DEPAUL UNIVERSITY
COLLEGE OF COMPUTING AND DIGITAL MEDIA

CSC 243 NOTES
Fall 2020: Amber Settle

Monday, November 16, 2020

Announcements

- Questions about the eighth assignment or seventh quiz?
- The ninth assignment is due Thursday, November 19th – questions?
- The final exam is Monday, November 23rd – questions?

Parsing HTML files

The goal of this portion of the class is to learn to mine web pages for information.

To do that we need tools to automatically parse HTML web pages. The Python module **html.parser** provides classes that make it easy to parse HTML files.

The class **HTMLParser** from this module parses HTML files as follows:

1. The HTMLParser class is instantiated without arguments.
2. An HTMLParser instance is fed HTML data and calls handler functions when tags begin and end.

Some of the **handler methods** we can override include:

- **handle_starttag(tag, attrs)**: Start tag handler
- **handle_endtag(tag)**: End tag handler
- **handle_data(data)**: Arbitrary text data handler

For example, **handle_starttag**() would be invoked when an opening tag (i.e. something of the form **<tag attrs>** is encountered.

Note that the attributes are contained in a list (attrs) where each attribute in the list is represented by a tuple storing the name and value of the attribute.

handle_endtag() would be invoked when a closing tag (i.e. **</tag>**) is encountered.

For an example see the **htmlParser.py** file.

In that file a subclass of the HTMLParser class is created that **overrides the handle_starttag() and handle_endtag() methods** so that they print information about the fact that the tag was encountered.

Exercises

To understand how to override the functions of the HTMLParser class and how to make progress toward gathering information using a parser, let's work through a series of exercises.

Exercise 2: Write a parser class **PrettyParser** that prints the names of the start and end tags in the order that they appear in the document, and with an indentation that is proportional to the element's depth in the tree structure of the document. The class should ignore HTML elements that do not require an end tag, such as p and br.

For example, it would work as follows:

```
>>>
testParser('http://facweb.cdm.depaul.edu/asettle/csc2
43/web/test.html')
html start
    head start
        title start
        title end
    head end
    body start
        h2 start
        h2 end
        h1 start
        h1 end
    body end
html end
```

Exercise 3: Write an HTML parser **DataCollector** that collects all of the information found in the data portions of the file into a string. Create a method of the class called **getData**() that returns the string collected. Make sure that the constructor correctly sets up the class variable that will hold the data.

The following is an example of how the class would be used.

```
>>>
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/test1.html')
'\r\n\r\n  Test page\r\n\r\n\r\n Hello World
\r\n Read the news \r\n  Link to
test3.html\r\n\r\nAmsterdam\r\nIstanbul\r\nChic
ago  \r\nMail a Canadian professor\r\n\r\n\r\n'
>>> s =
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/test2.html')
>>> s
'\r\n\r\n  Test 2 page\r\n\r\n\r\n This is test
2\r\n Click here to e-mail
Amber\r\n\r\nAmsterdam\r\nIstanbul\r\n\r\nGo to
test 4!\r\n\r\n\r\n'
```

```
>>>
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/test3.html')
'\r\n\r\n  Test 3 page\r\n\r\n\r\n This is test
3\r\n Send mail to
nobody.\r\n\r\nAmsterdam\r\nIstanbul Istanbul
Istanbul\r\n\r\nGo to test 4!\r\nOr visit the
University of Chicago!\r\n\r\n\r\n\r\n\r\n\r\n'
>>>
```

Exercise 4: Write an HTML parser **HeaderParser** that finds and collects all of the headings in an HTML file fed into it. The parser works by identifying when a header tag has been encountered and setting a Boolean variable in the class to indicate that. When the data hander for the class is called and the Boolean in the class indicates that a header is currently open, the data inside the header is added to a list. When a closing header tag is encountered the Boolean variable is unset.

The following is an example of how the class would be used.

```
>>> lst =
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/test1.html')
>>> lst
['Hello World']
>>> lst =
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/headings.html')
>>> lst
['Heading One', 'Small Heading', 'Third
Heading']
>>> lst =
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/noheadings.html')
>>> lst
[]
>>>
```

Exercise 5: Write an HTML parser **ListParser** that collects the contents of all the list items, both ordered and unordered, in an HTML file fed into it. The parser works by identifying and remembering, via a class variable, when a list item tag has been encountered. When the data handler for the class is called and the class variable indicates that a list item is currently open, the data in the list item is added to a list in the class. When the list item tag is closed, the class adjusts the internal variable to register this.

The following is an example of how the class would be used.

```
>>> lst =
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/list1.html')
>>> lst
['Item 1', 'Item 2', 'Item A', 'Item B', 'Item
B1', 'Item B2', 'Item B3', 'Item C', 'X', 'Y']
```

```
>>> lst =
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/list2.html')
>>> lst
['Cat', 'Dog', 'Hermit crab', 'Java', 'C++',
'Lisp', 'Scheme', 'Python', 'English',
'German', 'Finnish', 'Spanish', 'Work days',
'Monday', 'Montag', 'maanantai', 'Weekend',
'Saturday', 'Samstag', 'lauantai']
>>> lst =
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/test1.html')
>>> lst
[]
>>>
```

Find the solutions in the file **moreHTMLparsers.py**.

urllib.parse module

In a previous exercise, **all of the anchor tags were printed**. But what if we just want the ones that correspond to URLs?

We can't just look for the ones that begin with http. Why? What would we miss if we did that?

The problem is if we look for anchor tags with href as an attribute and display what we get from that, we will lose information. Why?

What we need is a way to **construct an absolute URL from a relative URL**.

The module urllib.parse provides a few methods that operate on URLs, including one that does what we want: **urljoin**().

The following shows how the urljoin() method is used to construct an absolute URL from a relative URL:

```
>>> from urllib.parse import urljoin
>>> url =
'http://facweb.cdm.depaul.edu/asettle/csc243/web/test
2.html'
>>> relative = 'test4.html'
>>> urljoin(url, relative)
'http://facweb.cdm.depaul.edu/asettle/csc243/web/test
4.html'
```

Exercise: Using urljoin() create another version of the **LinksParser** class called **Collector**.

It will collect only HTTP URLs and, instead of printing them, will **collect them into a list**. The URLs will be in their **absolute** format.

The class will store the URL upon which it was called to generate the collection of URLs. And it will also support a method **getLinks**() that will return the list of URLs collected by the class.

For example, it would work as follows:

```
>>>
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/test1.html')
['http://www.cnn.com',
'http://facweb.cdm.depaul.edu/asettle/csc243/we
b/test3.html']
>>>
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/test2.html')
['http://facweb.cdm.depaul.edu/asettle/csc242/w
eb/test4.html']
>>>
testParser('http://facweb.cdm.depaul.edu/asettl
e/csc243/web/test3.html')
['http://facweb.cdm.depaul.edu/asettle/csc243/w
eb/test4.html', 'http://www.uchicago.edu']
>>>
```

See the solution in the file **moreHTMLparsers.py**.

Web crawler

We conclude this chapter by considering a basic web crawler, which is a program that systematically visits web pages by following hyperlinks.

Version 1

The first approach to the web crawler will use two methods to crawl through web pages:

1. **crawl1**() takes a URL as a parameter. It analyzes the web page at the specified URL to get the URLs of all linked pages and then recursively calls itself on the URLs of neighboring pages
2. **analyze**() takes a URL as a parameter. It opens the web page at that URL and uses a Collector object to collect the URLs of the pages linked to it.

Consider the code in the file **firstcrawler.py**.

Note that crawl() does not have a base case. This means that the function will continue to **crawl through web pages indefinitely**, which may be completely appropriate depending on what we want to do.

The exercises in the book consider limiting this in various ways.

Let's **test the first version of the crawler** on a set of web pages.

The web pages we will use were constructed solely for this purpose, and look like the following:

1. Page One → Page Two, Page Three
2. Page Two → Page Four
3. Page Three → Page Four
4. Page Four → Page Five
5. Page Five → Page One, Page Two, Page Four

What happens when we run the crawler? Why?

Version 2

A fix to the problem is to make sure that we don't visit a web page more than once.

Exercise, part (a): To do that we need a variable that keeps track of the web pages we have already visited.

Exercise, part (b): The crawl() method needs to be changed so that it only visits a web page if the link has not been visited before.

See the solution in **crawler.py**.

When we test this version of the crawler on the sample pages we don't have the same issues we did before:

```
>>> crawler = Crawler()
>>>
crawler.crawl('http://facweb.cdm.depaul.edu/asettle/csc243/
web/one.html')
Visiting
http://facweb.cdm.depaul.edu/asettle/csc243/web/one.html
Visiting
http://facweb.cdm.depaul.edu/asettle/csc243/web/three.html
Visiting
http://facweb.cdm.depaul.edu/asettle/csc243/web/four.html
Visiting
http://facweb.cdm.depaul.edu/asettle/csc243/web/five.html
Visiting
http://facweb.cdm.depaul.edu/asettle/csc243/web/two.html
```

Limiting the crawl

There are several ways you can limit the reach of our web crawler:

1. Only visit pages on the same web server
2. Only visit a fixed number of pages
3. Only visit a fixed number of links out from the starting page

To implement the first option, we need some additional information about parsing HTML pages.

Parsing HTML files

There is another useful method in the urllib.parse module called **urlparse**().

It allows you to obtain part of a URL, for example, the path, server, or protocol.

```
>>> url =
'http://facweb.cdm.depaul.edu/asettle/csc243/web/test2.html
'
>>> from urllib.parse import urlparse
>>> o = urlparse(url)
>>> o.path
'/asettle/csc243/web/test2.html'
>>> o.netloc
'facweb.cdm.depaul.edu'
>>> o.scheme
'http'
```