

    html_file= open("output/O'Hare Temperature.html","w")
    html_file.write(output_string)
    html_file.close()

# now call the function:

html_output()
print("*** Look in the 'Output' folder of 'Web Scraping Part 3' to find the new H
```

```
*** Look in the 'Output' folder of 'Web Scraping Part 3' to find the new HTML f
ile. ***
```

Task #5: This is a multistep task:

1) Read this w3schools documentation (https://www.w3schools.com/tags/att_meta_http_equiv.asp)

2) Add the HTML `<meta http-equiv="refresh" content="3600">` between the head tags in your html_output() function

3) Read (but do not run) the code in the cell below:

In [ ]:
```
running = False
import time


while running:
    html_output()
    time.sleep(3600)
```

Question \#9: If you were to change *running* to *True* in the code cell above, then running this cell would start an infinite loop. What would the purpose be of an infinite loop and the HTML above, `<meta http-equiv="refresh" content="3600">`? What would it allow you to do?

Your Answer: It would allow you to get cconstant feedback and fresh results

Experiment and Explore

Task #6: With any extra time this period, go to http://w1.weather.gov/xml/current_obs/KORD.xml (http://w1.weather.gov/xml/current_obs/KORD.xml), but change the **KORD** portion of this URL to another ICAO airport code for an airport in the United States or its territories. Experiment. Show the results of your experimentation in an HTML output file, and produce a Google Maps Marker Map in this notebook that includes the location you chose to explore.

- Airport codes in the lower-48 states: https://en.wikipedia.org/wiki/List_of_airports_by_ICAO_code:_K (https://en.wikipedia.org/wiki/List_of_airports_by_ICAO_code:_K)
- All airport codes in the United States and territories: https://en.wikipedia.org/wiki/List_of_airports_in_the_United_States#Lists_by_ICAO_location_inc (https://en.wikipedia.org/wiki/List_of_airports_in_the_United_States#Lists_by_ICAO_location_in

In [40]:
```
# Your code here
from bs4 import BeautifulSoup       # Import BeaurtifulSoup
from urllib.request import urlopen  # Import urlopen

xml_page = urlopen("https://w1.weather.gov/xml/current_obs/KAAA.xml")   # Opens u
bs_obj = BeautifulSoup(xml_page, 'xml')  # Extracts the xml data

print(bs_obj.prettify())
```

```
 </longitude>
 <observation_time>
  Last Updated on May 5 2020, 9:55 am CDT
 </observation_time>
 <observation_time_rfc822>
  Tue, 05 May 2020 09:55:00 -0500
 </observation_time_rfc822>
 <weather>
  Heavy Drizzle
 </weather>
 <temperature_string>
  49.0 F (9.5 C)
 </temperature_string>
 <temp_f>
  49.0
 </temp_f>
 <temp_c>
  9.5
 </temp_c>
 <relative_humidity>
```

In [43]:
```
def html_output():
    output_string = """
    <html>
    <head>
        <style>
            body {
                background-color: #BBBBBB;
                text-align: center;
            }
        </style>
    </head>

    <body>
    <h1>Chicago Weather</h1>
    <p> The current temperature in Logan County is
    """

    output_string += tag_extractor('https://w1.weather.gov/xml/current_obs/KAAA.x

    output_string += """
    degrees Fahrenheit.
    </p>
    <br>

    </body>
    </html>
    """

    html_file= open("output/Logan Temperature.html","w")
    html_file.write(output_string)
    html_file.close()

# now call the function:

html_output()
print("*** Look in the 'Output' folder of 'Web Scraping Part 3' to find the new H
```

```
*** Look in the 'Output' folder of 'Web Scraping Part 3' to find the new HTML f
ile. ***
```

In [44]:
```
current_long = bs_obj.find("longitude").getText()
current_lat = bs_obj.find("latitude").getText()
current_point = current_long, current_lat
print(current_point)
```

```
('-89.33891', '40.15885')
```

In [46]:
```
refs_point = -89.33891, 40.15885
```

In [48]:
```
import gmaps
gmaps.configure(api_key="AIzaSyCLla6Q7krE9xNg6SnNMoGNIzjCLddE9EU")

location = []

from ipywidgets.embed import embed_minimal_html


location.append(refs_point)


marker = gmaps.marker_layer(location)

markermaps = gmaps.Map()
markermaps.add_layer(marker)

embed_minimal_html('MarkerMap2.html', views=[markermaps])
print("*** Check your 'Webscraping Part 4' folder to find the new HTML file named
```

```
*** Check your 'Webscraping Part 4' folder to find the new HTML file named "Mar
kerMap2". ***
```