Chapter 15 — Reading Data from the Web

Introduction

This chapter gives a brief introduction to using some modules within the `urllib` package and an open-source Python library called BeautifulSoup. The first is used to make URLs and to read data from a Web server, the later to parse the results and extract data from the HTML.

The `urrllib` package includes the following modules:

- `urllib.request` — module to read a stream from a Web server.
- `urllib.error` — module that handles errors that may happen during a request.
- `urllib.parse` — module to parse URLs into their parts.
- `urllib.robotparser` — module to read and parse the ”robots.txt” file on a Web site.

BaautifulSoup 4.8 is the version used in developing this chapter, the simple examples should work with any 4.7 or greater version.

Objectives

Upon completion of this chapter's exercises, you should be able to:

- Open a request and make a connection to a remote web server.
- Read a stream of bytes from a request.
- Build requests that send data to the remote server using the GET method.
- Build requests that send data to the remote server using the POST method.
- Use a library to select HTML tags based on their CSS selector.
- Get and display attributes and inner text of HTML tags.

Prerequisites

This Chapter requires...

Opening a Request

To read data fro, the web, we must open a connection to an HTTP or HTTPS daemon running on a network server. As we open that connection, we will ask for a page, document, image, or other item. The server will return it as a stream of bytes, that we can read into our program.

import urllib.request	Module
Import the request module that us part of the urllib package. This module opens a connection to the web server and allows for retrieval of its output. https://docs.python.org/3/library/urllib.request.html	

urlopen(*url*)	Method of urllib.request
In its simple form, as shown here, it opens a connection to a web server to the resource specified in the URL. An HTTPResponse object is returned, if the URL is a HTTP or HTTPS request. The returned object will allow for reading the returned data. https://docs.python.org/3/library/urllib.request.html#urllib.request.urlopen	

read() **read(*amount*)**	Method of HTTPResponse
Read is method of HTTPresponse. It waits for the server to respond to the request and returns an array of bytes with data from the server. You may optionally specify the maximum number of bytes to reteieve. https://docs.python.org/3/library/http.client.html#http.client.HTTPResponse.read	

In this example, we open this book's Website and get the first 100 bytes of the home page. In the output you can see that it is an array of bytes that contains the start of an HTML document.

```
1| import urllib.request
2|
3| with urllib.request.urlopen("http://www.syw2l.org") as stream:
4|     html = stream.read(100)
5|     print(html)
```

```
b'<!DOCTYPE html>\r\n<html lang="en-US" class="no-js no-svg">\r\
n<head>\r\n\t<meta charset="UTF-8">\r\n\t<meta n'
```

The second example extends the previous one. The data is read from the server as an array of bytes and then decoded into a string using the utf-8 encoding. That encoding is very common and was chosen because of the charset meta in the data.

```
1| import urllib.request
```

```
2|
3| with urllib.request.urlopen("http://www.syw2l.org") as stream:
4|     # read and decode bytes back to a string
5|     html = stream.read(100).decode('utf-8')
6|     print(html)
```

```
<!DOCTYPE html>
<html lang="en-US" class="no-js no-svg">
<head>
  <meta charset="UTF-8">
  <meta n
```

Building a Request With the GET Method

A web client may send data to the web server, by adding it to the end of the URL in key/value pairs. This is called the GET method. Data sent by appending it to URL, after a question mark, can be cached, bookmarked, and stored in browser history. Data sent this way should never contain sensitive or private information. You should only send a limited amount of data using this method.

The `urrllib.parse` module makes appending data from a Python dictionary, simple. It automatically encodes characters that have meaning in a URL into their URL safe equivalent. All you have to do is append a question mark to the end of the URL and then append the encoded data.

This sample program uses a PHP program on one of the author's servers. The "reflect.php" script simply takes the data sent o the page, in either the GET or POST method, and returns a page with a table showing what was sent.

urlencode(dictionary_of_data)	Method of urllib.parse
Convert a dictionary of data into a unicode string in the URL format. Key/value pairs are returned in a format like: key0=value0&key1=vlalue1&key2=value2. Characters that have special meaning in a URL are converted to a safe format. link	

```
1| import urllib.request
2| import urllib.parse
3|
4| data = {"user":"jim","email":"jimbo@bogus.domain", "quantity":3}
5| parameters = urllib.parse.urlencode(data)
```

```
 6| url = "http://www.basicbook.org/reflect.php?" + parameters
 7| print(url)
 8|
 9| with urllib.request.urlopen(url) as stream:
10|     # read and decode bytes back to a string
11|     html = stream.read().decode('utf-8')
12|     print(html)
```

```
http://www.basicbook.org/reflect.php?email=jimbo
%40bogus.domain&quantity=3&user=jim
...
```

Making a Request With the POST Method

Data sent to a web server, using the POST method, is actually sent as a separate MIME document. The size limitation of the GET method is removed, a significant amount of data can be sent. If the data is being sent through an HTTPS connection, sensitive or private information may be sent this way.

In the sample program below, notice that the URL encoded data is converted into plain ASCII.

```
 1| import urllib.request
 2| import urllib.parse
 3|
 4| sendthis = {"user":"jim","email":"jimbo@bogus.domain",
     "quantity":3}
 5| data = urllib.parse.urlencode(sendthis)
 6| data = data.encode('ascii')
 7|
 8| url = "http://www.basicbook.org/reflect.php"
 9| print(url)
10|
11| with urllib.request.urlopen(url, data) as stream:
12|     # read and decode bytes back to a string
13|     html = stream.read().decode('utf-8')
14|     print(html)
```

```
http://www.basicbook.org/reflect.php
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Reflect Form Contents To Client</title>
```

```
  </head>
  <body>
    <section>
    <h1>Reflect Form Contents To Client</h1>
                                <table border=1>
         <caption>Post variables sent to the server.</caption>
         <tr><th>Name</th><th>Value</th></tr>


<tr><td>email</td><td>jimbo@bogus.domain</td></tr><tr><td>quanti
ty</td><td>3</td></tr><tr><td>user</td><td>jim</td></tr>
    </table>
              </section>
  </body>
</html>
```

Installing BeautifulSoup 4.x

By default, BeautifulSoup is not installed with Python3. Below you will see how to install it on a Windows system with Python3 installed or on two common LINUX distributions.

Windows

From the Windows command prompt (you may need to open as Administrator) type:

```
pip install beautifulsoup4
```

If the 'pip' installer is not installed you may need to:

- Download `get-pip.py` from https://github.com/pypa/get-pip and save it to a folder.
- Open the Windows command prompt and change into the directory containing `get-pip.py`.
- `python get-pip.py`
- `pip install beautifulsoup4`

Execute

Ubuntu/Debian style LINUX

From a terminal window, like xterm, you may install BeautifulSoup 4.x using pip. If pip is not available to you, you may install it by using the Debian Advanced Package Tool (apt).

```
pip install beautifulsoup4
```

or

```
apt-get install python3-bs4
```

Red Hat/CentOS style LINUX

From a terminal window you may install BeautifulSoup 4.x using pip, or by using the Debian Advanced Package Tool (apt).

```
pip install beautifulsoup4
```

or

```
yum install python-beautifulsoup4
```

Parsing HTML and Showing it Nicely

Before using BeautifulSoup, you must import the `bs4` library into your program. Once we do this, we can create an object that will parse our HTML document for us. BeautifulSoup will automatically convert the page it is asked to process into UTF-8 and will accept the HTML as a string or as a byte array, as returned by a `urllib.request` object.

`import bs4`	Module
Import the BeautifulSoup 4.x library into your program. https://www.crummy.com/software/BeautifulSoup/bs4/doc/#	

`bs4.BeautifulSoup(`*`source, parser`*`)`	Object in bs4
The BeautifulSoup object creator takes a minimum of two arguments; the first should contain either a string or a byte array with the HTML document; the second needs to be the name of the parser to use to read the page. For most pages, It is recommended that you use the string `'html.parser'`. There are many additional options that can be found in the documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/#beautifulsoup	

soup.**prettify()**	Method of BeautifulSoup
The `.prettify()` method returns a string with HTML that is formatted for easier reading. https://www.crummy.com/software/BeautifulSoup/bs4/doc/#pretty-printing	

The statement, on line 5, creates the BeautifulSoup object. It requires two arguments. The first is the HTML to parse, and the second is the parser to use. For HTML documents, it is recommended that you use the `'html.parser'`. The `.prettify()` method of the BeautifulSoup object will take the original HTML document and add line breaks and spaces to make the code display nicely.

```
1| import bs4
2|
3| pg = "<html><head><title>Header Foo</title></head><body><h1>Page
   Header</h1><p>para1</p><p>para2</p><p>para3</p></body></html>
   "
4|
5| soup = bs4.BeautifulSoup(pg, "html.parser")
6|
7| print(soup.prettify())
```

```
<html>
 <head>
  <title>
   Header Foo
  </title>
 </head>
 <body>
  <h1>
   Page Header
  </h1>
  <p>
   para1
  </p>
  <p>
   para2
  </p>
  <p>
   para3
  </p>
 </body>
```

```
</html>
```

Selecting Elements on a Page

BeautifulSoup 4.7 and versions after includes the ability to use virtually all CSS selectors to find elements within a page. There are additional methods to find element, but this introduction will just cover one. Before we show how to get an element in BeautifulSoup, we will start with a short introduction in creating CSS selectors and how to combine them.

Simple CSS Selectors

Cascading Style Sheets (CSS) consist of selectors that are used to match to elements on a page, followed by rules to apply to the presentation of those tags. Simple CSS selectors fall into four groups:

1. The general selector gets everything.
   a) *
2. Tag selectors that use a tag name and apply to all tags of that type, like:
   a) a
   b) p
   c) h1
3. Id selectors, starting with a #, that select a tag with that specific id:
   a) #page_name
   b) #author
4. Class selectors, beginning with a period (.), that will select the tags that have that class assigned:
   a) .important_items
   b) .cruddystuff

Some CSS Combiantors

There are also many ways to combine simple selectors into complex selectors that can be used to zero in on specific tags. The list below shows a few of the CSS combinators, but it by no means is complete.

- This and That Combinator
  selector_one, selector_two
  Using a comma (,) to separate selectors will select either. The selector “h1, h2” would return a list of both the h1 and h2 elements on a page.
- Descendant Combinator
  selector_one selector_two

By placing a space ( ) between selectors you select inside the previous select. The selector "#masthead a" would return the anchors inside the block element with the id of "masthead".

- Adjacent Sibling Combinator
  selector_one + selector_two
  Use a plus (+) between selectors to find the element directly after the first selector on the same level. This finds siblings not children. The selector ".stuff p" would find the paragraph that directly follow elements with the class "stuff".
- General Sibling Combinator
  selector_one ~ selector_two
  When you put a tilde (~) between selectors you get all of the second one after the first on the same structural level. The selector "#chapter_one ~ p" would get all of the paragraphs following chapter_one. The siblings do not have to be adjacent, just follow.
- Direct Children Combinator
  selector_one > selector_two
  A greater than (>) directs CSS to look for direct children, elements directly within another element. The selector "a > img" would return image tags that are directly inside anchor tags, and no others. This only gets direct children and not children of children.

There are additional CSS selectors, combinators, pseudo-tags, and other features that are beyond the scope of this. Please see one of the many CSS selector references on-line.

Using a CSS Selector in BeautifulSoup

The `.select()` method of BeautifulSoup uses a CSS selector and returns a collection of elements that meet the selection criteria. If there are no elements that meet the criteria, then an empty collection will be returned. You may test for a non-empty collection by using the `len()` function or by using an `if` statement, from earlier in the book.

soup.**select(***css_selector***)**	Method of BeautifulSoup
This method returns an iterable collection of elements on the page that were selected based upon the CSS selector passed. If no elements are retrieved an empty colelction will be returned. https://www.crummy.com/software/BeautifulSoup/bs4/doc/#css-selectors	

```
1| import urllib.request
2| import bs4
3|
4| with urllib.request.urlopen("http://www.syw2l.org") as stream:
5|     html = stream.read()
```

```
6|      soup = bs4.BeautifulSoup(html, 'html.parser')
7|
8|      # get first paragraph inside a tag with a class of site-info
9|      # remember to use the indexing operator to get the first one
10|     print('********** get first paragraph in site_info')
11|     tag = soup.select(".site-info p")[0]
12|     print(tag)
13|
14|     # find all img tags
15|     print("********** all em tags on page")
16|     for tag in soup.select("em"):
17|         print(tag)
```

```
********** get first paragraph in site_info
<p>C) 2019 J.M.Reneau - All Rights Reserved - Wordpress design
by <a href="https://www.luzuk.com/"
target="_blank">Luzuk</a></p>
********** all em tags on page
<em>So you Want to Learn to Program - BASIC-256 - Book</em>
<em> </em>
<em>An Introduction to STEM Programming with Python 3 -
Book</em>
```

Getting Text and Attributes from Elements

In the previous example we can see each of the HTML tags found, and we printed out the HTML we found. Now that we have our list of tags, from select, let's get attributes from a tag and the text

The following sample program shows two ways to get attributes of a HTML statement, and two ways to get the inner stuff.

`soup_element[attr_name]`	BeautifulSoup Element Operator
The indexing operator, on an element, will get an attributes value. If the attributed does not exist, an error will be thrown. If the attribute is optional, it is recommended that the `.get()` method of generic collections be used, with a default value. https://www.crummy.com/software/BeautifulSoup/bs4/doc/#quick-start	

`soup_element.string`	BeautifulSoup Element Property

This property returns the string value between the opening tag and the closing tag. If there is ANY HTML inside the tags, this property will return the Python None value. https://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigablestring	

soup_element.**get_text()**	BeautifulSoup Method
This property returns the text found inside a pair of tags with the HTML stripped out. This method is very useful for getting the text of a paragraph or other tgs. https://www.crummy.com/software/BeautifulSoup/bs4/doc/#get-text	

```
1| import urllib.request
2| import bs4
3|
4| page = """
5| <!DOCTYPE html>
6| <html lang='en'>
7| <head>
8| <title>Test Page With Table</title>
9| <style>
10| body { background-color: #dddddd; }
11| </style>
12| </head>
13| <body>
14| <h1>A table</h1>
15| <table>
16| <tr><th>Fruit</th><th>Color</th></tr>
17| <tr><td>Banana</td><td>Yellow</td></tr>
18| <tr><td>Lemon</td><td>Yellow</td></tr>
19| <tr><td>Orange</td><td>Orange</td></tr>
20| </table>
21| <h1>Links</h1>
22| <ul>
23| <li><a href="http://www.syw2l.org">The Book's Web Site</a></li>
24| <li><a
      href="https://www.crummy.com/software/BeautifulSoup/">Beautif
      ul Soup</a></li>
25| <li><a href="http://www.python.org">python</a></li>
26| </ul>
27| </body>
28| </html>
```

```
"""

soup = bs4.BeautifulSoup(page, 'html.parser')

# get the linked pages from te anchors on page
tags = soup.select('a')
for tag in tags:
   print(tag['href'])

# get the attribute language from the html tag
# it may not exist - if not use unknown
tag = soup.select("html")[0]
print("this page's language is", tag.get('lang','unknown'))

# create a comma separated printout of the
# string content of all th and td tags
rows = soup.select("table tr")
for row in rows:
    cols = row.select("th,td")
    print(cols[0].string, ',', cols[1].string)

# show text on first row - strip out all html
print(rows[0].text)
```

```
http://www.syw2l.org
https://www.crummy.com/software/BeautifulSoup/
http://www.python.org
this page's language is en
Fruit , Color
Banana , Yellow
Lemon , Yellow
Orange , Orange
FruitColor
```

Another example that gets the current weather observation at an airport from the U.S. National Weather Service, follows:

```
import bs4
import urllib.request

#change city code to four letter US airport code
# KLAX-Los Angeles, KJFK-New York City, KDWU - Ashland Kentucky
city = "KDWU"
```

```
 7| url = "https://w1.weather.gov/obhistory/"+ city+ ".html"
 8|
 9| print(url)
10|
11| with urllib.request.urlopen(url) as stream:
12|     html = stream.read()
13|     soup = bs4.BeautifulSoup(html, 'html.parser')
14|
15|     # get the actual name from the first element with
16|     # class name of "white1"
17|     cityname = soup.select(".white1")[0].get_text()
18|
19|     # find the first table row with a bakground color of
   "#eeeeee"
20|     # is is the most recent observation
21|     datarow = soup.select('tr[bgcolor="#eeeeee"]')[0]
22|
23|     # get cells from the row
24|     cells = datarow.select("td")
25|
26|     # print out weather
27|     print("At the", cityname,
28|           'at', cells[1].text,
29|           'it is', cells[4].text,
30|           'with a tempature of', cells[6].text,
31|           'and a relative humidity of', cells[10].text,
32|           '.')
```

```
https://w1.weather.gov/obhistory/KDWU.html
At the Ashland Regional Airport at 22:56 it is Fair with a
tempature of 73 and a relative humidity of 76% .
```

Summary

Goes here

Important Terms

here

Exercises

Here

Word Search

References

https://docs.python.org/3/library/urllib.html
https://www.w3schools.com/tags/ref_httpmethods.asp
https://en.m.wikipedia.org/wiki/MIME
https://www.crummy.com/software/BeautifulSoup