Chapter 4: Web Scraping and Data Collection

In this chapter, you'll first learn the basics of webscraping, which is an effective and powerful tool to collect business data online.

We'll discuss different types of Hypertext Markup Language (HTML) tags and how they construct webpages. You'll learn how to use the *Beautiful Soup* library to parse HTML files and extract information you need. We'll use the EKU finance faculty webpage as an example and extract the name, phone number, and email address of all faculty members.

Armed with this technique, you'll learn how to parse the source file of online podcasts and locate the mp3 file corresponding to a podcast. Then you can use the *webbrowser* library to play the online mp3 file.

Before you start, you'll need to install the latest version of the *Beautiful Soup* library, *bs4*. With your virtual environment activated, enter this:

```
pip install bs4
```

Follow the instructions all the way through.

A Primer on Webscraping

You can use the *Beautiful Soup* library to extract information from various websites. In this section, you'll learn the basics of webscraping.

We'll first discuss how HTML programming works, definitions of different types of HTML tags, and how various types of tags form different blocks on a website. We'll then discuss the *Beautiful Soup* library, and how to use the library to extract information from websites by parsing the source code.

What Is HTML?

HTML stands for Hypertext Markup Language. It's the standard markup language for webpage displays. It's the programming language that tells the browser how to construct and display the contents of webpages. HTML uses various types of *tags* to build blocks of webpages.

We'll first discuss how HTML tags work, and then show you a simple example of how different tags construct a webpage.

Anatomy of An HTML Tag

HTML tags tell the browser how to display the webpage content. Table 4-1 lists some of the commonly used tags and their main functions:

INSERT TABLE 4-1 HERE

Table 4-1: Commonly Used HTML Tags

Tag Name	Description
<html> tag	It is the root level tag of an HTML document. It encapsulates all other HTML tags.
<head> tag	It is the head section of an HTML document.
<title> tag	It specifies the title of the webpage, to be displayed on the frame of browser.
<body> tag	It contains the body of an HTML document.
<h1> tag	It is a level 1 heading, for example, the title of a news article.
tag.	It is a paragraph in the content of the webpage.
tag	It is a container for other page elements to divide the HTML document into sections.
<a> tag	It is a hyperlink to link one page to another.
tag	It defines a list item.

All tags start with an opening tag and a closing tag, so that the bowser can separate out different tags. For example, <a> tags start with <a> and close with , tags start with and close with , and so on.

NOTE: For a complete list of all HTML tags and their uses, go to the following site: https://eastmanreference.com/complete-list-of-html-tags

Let's use the <a> tag as an example to examine different components of HTML tags. Below is an example of an <a> tag:

```
<a class="redtext" href="http://libraries.uky.edu">Libraries</a>
```

In the above example, there are optional attributes in the opening tag: `<a class="redtext" href="http://libraries.uky.edu">`. The tag tells the browser that the `class` attribute is `redtext` and the `href` attribute is `http://libraries.uky.edu`. The content of the tag is between the opening and closing tags. In this example, the content is `Libraries`.

From HTML Tags to Webpages

HTML files have extensions with .html or .htm. To understand how the HTML language uses various tags to construct a webpage, let's look at an extremely simplified example with bare-bone elements needed.

Open the file *UKYexample.html* that I put on Canvas, as shown in Listing 4-1, using a text editor such as Notepad.

```
❶ <html>
❷   <head>
    <title>Example: University of Kentucky</title>
      <style>
      .redtext {
        color: red;
      }
      .leftmargin {
        margin-left: 10px;
      }
      </style>
  </head>
❸   <body>
    <p>Below are some links:</p>
❹     <p><a class="redtext" href="http://libraries.uky.edu">University of
Kentucky Libraries</a></p>
❺     <p><a class="web" href="http://www.uky.edu/directory/">University of
Kentucky Directory</a></p>
  </body>
</html>
```

Listing 4-1: HTML code for a simple webpage

Before we explain the code, let's see how the actual webpage looks like.

Go to the folder that you have saved the file *UKYexample.html*, and open it with your preferred web browser (right click, then select Open As). I use Google Chrome, and the webpage comes out as in Figure 4-1 below.

INSERT FIGURE 4-1 HERE

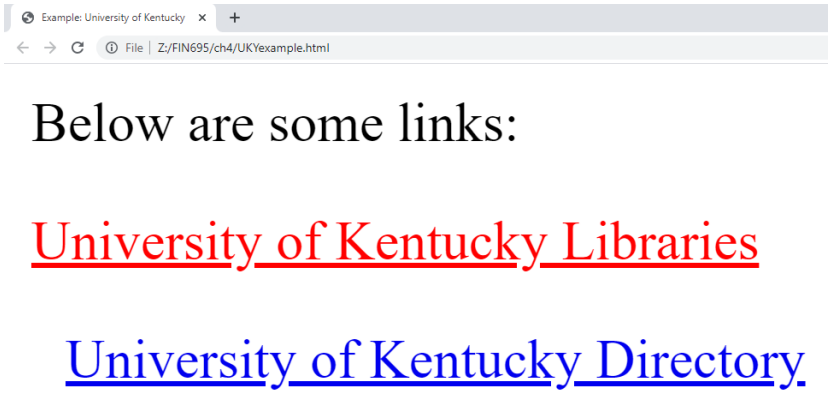

Figure 4-1: A simple webpage

Now let's link the HTML code to what's displayed on the above webpage.

At ❶, we start an opening `<html>` tag to contain all the code in the program. At ❷, we have a `<title>` tag nested in a `<head>` tag. The content of the `<title>` tag is "Example: University of Kentucky," hence the title of the webpage you see at the top left corner in Figure 4-1. We also use two `<style>` tags to define two classes `redtext` and `leftmargin` that we'll use later. I am using CSS style sheets to define the two classes. This is beyond the scope of this class, but you can have a look here https://www.w3schools.com/css/css_howto.asp if you are interested.

At ❸, we have a `<p>` tag nested in a `<body>` tag. The content of the `<p>` tag is "below are some links:" which you can see from the first line in the webpage as shown in Figure 4-1.

The first hyperlink is provided by an `<a>` tag nested in a `<p>` tag ❹. If you click on the link, it will bring you to the University of Kentucky Libraries. The second hyperlink is also an `<a>` tag nested in a `<p>` tag ❺. If you click on it, you'll be directed to the University of Kentucky Directory. Note that since the first `<a>` tag has class attribute `redtext`, you see red text in the

second line in Figure 4-1. Similarly, since the second `<a>` tag has class attribute `leftmargin`, you see a left margin in the third line in Figure 4-1.

Use Beautiful Soup to Extract Information from HTML

Now that you know how different tags in HTML work, you'll use the *Beautiful Soup* library to parse the HTML code and extract the information you want.

We'll first discuss how to parse a locally saved HTML file. After that, you'll learn how to extract information from a live webpage.

Parse Local HTML Files

Let's revisit the simple example *UKYexample.html* that's saved on your computer. Suppose you want to extract web address as well as the content in the `<a>` tag that provides the hyperlink to the University of Kentucky Directory. You can use the following program *ParseLocal.py* to accomplish the task. The code is provided in Listing 4-2.

```
#import the Beautiful Soup library
from bs4 import BeautifulSoup
#open the local HTML file as a text file
textfile = open("UKYexample.html", encoding='utf8')  #❶
#use the findAll() function to locate all <p> tags
soup = BeautifulSoup(textfile, "html.parser")
ptags = soup.findAll("p")   #❷
#print out the third <p> tag
print(ptags[2])
#find the <a> tag nested in the third <p> tag
atag = ptags[2].find("a")   #❸
print(atag)
#print the web address of the hyperlink
print(atag['href'])   #❹
#print the content of the <a> tag
print(atag.text)   #❺
```

Listing 4-2: Python code to parse a local HTML file

We first import the `BeautifulSoup()` function from the *bs4* library, the latest version of *Beautiful Soup*. At ❶, we open the local HTML file as a text file using the built-in Python

function `open()`. We then use the `findAll()` function to locate all `<p>` tags in the HTML file, and put them in the list *ptags* ❷.

NOTE: You need to put the file UKYexample.html *in the same folder as the program* ParseLocal.py. *If not, you must specify the path of the file* UKYexample.html *in line* ❶ *in Listing 4-2.*

There are three `<p>` tags in the list *ptags*, and the third one is as follows:

```
<p><a class="leftmargin" href="http://www.uky.edu/directory/">University of
Kentucky Directory</a></p>
```

At ❸, we locate the `<a>` tag nested in the third `<p>` tag. At ❹, we print out the `href` attribute of the `<a>` tag, and the output is as follows:

```
http://www.uky.edu/directory/
```

At ❺, we print out the content of the `<a>` tag by using the `.text` attribute, and the output is below:

```
University of Kentucky Directory
```

Scape Live Webpages

Now let's scape a live webpage. The HTML file for a live webpage is much more complicated than the simple example we have seen above. Sometimes the HTML file is thousands of lines long, and you'll need to learn how to locate the lines of code you want quickly.

Suppose you want to extract the contact information from the EKU finance faculty website. First go to the following website:

https://finance.eku.edu/people#_ga=2.106102083.1757686072.1659636719-6595508.1659636719

The page contains the information of the finance faculty members of EKU, as shown in Figure 4-2:

INSERT FIGURE 4-2 HERE

People

Maggie Abney

Executive in Residence for Banking
Department: Business - AFIS / MMIB
Office: BTC 163
Mailing Address: BTC 108
Email: maggie.abney@eku.edu
Phone: 859-622-8998

EDUCATION/CREDENTIALS: MBA – Eastern Kentucky University; **BS** - Finance, University of Kentucky; **Certified** - Corporate Treasury Management, plus 15 years experience working in the finance/banking industry
TEACHING INTERESTS: Personal Financial Management, Financial Planning, Entrepreneurial Finance, Banking and Financial Institutions, Financial Services; Summer Session faculty member at the Graduate School of Banking at Colorado.

Dr. Zek Eser

Assistant Professor of Finance
Department: Business - AFIS / MMIB
Office: BTC 177
Mailing Address: BTC 108
Email: zekeriya.eser@eku.edu
Phone: 859-622-6156

Figure 4-2: Information you want from a live webpage

Suppose you want to extract the name, phone number, and email address of the four faculty members. First you need to locate the corresponding tags in the HTML program.

Go back to the webpage and press CTRL-U on your keyboard. The source code for the webpage will appear. It is more than 400 lines long. To locate the tags you need, press CTRL-F and a search box will appear at the top right corner. Type in `abney` and search, and you'll find the corresponding HTML code as shown in Listing 4-3.

```
--snip--
❶     <div class="people-profile">
❷     <h2><a href="https://finance.eku.edu/people/abney" title=" Maggie
Abney "> Maggie   Abney </a></h2>
      <a
href="https://finance.eku.edu/sites/finance.eku.edu/files/people_images/maggi
e_abney_headshot_2020.jpg" rel="lightshow[field_people_photo][Maggie Abney]"
```

```
class="imagefield imagefield-lightbox2 imagefield-lightbox2-
directory_thumb_preset imagefield-field_people_photo imagecache imagecache-
field_people_photo imagecache-directory_thumb_preset imagecache-
field_people_photo-directory_thumb_preset lightbox-processed"><img
src="./People _ College Of Business _ Eastern Kentucky
University_files/maggie_abney_headshot_2020.jpg" alt="Maggie Abney"
title="Maggie Abney" width="125" height="156"></a>    <ul>
      <li>Executive in Residence for Banking</li>
<li><span>Department:</span> Business - AFIS / MMIB</li>
<li><span>Office:</span> BTC 163</li>    <li><span>Mailing Address:</span>
BTC 108</li>
❸  <li><span>Email:</span> <a
href="mailto:maggie.abney@eku.edu">maggie.abney@eku.edu</a></li>
❹<li><span>Phone:</span> 859-622-8998</li>
     <li> <br><span></span> <strong>EDUCATION/CREDENTIALS: </strong>
<strong>MBA</strong> - Eastern Kentucky University; <strong>BS</strong>
- Finance, University of Kentucky; <strong>Certified</strong> - Corporate
Treasury Management, plus 15 years experience working in the
finance/banking industry
<strong>TEACHING INTERESTS: </strong>
Personal Financial Management, Financial Planning, Entrepreneurial Finance,
Banking and Financial Institutions, Financial Services; Summer Session
faculty member at the Graduate School of Banking at Colorado.</li>



     </ul>
    </div>




</div>
  </div>
  <div class="views-row views-row-2 views-row-even">
```

```
<div id="node-24" class="node clear-block">

❺     <div class="people-profile">
    <h2><a href="https://finance.eku.edu/people/eser" title="Dr. Zek  Eser
">Dr. Zek  Eser </a></h2>
--snip--
```

Listing 4-3: Part of the source code for a live webpage

If you examine the code, you'll notice that the information for each faculty is encapsulated in a `<div>` tag with `class` attribute of `people-profile` ❶❺. Various information for the faculty (name, phone number, and email address) is in this parent `<div>` tag. The name of the faculty is in a level 2 heading; i.e., a `<h2>` tag❷. Within the parent `<div>` tag, there are multiple `<li>` tags. The email and phone number are in the 5th and 6th `<li>` tag❸❹.

To make things easy to understand, let's first retrieve the information about the first faculty member. Download the program *EKU_finance_first.py* from Canvas and save on your computer. The first part of the program is shown in Listing 4-4, and it locates the `<div>` tags for each of the three areas:

```
#import needed modules
from bs4 import BeautifulSoup
❶  import requests
#provide the web address of the live web
❷  url =
'https://finance.eku.edu/people#_ga=2.106102083.1757686072.1659636719-
6595508.1659636719'
#obtain information from the live web
❸  page = requests.get(url)
# parse the page to obtain the parent div tag
soup = BeautifulSoup(page.text, "html.parser")
# find all div tags with class attribute people-profile
❹  profs = soup.find_all('div', class_="people-profile")
# Let's first work on the first profile
prof1=profs[0]
#print(prof1)
# name is in the <h2> tag
```

```
name = prof1.find("h2")
❺  print(name.text)
# find all listings for this faculty
listings=prof1.find_all("li")

#print(listings)

# The fifth listing is the email
# first method to get email
email=listings[4].text
print(email)
# second method to get email
email2=listings[4].find("a")["href"]
print(email2)
# The sixth listing is the phone number
# get phone number
phone=listings[5].text
print(phone)
```

Listing 4-4: Python code to scrape a live webpage

We import the *requests* library at ❶ to obtain the source code from the live webpage. The address of the webpage is defined in the variable *url* ❷. At ❸, we use the `get()` function to fetch the HTML code. At ❹, we find all `<div>` tags with `class` value of `people-profile` and each contains the information about a faculty member.

We focus on the first faculty member. The name of the faculty member is in the level 2 heading, i.e., a `<h2>` tag. The name is the text in that tag ❺. We locate all the the three with the `<li>` tags within the parent `<div>` tag. The fifth element contains the email and the sixth the phone number. If you run the script, the output is the same as in Listing 4-5.

```
Maggie   Abney
Email: maggie.abney@eku.edu
mailto:maggie.abney@eku.edu
Phone: 859-622-8998
```

Listing 4-5: Output from EKU_finance_first.py

Now, we use the program *EKU_finance.py* to print out the name, email and phone number of all the faculty members. It is shown in Listing 4-6.

```
#import needed modules
from bs4 import BeautifulSoup
import requests
#provide the web address of the live web
url = 'https://finance.eku.edu/people#_ga=2.106102083.1757686072.1659636719-
6595508.1659636719'
#obtain information from the live web
page = requests.get(url)
# parse the page to obtain the parent div tag
soup = BeautifulSoup(page.text, "html.parser")
profs = soup.find_all('div', class_="people-profile")
# Now use a loop to print out all four professors' info
for prof in profs: #❶
    name = prof.find("h2")#❷
    print(name.text)
    listings=prof.find_all("li") #❸
    email=listings[4].text
    print(email)
    phone=listings[5].text
    print(phone)
```

Listing 4-6: Python code to print out the information for all faculty members

We then go into each element in the list *profs* ❶. To print out the name, we locate the `<h2>` tag ❷. The content of the tag is the faculty name. The two `<li>` tags contain the phone number and the email address of each faculty ❸, and we also print them out. The output is below:

```
Maggie   Abney
Email: maggie.abney@eku.edu
Phone: 859-622-8998
Dr. Zek  Eser
Email: zekeriya.eser@eku.edu
Phone: 859-622-6156
Dr. Benjamin   Woodruff
Email: benjamin.woodruff@eku.edu
Phone: 859-622-8881
 Sarah  Butler
Email: sarah.butler@eku.edu
Phone: 859-622-1087
```

NOTE: In the Beautiful Soup library, we sometimes have more than one way of accomplishing the same task. For example, `findAll()` *and* `find_all()` *work the same, and* `find_all('div', class_="people-profile")` *and* `find_all('div', {"class":"people-profile"})` *produce the same result. This is quite often in Python libraries because when a module/package/library is upgraded to a new version, functions in the old version still work in the new version, in order to retain backward compatibility.*

Play Online Podcasts

Our end goal is to write a program so that the program will broadcast the news brief from the website on your computer. We'll first learn how to extract the mp3 file associated with the podcast and play it.

Extract and Play Podcasts

First, you need to find a website with the newscast that you like. NPR News Now is a good place to get the latest news. It's updated every hour, 24/7. The web address is

https://www.npr.org/podcasts/500005/npr-news-now

If you go to the website, you will see a screen similar to Figure 4-3 below.

INSERT FIGURE 4-3 HERE

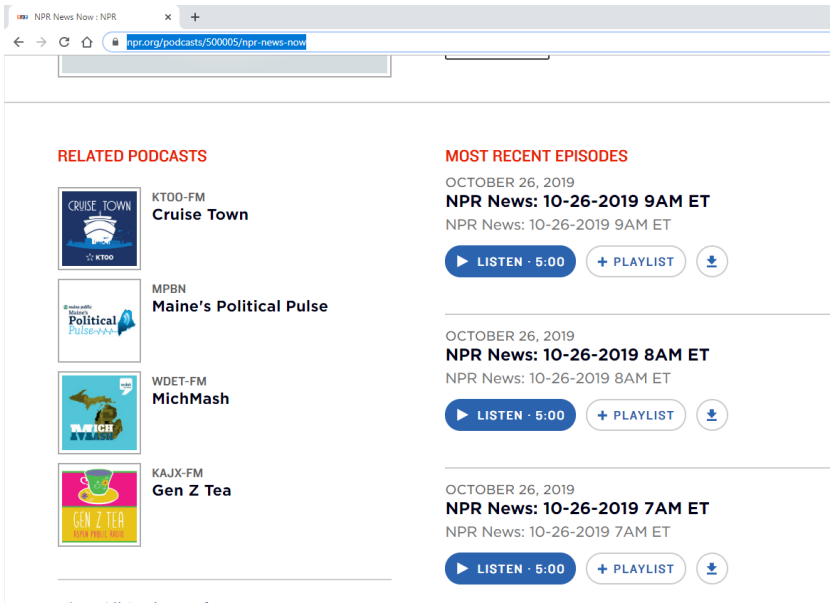

Figure 4-3: Front page of NPR News Now

As you can see, the latest news brief was updated 9am ET on Oct 26, 2019. Below it, you can also see news briefs at 8am, 7am, and so on.

To locate the mp3 file that contains the news brief, right click your mouse anywhere on the webpage, and a menu will show up. Select the *View page source* option and you will see the source code as in Figure 4-4:

INSERT FIGURE 4-4 HERE

Figure 4-4: Source code for the NPR News Now website

You'll notice that the mp3 files are contained in `<a>` tags. Therefore, we need to use the *Beautiful Soup* library to extract all `<a>` tags that contain mp3 files and extract the link from the first tag (because the first tag contains the latest news brief in this particular example). You can listen to previous news briefs as well: for example, the second and the third tags contain news briefs at 8am and 7am in Figure 4-3.

Next, we need to extract the link, remove unwanted components, and use the *webbrowser* library to pay the mp3 file.

The following program, *nprnews.py* in Listing 4-7, shows us how to accomplish the above:

```
#import needed modules
import requests
import bs4
import webbrowser
#locate the website for the NPR news brief
url = 'https://www.npr.org/podcasts/500005/npr-news-now'
#convert the source code to a soup string
```

```
response=requests.get(url)    #❶
soup = bs4.BeautifulSoup(response.text, 'html') #❷
#locate the tag that contains the mp3 files
casts = soup.findAll('a', {'class': 'audio-module-listen'})  #❸
print(casts)
#obtain the weblink for the mp3 file related to the latest news brief
cast=casts[0]['href']  #❹
print(cast)
#remove the unwanted compoents in the link
pos=cast.find("?")  #❺
print(cast[0:pos])    #❻
#extract the mp3 file link, and play the file
mymp3=cast[0:pos]
webbrowser.open(mymp3)    #❼
```

Listing 4-7: A program to play online podcasts

We first use the `get()` function in the *requests* library to obtain the source code of the NPR News Now website ❶. At ❷, we use the *Beautiful Soup* library to parse the text. We have seen from Figure 4-4 that the mp3 file is in an `<a>` tags with a `class` attribute of `audio-module-listen`. Therefore, line ❸ puts all those tags in the list *casts* by using the `findAll()` function in the *Beautiful Soup* library. The content of the list *casts* is shown in Listing 4-8.

```
[<a class="audio-module-listen" href="https://play.podtrac.com/npr-
500005/edge1.pod.npr.org/anon.npr-
mp3/npr/newscasts/2019/10/26/newscast090809.mp3?siteplayer=true&amp;size=4500
000&amp;awCollectionId=500005&amp;awEpisodeId=773692514&amp;dl=1">
<b class="audio-module-listen-inner">
<b class="audio-module-listen-icon icn-play"></b>
<b class="audio-module-listen-text">
<b class="audio-module-cta">Listen</b>
<b class="audio-module-listen-duration">
<span>· </span>
<span>5:00</span>
</b>
</b>
</b>
```

```
</a>, <a class="audio-module-listen" href="https://play.podtrac.com/npr-
500005/edge1.pod.npr.org/anon.npr-
mp3/npr/newscasts/2019/10/26/newscast080817.mp3?siteplayer=true&amp;size=4500
000&amp;awCollectionId=500005&amp;awEpisodeId=773687994&amp;dl=1">
<b class="audio-module-listen-inner">
<b class="audio-module-listen-icon icn-play"></b>
<b class="audio-module-listen-text">
<b class="audio-module-cta">Listen</b>
<b class="audio-module-listen-duration">
<span>· </span>
<span>5:00</span>
</b>
</b>
</b>
</a>, <a class="audio-module-listen" href="https://play.podtrac.com/npr-
500005/edge1.pod.npr.org/anon.npr-
mp3/npr/newscasts/2019/10/26/newscast070816.mp3?siteplayer=true&amp;size=4500
000&amp;awCollectionId=500005&amp;awEpisodeId=773683876&amp;dl=1">today is
August 20, 19
--snip--
</a>]
```

Listing 4-8: All <a> tags with `class` attribute of `audio-module-listen`

As you can see, there are multiple `<a>` tags with mp3 files. Line ❹ extracts the first `<a>` tag in the list and obtain the `href` attribute of the tag (i.e., the link to the mp3 file). The link is as follows:

```
https://play.podtrac.com/npr-500005/edge1.pod.npr.org/anon.npr-
mp3/npr/newscasts/2019/10/26/newscast090809.mp3?siteplayer=true&amp;size=4500
000&amp;awCollectionId=500005&amp;awEpisodeId=773692514&amp;dl=1
```

We need to trim the link so that it ends with ".mp3." To do that, we use the fact that the characters "?" are right after ".mp3" in the link. We use the string method `find()` to locate the position of "?" in the link ❺, and trim the link accordingly and print it out ❻. The trimmed link is as follows:

```
https://play.podtrac.com/npr-500005/edge1.pod.npr.org/anon.npr-
mp3/npr/newscasts/2019/10/26/newscast090809.mp3
```

Finally, we extract the link to the online mp3 file and play it ❼ by using the *webbrowser* library.

If you play the program, you will hear the latest NPR News Brief playing.

BEGIN BOX

TRY IT YOURSELF

4-1. Wait Wait… Don't Tell Me! is a weekly radio show and recent episodes are available as podcasts on this website:
https://www.npr.org/programs/wait-wait-dont-tell-me/
The source code is similar to that of the NPR News Now. Create a program similar to *nprnews.py* to play the latest episode of the online broadcast.

END BOX

Summary

In this chapter, you learned the basics of webscraping: how HTML works, what are different types and uses of HTML tags, how to use the *Beautiful Soup* library to parse HTML files and scrape the information you need.

Armed with this technique, you learned how to parse the source file of the online podcasts of NPR News Now and locate the mp3 file corresponding to the podcast. You then used the *webbrowser* library to play the online mp3 file.

End of Chapter Exercises

4-1. Modify *ParseLocal.py* to print out the `class` attribute value and the web address (i.e., the href attribute) of the first <a> tag in the file *UKYexample.html*.

4-2. If you go to the UK libraries webpage http://libraries.uky.edu/
You'll see the following at the bottom of the page:

Phone	(859) 218-1881
Emergencies	911 / #UKPD (#8573)
UKPD Dispatch	(859) 257-1616
Environmental Health & Safety	(859) 257-1376

Write a program from scratch so that you can print out the area and contact information for the four areas. Your output should look like this:

```
Phone
(859) 218-1881
Emergencies
911 / #UKPD (#8573)
UKPD Dispatch
(859) 257-1616
Environmental Health & Safety
(859) 257-1376
```

4-3. Create a program similar to *nprnews.py* to play the latest episode of the online podcast Wait Wait… Don't Tell Me! on this website (your browser may just download the mp3 file to your computer and then you need to click on the file to play it): https://www.npr.org/programs/wait-wait-dont-tell-me/