Scraping Data From Webpages

Develop code to extract a table
from an HTML-coded webpage

A Project

- An online news article included a table of congressional members, and contributions to each from the Comcast Corporation.
- The goal is to extract the table from the HTML markup, and save it into a spreadsheet-compatible format.
- This scripting exercise practices Python programming techniques to extract the data.
  - A straightforward copy-and-paste also works, but as the amount of data increases that becomes less convenient

The Task:

- Read contents of a webpage as one big text block
  - HTML can be spread across lines arbitrarily
- Search for a specific pattern
  - Must inspect webpage for suitable textstrings
- Extract desired fields into a Python data structure
  - list or dictionary
- Sort data by how much money went to each state

Some Python Modules

- re
  - Brings in support for *regular expressions*
- csv
  - Read and write spreadsheet-compatible .csv "comma-separated-values" files
- matplotlib
  - Provides object-oriented data-plotting facilities
  - Sub-modules provide specific features
  - Integrated with Jupyter
  - Works well with...
- numpy
  - Array-oriented mathematical operations

(Review) File Reading

- Open a file for reading and use it:
  - Open the file, make a *filehandle*
  - Use the filehandle
  - Close the filehandle

```
filehandle = open('filename', 'r')
alldata = filehandle.read()  # or other ops
filehandle.close()
```

- Or, create an "opened-file" block:
  -

```
with open('filename', 'r') as filehandle:
   alldata = filehandle.read() # or other ops
```

Read the Webpage's Contents All at Once

- Open the file, read it, and close it again:

`handle = open(webpage_file, 'r')`

  - » Generates a "handle" for accessing the file

`contents = handle.read()`

  - » Read entire contents from the handle, into a variable, with one big "read()" call

`handle.close()`

  - » Close webpage file (and set it aside mentally)

...OR...

- Use the "with" statement to limit the scope of the "open()" and do the "close()" automatically

```
with open(webpage_file, 'r') as handle:
    contents = handle.read()
```

(Reading the Webpage's Contents One Line At a Time)

- As before, open and close the file in a "with" block:

```
with open(webpage_file, 'r') as handle:
    …
```

...or explicitly:

```
handle = open(webpage_file, 'r')
    …
handle(close)
```

- Get the response contents as a list of lines:

```
lines = []
for l in handle.readlines():
    lines.append(l)
```

...or as a *list comprehension*:

```
lines = [ l for l in handle.readlines() ]
```

Practice:

- Download:
  - montcs.bloomu.edu/215/**Datasets/webpages/**
    - Download one or more of the pages
- Read file into one big variable
  - Ask user for filename (or get from command line)
  - Open file
  - Read contents
- Re-read file, into a list of lines
  - Rewind (or close and re-open) the file
  - Loop-and-append or use a list comprehension

Desired Data Within the Webpage

- Data in tabular format
  - Each row contains 4 items ("cells")
  - Rows marked by "<tr …>" and "</tr>" tags
  - Items/cells marked by "<td …>" and "</td>", or by "<th …>" and "</th>"
  - Items separated by optional whitespace, possibly including newlines
- Arbitrary data within each cell
  - But doesn't include the "<" character, which starts an HTML tag

Ways to Search For Desired Text

- String methods `.find()`, `.index()`
  - locate a constant or variable substring within a source string
  - return the position within the source string
- Regular expression matching
  - regex defines a *pattern* that matches the desired text

`re.search()`
  - » find a matching substring within the source

`re.match()`
  - » regex must match entire source string

`re.findall()`
  - » find all occurrences of the pattern within the source

Text to Search For

- Strings of 4 table-data or table-header elements
  - Possibly separated by whitespace
- HTML table-data elements:
  - `<td>stuff… </td>`
    - We don't care about the "`td`", but we want the "`stuff…`"
- HTML table-header elements:
  - `<th>stuff… </th>`
    - We don't care about the "`th`", but we want the "`stuff…`"
- We want to save "`stuff…`" into a variable

A Suitable Regular Expression:

- Pieces:

```
open_tag = '<t[dh][^>]*>'
```

  - Starts with '<t', includes either 'd' or 'h', optionally followed by non-'>' text, ends with '>'

```
contents = '([^<>]+)'
```

  - At least one of any text *except* '<' or '>'; in a group

```
close_tag = '</t[dh]>'
```

  - Either '</td>' or '</th>'

```
whitespace = '\s*'
```

  - Optional whitespace ('\s') characters, including newlines

- Complete regex:

```
regex = 4 * \
(open_tag + contents + close_tag + whitespace)
```

Sorting a List of Lists

A data list:

```
lst = [(1,2,3), (2,1,4), (0,3,2), (9,9,9)]
```

- Sort lst according to *last* value in each triple:

```
def sortftn(k): # operate on an element-to-be-sorted
    return k[-1]

slst = sorted(lst, key=sortftn)   # use sortftn
```

- Display the sorted list:

```
print(slst)
```

```
[(0,3,2), (1,2,3), (2,1,4), (9,9,9)]
```

done