Text Data

LECTURE 4

Organization of Lecture 4

- Processing HTML Files
- Handling CSV Files
- Reading JSON Files
- Processing Text in Natural Languages

Origins of Raw Data

- Structured docs (HTML, XML, CSV, and JSON files)
- Unstructured docs (plain, human-readable text)
  - Hardest to process and where programming skills necessary

- All these are readable by human
- Can open any in a simple text editor (eg. NotePad)
- Anaconda has modules for text analysis
  - – BeautifulSoup, csv, json, nltk

Processing HTML Files

- HTML – markup language used on web for human-readable representation of information
- HTML document
  - Consists of text and predefined tags in < >'s
  - Control presentation and interpretation of text
  - Tags may have attributes
  - Table 3 shows some tags
    - <HTML>, <HEAD>, <TITLE>, <BODY>, <H1>, <i>, <b>, , <br>, <img>, <table> ,

A Tiny HTML Document

html5temp.html

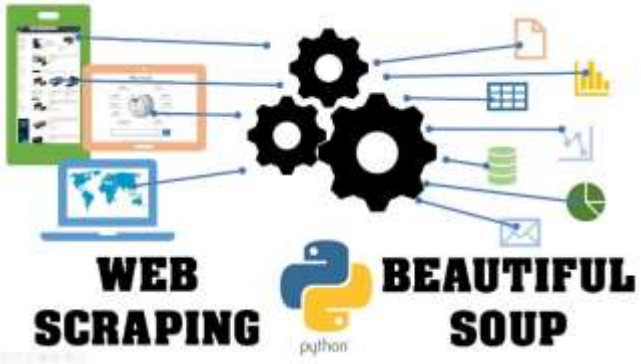
```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title> A Tiny HTML Document </title>
<link href ="styles.css" rel="stylesheet">
<script src="scripts.js"></script>
</head>

<body>
<p>Let's rock the browser, HTML5 style.</p>
</body>
</html>
```

Meaning of text is available in Alt Text box.

XML

- HTML precursor to XML
- XML ≠ HTML
- XML – family of markup languages with similar structure
- XML tags are application-specific
  - Any alphanumeric string can be a tags
  - Must follow rules
  - Control *interpretation* of text only
  - Not intended for human reading
  - (eXtensible Stylesheet Language Transformation (XSLT) can transform XML to HTML and to CSS
  - Cascading Style Sheets add styles to HTML documents

BeautifulSoup Module



- Used for parsing, accessing, and modifying HTML and XML documents
- Can construct BeautifulSoup object from a markup string, markup file, or URL of markup document

```
from bs4 import BeautifulSoup

#construct soup from a String
soup1 =BeautifulSoup("<HTML><HEAD><<headers>></HEAD><<body>></HTML>")
print (soup1)
```

Console Shell

```
<html><head> </head><body>&lt;<headers>&gt; &lt;&gt;</headers></body></html>
>
```

BeautifulSoup Ex. 2

```
from bs4 import BeautifulSoup
#construct soup from a local File
soup2 =
BeautifulSoup(open("myDoc.html"),features="html5lib")
print (soup2)
```

Console Shell

```
<html lang="en"><head>
    <meta charset="utf-8"/>
    <title> A Tiny HTML Document </title>
    <link href="styles.css" rel='stylesheet"'/>
    <script src="scripts.js"></script>
  </head>

  <body>
    <p> Let's rock the browser, HTML5 style. </p>


</body></html>
>
```

BeautifulSoup Ex. 3

```
from bs4 import BeautifulSoup
from urllib.request import urlopen

#construct souop from a web document
#remember urlopen() does not add "http://"!
soup3 =
BeautifulSoup(urlopen(http://www.networksciencelab.com/”),features=“html5lib”)
print (soup3)
```

- Note second optional parameter:
  - this is the markup parser and the default is **=“html”**

Markup Parsers

- `html`– default, very fast, not lenient; used for simple HTML
- `lxml` – (very fast, lenient)
- `Xml` – for XML files only
- `html5lib` – very slow, extremely lenient; used for HTML docs with complicated structure or is speed is not issue

More `BeautifulSoup` functions

- print (soup.prettify()) – indents (fig 1)
- soup.get_text() returns text part of markup doc (fig 2)

Console Shell

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   A Tiny HTML Document
  </title>
  <link href="styles.css" rel='stylesheet"'/>
  <script src="scripts.js">
  </script>
 </head>
 <body>
  <p>
   Let's rock the browser, HTML5 style.
  </p>
 </body>
</html>
>
```

Figure 1

Console Shell

```
A Tiny HTML Document

Let's rock the browser, HTML5 style.

>
```

Figure 2

BeautifulSoup tag functions

- Markup tags locate file fragments

```
print (soup2.title)
```

Console Shell

```
<title> A Tiny HTML Document </title>
>
```

- Any tag has a name, value, parent, next, prev tags, and children attributes

```
print (soup2.title.name)
```

Console Shell

```
title
>
```

BeautifulSoup tag functions

- Finds the first instance of a tag

```
print (soup2.find("title"))
```

Console Shell

```
<title> A Tiny HTML Document </title>
>
```

- Finds all the instances of a certain tag

```
links = soup3.find_all("a")        #returns list
if links[0].has_attr("href"):      #check if present
  print (links[0].get("href"))
```

Console Shell

```
https://pragprog.com/book/dzpyds/data-science-essentials-in-pyth
>
```

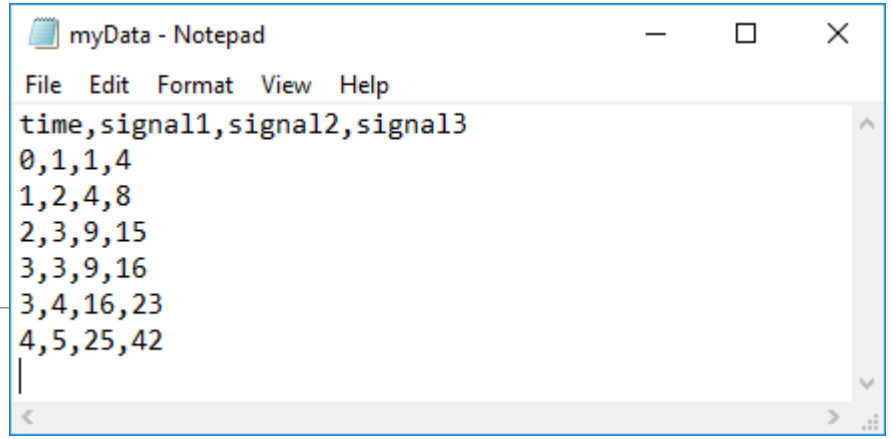
BeautifulSoup and list comprehension

- To get all links and respective URLs and labels (useful for web crawling)

```
from bs4 import BeautifulSoup
from urllib.request import urlopen

#construct soup from a file
with urlopen("http://www.networksciencelab.com") as doc:
  soup = BeautifulSoup(doc, features="html5lib")

links = [(link.string,link["href"])
  for link in soup.find_all("a")
  link.has_attr("href")]
print(links[5])
```

```
('Network Science Workshop', 'http://www.slideshare.net/DmitryZinoviev/workshop-20212296')
>
```

CSV Files



- Structured file format for (nearly) tabular data.
- Choice format for lots of spreadsheet software
  - Excel, OpenOffice, data.gov (12,550+ data sites)
- Consist of columns (variables) and rows(records)
- Fields in record separated by commas (or other delimiters)
  - What looks like a delimiter might not be data, “Hello, World”, data

Opening & Reading `csv` files

```
with open ("somfile.csv", newline = ' ') as infile:
  reader = csv.reader (infile, delimiter = ',', quotechar = '"')
```

- Delimiter and quotechar are optional parameters & there are others
- First record in a `csv` file contains the column headers and is usually treated differently
- `reader` provides an iterator interface for use in a for each loop
  - Returns next record as a list of string fields
  - does NOT do any conversions or stripping unless pass optional parameter `skipinitialspace = True`
  - If file is not known and potentially very large, then read/process one at a time

Writing **csv** files

- **csv writer** provides functions for writing
  - Functions **writerow()** – writes a sequence of strings or numbers as one record
  - Function **writerows()** – writes a list of sequences
  - Everything written as strings (so no conversion necessary)

```
import csv

with open('names.csv', 'w', newline='') as csvfile:
    fieldnames = ['first_name', 'last_name', 'result']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerow({'first_name': 'John',
                     'last_name': 'Smith',
                     'result' : 54})
    writer.writerow({'first_name': 'Jane',
                     'last_name': 'Lewis',
                     'result': 63})
    writer.writerow({'first_name': 'Chris',
                     'last_name': 'Davies',
                     'result' : 72})
```

Example to extract “Answer.Age” column

```
import csv
import statistics

#open file and read data
with open("CovidDeathCounts.csv") as infile:
data = list(csv.reader(infile))

#examine header row to find column of interest
ageIndex = data[0].index("Age")

#access field of interest in records and calculate/display stats
ages = [int(row[ageIndex]) for row in data[1:]]
print ("mean\tstdev")
print(statistics.mean(ages), "\t", “{:.2f}”.format(statistics.stdev(ages)))
```

Console Shell

```
mean    stdev
42.5     24.90

>
```

`CSV` and `statistics` vs `pandas`

- Modules `CSV` and `statistics` are low-end, “quick and dirty” tools
- Ch. 6: `pandas` data frames go beyond trivial exploration of a few columns

JSON files

- Lightweight data interchange format
- Language-independent, but more restricted in data representation then pickle
- Many popular websites (Twitter, Facebook, etc) provide APIs that use JSON

JSON data types

- Atomic types – strings, numbers, true, false, null
- Arrays – corresponds to list, uses []s; elements do not have to be same type `[1, 3.14, “a string”, true, null]`
- Objects – corresponds to dictionary, uses {}s

```
{“age” :37, “gender” : “male”, “married” : true}
```

- Any recursive (nested) combinations
- Sets and complex numbers cannot be stored in JSON file, so must convert before exporting to JSON.
  - Store a complex numbers as array of two doubles

Serialization/Deserialization in `JSON`

- Functions in module JSON
  - `dump`() exports ONE Python object to text file
  - `dumps`() exports ONE Python object to a text string
  - `load`() converts contents of a text file into one Python object
  - `loads`() converts a valid JSON string into a Python object
- Note: If an existing file contains more than one object, read it as text, convert to array of objects, and use loads() to deserialize the text
- Note: When you save to `JSON` file, value of your variables are saved;

  When you load, values become *independent – they do not reference the same variable. Hence the value of pickling.*

Code Fragment Sample

```
import json
object1 = {"age" :37, "gender" : "male", "married" : True}
#save an object to a file
with open ("data.json", "w") as out_json:
  json.dump(object1, out_json, indent=None, sort_keys = False)
#load an object from a json file
with open ("data.json") as in_json:
  object2= json.load(in_json)
print ("object2 loaded is ", object2)
#serialize an object to a string
json_string = json.dumps (object2)
#parse a string as JSON
object3 = json.loads(json_string)
print ("object3 is ", object3)
```

```
object2 loaded is  {'age': 37, 'gend
'male', 'married': True}
object3 is  {'age': 37, 'gender': 'male',
 'married': True}
>
```