Automated Data Collection With Python

The Steps of Automatically Collecting Data Online

- Download the HTML Source of a page
- Extract the content from the HTML
- Save the Content
- Repeat the Process on A Different Page

All of these steps can be automated, running indepdent of human interaction

Step 0: Import the needed libraries

Before beginning, make sure you have Python installed on your computer with the necessary libraries.

If you do not have Python installed, a recommended distribution is **Anaconda**:
https://www.continuum.io/downloads (https://www.continuum.io/downloads)

The 2.7 version of Python is recommended for people new to Python.

We are first going to load two key libraries

- the urllib library
- the BeautifulSoup library

The "urllib" library allows us to access a webpage with Python and download the HTML

The "BeautifulSoup" library allows us to parse the HTML of a webpage and isolate content within HTML tags

In [ ]:
```
import urllib
from bs4 import BeautifulSoup
```

Step 1: Download the HTML Source of a page

- 1.1 Direct Python to Access the Webpage, and Save the Page's HTML to a variable
- 1.2 Pass the downloaded HTML to an HTML Parser in Python (BeautifulSoup)
- 1.3 Examine the downloaded content

1.1 Direct Python to Access the Webpage, and Save the Page's HTML to a variable

The first step of scraping information from the web is downloading the source code of a specific webpage.

You should have an idea in mind of the page(s) that contain the information you need.

When you know what page you need to extract content from, then you can direct Python to download it.

The code below:

- Create a variable, called "url", that has the address of the webpage we want to scrape
- Download the html for the webpage and save it as a variable called "sourcecode"

In [137]:
```
url = "http://ivanhernandez.com/example.html"
sourcecode = urllib.urlopen(url)
```

1.2 Pass the downloaded HTML to an HTML Parser in Python (BeautifulSoup)

Now that we have the webpage downloaded, we need to parse through the elements of the source code in an easy way.

The BeautifulSoup library allows us to easily access the different elements of a webpage.

The code below, passes the source code to the parser, and we can query the different elements of the page through the "soup" variable

In [138]:
```
soup = BeautifulSoup(sourcecode, 'html.parser')
```

We can examine the contents of the source code using the "prettify" function, which displays the source code of the webpage.

```
In [133]: print soup.prettify()
```

```
<html>
 <head>
  <title>
   Example Page
  </title>
 </head>
 <body>
  <h1>
   This Title is in an H1 tag
  </h1>
  <div class="box1">
   This text is inside a div tag, whose class is equal to box1
  </div>
  <br>
   <div class="box2">
    This text is inside a div tag, whose class is equal to box2
   </div>
   <br>
    <span class="box3">
     This text is inside a span tag, whose class is equal to box3
    </span>
    <p id="box4">
     This text is inside a p tag, whose id is equal to box4
    </p>
    <a href="http://google.com">
     This is a link to Google
    </a>
    <br>
     <br>
      Additional Content: This content is not inside any tag
     </br>
    </br>
   </br>
  </br>
 </body>
</html>
```

Step 2: Extracting Content from a Page

Once you have the page source downloaded and parsed, you are ready to tell Python to extract the content you want.

If we know the type of tag (div, p, a, etc.), and we know an identifying feature (class, id, name, etc.), we can grab the content in that specified tag.

Depending on your project and goals, you will find yourself in one these possible situations:

- 2.1 You want a **Single Piece of Text** Data from a Page
  - a. You want content from a specific tag with a specific id/class/name
  - b. You want content from a specific tag, and the id/class/name does not matter

- 2.2 You want **Multiple Pieces of Text** Data from a Page
  - a. You want all the content from a specific tag that occurs many times
  - b. You want all the content that could come from many possible tags
  - c. You want all the content that could come from many possible id/class names
  - d. You want all the content that comes from a specific tag, and partially matches an id/class/name

- 2.3 You want **Information from Links**
  - a. You want to get the link text
  - b. You want to get the url of a link

- 2.4 You want to Extract **Non-tagged** Content

2.1 Collecting a Single Piece of Text Data from a Page

2.1.a When you know the tag name and identifying information

Let's grab the content from the div tag with a class equal to "box1"

```
In [127]: content = soup.find("div", {"class": "box1"})
          content.text
```

```
Out[127]: u'This text is inside a div tag, whose class is equal to box1'
```

Let's grab the content from the div tag with a class equal to "box2"

```
In [128]: content = soup.find("div", {"class": "box2"})
          content.text

Out[128]: u'This text is inside a div tag, whose class is equal to box2'
```

2.1.b When you only know/need the tag name

Let's grab the content from the p tag

```
In [129]: content = soup.find("p")
          content.text

Out[129]: u'This text is inside a p tag, whose id is equal to box4'
```

Let's grab the content from the span tag

```
In [139]: content = soup.find("span")
          content.text

Out[139]: u'This text is inside a span tag, whose class is equal to box3'
```

2.2 Collecting Multiple Pieces of Text Data from a Page

If there are multiple tags you want to extract content from, you can use the "findAll" function.

2.2.a When you know the tag name

For example, there are two div tags in the entire HTML (the one with class=box1 and the one with class=box2).

If we wanted to extract all of the content at once, then just ask BeautifulSoup to find all the div tags and save them to a list.

```
In [140]: contents = soup.findAll("div")
          contents

Out[140]: [<div class="box1">This text is inside a div tag, whose class is equa
          l to box1</div>,
           <div class="box2">This text is inside a div tag, whose class is equa
          l to box2</div>]
```

The findAll function saves the results in a list (indicated by the surrounding square-brackets)

We can iteratively access the specific content in the list using a "for loop"

```
In [141]: for content in contents:
              print content.text
```

```
This text is inside a div tag, whose class is equal to box1
This text is inside a div tag, whose class is equal to box2
```

2.2.b When there are many different tag names

If we wanted to extract content that could be in either a div or a span or a p tag, then, we can place them in a list (within square brackets), and run the findAll function using that list of tags.

Below, we specify that we want returned any content within a div, span, or p tag.

```
In [142]: contents = soup.findAll(["div","span","p"])
          for content in contents:
              print content.text
```

```
This text is inside a div tag, whose class is equal to box1
This text is inside a div tag, whose class is equal to box2
This text is inside a span tag, whose class is equal to box3
This text is inside a p tag, whose id is equal to box4
```

2.2.c When there are many different id/class names

If we know precisely the names of the classes/ids that could match, we can specify the tag name, as well all the id and class names we want to match in a list.

```
In [143]: import re

          contents = soup.findAll("div", {"class" : ["box1","box2"]})

          for content in contents:
              print content.text
```

```
This text is inside a div tag, whose class is equal to box1
This text is inside a div tag, whose class is equal to box2
```

2.2.d When you only know the tag and part of the class name

We may be a situation where we know what we want to extract (e.g., we want something that is in between div tags , that has a class with the word "box" in it).

Using the regular expression library (whose library is called "re"), we can have partial matches of tags or classes/ids.

```
In [144]: import re

          contents = soup.findAll("div", {"class" : re.compile('box.*')})

          for content in contents:
              print content.text

          This text is inside a div tag, whose class is equal to box1
          This text is inside a div tag, whose class is equal to box2
```

2.3 Extract content from a link

We may have link that, if clicked, direct the user to a different page. We can extract various content from a link including

- The text the user sees for the link
- The address the link directs the user to go, when clicked

2.3.a Extract the text from the link

```
In [145]: content = soup.find("a")
          content.text

Out[145]: u'This is a link to Google'
```

2.3.b Extract the url from the link

```
In [146]: content = soup.find("a")
          content["href"]

Out[146]: u'http://google.com'
```

2.4 Extract Everything Else

Sometimes, we have content that is not contained within tags, or has irregular structure. We can still extract this information if we know the characters that come immediately before and immediately after the content.

- Determine the HTML that goes immediately before the content (precontent)
  - In this example, "Additional Content:" came before the text we want to extract
  - We will split the HTML on where it says "Additional Content:" and keep the text after it

- Determine the HTML that goes immediately after the content (postcontent)
  - In this example, "<" came immediately after the text we want to extract
  - We will split the HTML that we kept on where it says "<" and keep the text before it

In [147]:
```
soup.text.split("Additional Content:")[1].split("<")[0]
```

Out[147]:
```
u' This content is not inside any tag\r\n\n'
```

Step 3: Saving the content

When we have extracted the content, we need to preserve the information for subsequent analysis.

You can save the content you extracted in a

- list (within Python's memory)
- text file (on your hard drive)

3.1 Saving the content to a list

In [148]:
```
data = []

content = soup.find("div", {"class": "box1"})
data.append(content.text)

content = soup.find("div", {"class": "box2"})
data.append(content.text)

content = soup.find("span", {"class": "box3"})
data.append(content.text)

content = soup.find("p", {"id": "box4"})
data.append(content.text)
```

In [149]:
```
for item in data:
    print item
```

```
This text is inside a div tag, whose class is equal to box1
This text is inside a div tag, whose class is equal to box2
This text is inside a span tag, whose class is equal to box3
This text is inside a p tag, whose id is equal to box4
```

3.2 Saving the contents to a text file

```
In [150]: textfile = open("data.txt","ab")

          content = soup.find("div", {"class": "box1"})
          textfile.write(content.text)
          textfile.write("\r\n")

          content = soup.find("div", {"class": "box2"})
          textfile.write(content.text)
          textfile.write("\r\n")


          textfile.close()
```

Viewing the contents of a text file

```
In [151]: textfile = open("data.txt","rb")
          print textfile.read()
```

```
This text is inside a div tag, whose class is equal to box1
This text is inside a div tag, whose class is equal to box2
```

4. Loop through many pages

When you only want a single piece of information from each page

Below, we know that we want to access the health section, the techonology section, and the science section of the New York Times. If we know ahead of time the ways we want to modify the url to access the pages,

- Start with a list of what we want to add to the url
- Make an empty list to hold all of the data you collect
- Make a url by combining a baseline url with the part we want to add
- Go through each page
- On each page, extract the data
- Add the data to the list that holds all of the data

In [152]:
```python
pages = ["health","technology","science"]

titles = []
for topic in pages:
    url = "https://www.nytimes.com/section/" + topic
    sourcecode = urllib.urlopen(url)
    soup = BeautifulSoup(sourcecode, 'html.parser')
    title = soup.find('title')
    titles.append(title.text)
```

In [153]:
```python
print titles
```

```
[u'Health - The New York Times', u'Technology - The New York Times',
 u'Science - The New York Times']
```

When you want multiple pieces of information from each page

We may want to access multiple pieces of content, from many different webpages. This situation requires two for-loops. The first loop goes through each webpage extension ("health", "technology", "science), to access the specific topic's page. Then, when we are on the page, we with find all of the relevant content, and make a for-loop that goes through each result and appends it to our list of results.

- Start with a list of webpage urls
- Make an empty list to hold all of the data you collect
- Make a For-Loop to go through each page
- On each page, find all instances of the tag
- Make a For-Loop to go through each instance found, and extract the data
- Add the data to the list that holds all of the data

In [154]:
```python
pages = ["health","technology","science"]

headlinelist = []
for topic in pages:
    url = "https://www.nytimes.com/section/" + topic
    sourcecode = urllib.urlopen(url)
    soup = BeautifulSoup(sourcecode, 'html.parser')
    contents = soup.findAll(["h2","h3"],{"class":"headline"})

    for content in contents:

        headline = content.text #Get the text
        headlinelist.append(headline)
```

```
In [ ]: print headlinelist
```

Combining Everything

There are many possibilities for what you can do. You can combine the different functions together to collect the links from one page and then use those links to collect data from the subsequent page.

The code below, retrieves all of the links to job specific pages, and then goes to each of the links and retreives the job name and median wage.

Finally, it writes both pieces of information to a textfile for every job.

```
In [101]: url="https://www.onetonline.org/find/family?f=27&g=Go"
sourcecode = urllib.urlopen(url)
soup = BeautifulSoup(sourcecode, 'html.parser')
joblinks = soup.findAll("a", {"href": re.compile("https://www.onetonl
ine.org/link/summary.*")})

for link in joblinks:
    url = "https://www.onetonline.org/link/details/"+jobid.text
    sourcecode = urllib.urlopen(url)
    soup = BeautifulSoup(sourcecode, 'html.parser')
    income = soup.text.split("Median wages (2016)")[1].split(" ")
[0].strip()
    job = soup.find("title").text
    textfile = open("data.txt","a")
    textfile.write(job + "," + income + "\r\n")
    textfile.close()
```

Summary: Automated Data Collection

- Decide on what content you want to scrape
- Get a list of the page urls you want to scrape
- Make a For-Loop that goes through every single page
- Scrape the content from each page
- Save the content to a text file

Additional notes

- Use the time.sleep(10) function to pause your script's execution for 10 seconds to not overload the server you are trying to collect
- Be respectful of a site's Terms of Service Policies
- Use the "mechanize" library to fill in forms for websites that want you to sign-in or enter information to access the content
- Try different things: Often there is not just one solution for collecting the data - many ways to parse the HTML