

Web Scraping

LO 4

Objective

After attending this session, you should be able

- Introduction to web scraping
- BeautifulSoup package installation
- Hands on exercise

Web Scraping

- **Web scraping** is the process of gathering information from the Internet.
- However, the term web scraping usually involves **automation**.

Web Scraping

- Python has numerous libraries for approaching this type of problem, many of which are incredibly powerful
- Popular web scrapping python packages:
 - » Pattern
 - » Requests
 - » Scrapy
 - » BeautifulSoup
 - » Mechanize
- In this course we are covering Beautiful Soup which is most popular in the lot
- But these packages can work together too

Typical HTML structure

```
simple.html X
<html>
  <head>
    <title>
      Simple Web page
    </title>
  </head>
  <body>
    <p id="First para" align="center">
      First paragraph
      <b>
        Hello Students
      </b>
    </p>
    <p id="Second para" align="center">
      This is basic HTML
      <b>
        two
      </b>
    </p>
  </body>
</html>
```



Note: Save this file with a .html extension

What's Beautiful Soup?



- Beautiful Soup is a Python library for pulling data out of HTML and XML files via screen scraping
- Three key features:
 - It provides a few simple methods for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need.
 - It converts incoming documents to Unicode and outgoing documents to UTF-8, so you don't have to think about encodings
 - It sits on top of popular Python parsers like **lxml** and **html5lib**, allowing you to try out different parsing strategies or trade speed for flexibility.
 - A **parser** is a compiler or interpreter component that breaks data into smaller elements for easy translation into another language
 - **lxml** provides a very simple and powerful API for parsing XML and HTML.
 - **html5lib** is a pure-python library for parsing HTML

BeautifulSoup Installation

→ If you run Debian or Ubuntu, you can install Beautiful Soup with the system package manager:

- » `sudo apt-get install python-bs4`

→ To install from PyPi:

- » `easy_install beautifulsoup4`
- or
- » `pip install beautifulsoup4`

→ If you have downloaded the source tarball and want to install manually:

- » `python setup.py install`

→ Refer <http://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-beautiful-soup> to avoid any installation related errors and to install other useful packages like lxml parser

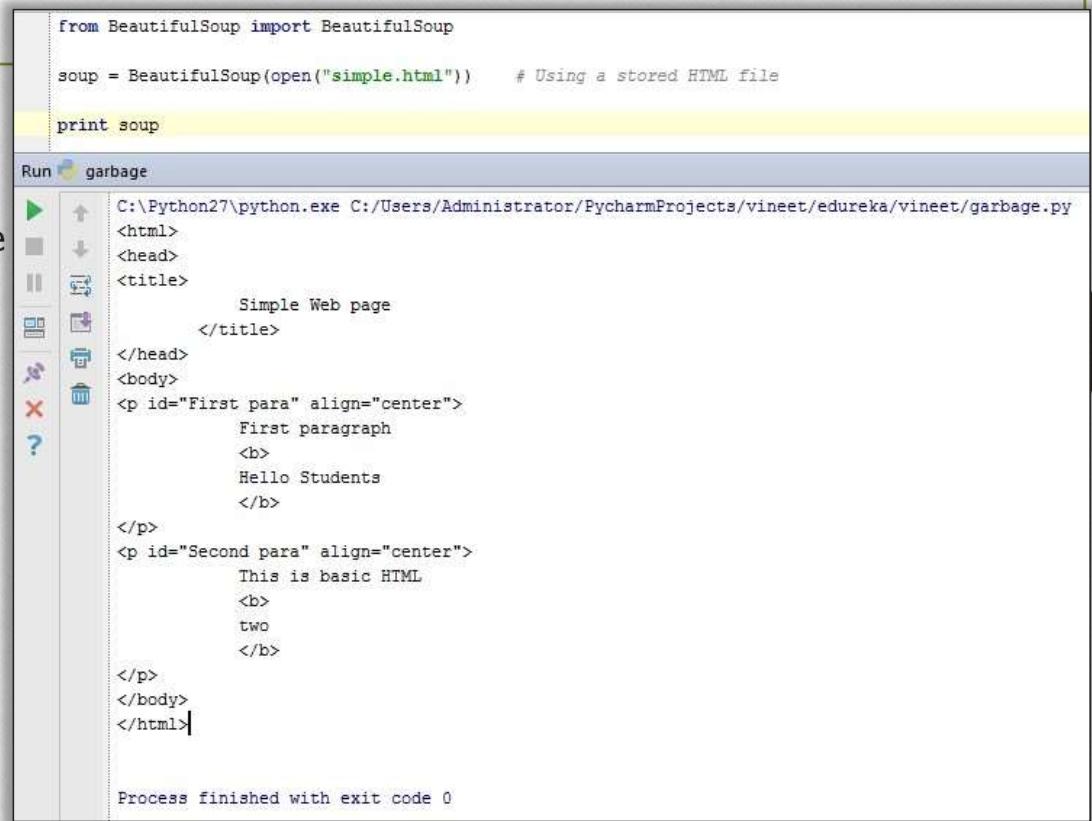
BeautifulSoup for Parsing a Doc

- To parse a document, pass it into the BeautifulSoup constructor
- We can pass in a string or an open filehandle
- Example:

```
from bs4 import BeautifulSoup

# Using a stored HTML file
soup = BeautifulSoup(open("simple.html"))

# Entire HTML doc can be passed
soup = BeautifulSoup("<html>data</html>")
```



The screenshot shows the PyCharm IDE interface. The code editor contains Python code to parse an HTML file. The run tool window is open, showing the output of the script. The output pane displays the parsed HTML structure, including the title, body, and two paragraphs with bold text.

```
from BeautifulSoup import BeautifulSoup

soup = BeautifulSoup(open("simple.html"))      # Using a stored HTML file
print soup

Run garbage
C:\Python27\python.exe C:/Users/Administrator/PycharmProjects/vineet/edureka/vineet/garbage.py

<html>
<head>
<title>
    Simple Web page
    </title>
</head>
<body>
<p id="First para" align="center">
    First paragraph
    <b>
        Hello Students
    </b>
</p>
<p id="Second para" align="center">
    This is basic HTML
    <b>
        two
    </b>
</p>
</body>
</html>

Process finished with exit code 0
```

Different Objects

→ Beautiful Soup transforms a complex HTML document into a complex tree of Python objects.

→ Example: `soup = BeautifulSoup('<b class="price">New Rate ')`

→ Tag Object:

» A Tag object corresponds to a HTML tag in the original document.

```
In[7]: soup = BeautifulSoup('<b class="price">New Rate</b>')
In[8]: tag = soup.b ←
In[9]: type(tag)
Out[9]: BeautifulSoup.Tag
```

→ Attributes Object:

» A tag may have any number of attributes.
» The tag `<p class="price">` has an attribute "class" whose value is "price".
» You can access a tag's attributes by treating the tag like a dictionary:
» `tag['class']`

```
In[10]: tag['class'] ←
Out[10]: u'price'
```

» Access attributes by `.attrs`:

```
In[11]: tag.attrs ←
Out[11]: [(u'class', u'price')]
```

Different Objects(Contd.)

→ NavigableString Object:

- » Beautiful Soup uses the NavigableString class to contain bits of text within a tag:

```
In[12]: tag.string ←  
Out[12]: u'New Rate'
```

→ Comments:

- » This is a special type of NavigableString Object:

```
In[13]: soup = BeautifulSoup("<b><!--This is comment--></b>") ←  
In[14]: comment = soup.b.string  
In[15]: comment  
Out[15]: u'This is comment'  
In[16]: type(comment)  
Out[16]: BeautifulSoup.Comment
```

Let's explore practical

- Installation of modules

Pip install beautifulsoup4

Pip install lxml #parsers to correct missing information of the html data

Pip install requests # allows you to send HTTP requests using Python

Test Website

Article 1 Headline

This is a summary of article 1

Article 2 Headline

This is a summary of article 2

Footer Information

```
1  <!doctype html>
2  <html class="no-js" lang="">
3      <head>
4          <title>Test - A Sample Website</title>
5          <meta charset="utf-8">
6          <link rel="stylesheet" href="css/normalize.css">
7          <link rel="stylesheet" href="css/main.css">
8      </head>
9      <body>
10         <h1 id='site_title'>Test Website</h1>
11         <hr></hr>
12         <div class="article">
13             <h2><a href="article_1.html">Article 1 Headline</a></h2>
14             <p>This is a summary of article 1</p>
15         </div>
16         <hr></hr>
17         <div class="article">
18             <h2><a href="article_2.html">Article 2 Headline</a></h2>
19             <p>This is a summary of article 2</p>
20         </div>
21         <hr></hr>
22
23     <div class='footer'>
```

Test Website

Article 1 Headline

This is a summary of article 1

Article 2 Headline

This is a summary of article 2

Footer Information

```
1 from bs4 import BeautifulSoup
• 2 import requests
3
4 with open('simple.html') as html_file:
5     soup = BeautifulSoup(html_file, 'lxml')
6
7 print(soup)
8
```

```
<h2><a href="article_1.html">Article 1 Headline</a></h2>
<p>This is a summary of article 1</p>
</div>
<hr/>
<div class="article">
<h2><a href="article_2.html">Article 2 Headline</a></h2>
<p>This is a summary of article 2</p>
</div>
<hr/>
```

Prettify method

Test Website

Article 1 Headline

This is a summary of article 1

Article 2 Headline

This is a summary of article 2

```
1 from bs4 import BeautifulSoup
• 2 import requests
3
4 with open('simple.html') as html_file:
5     soup = BeautifulSoup(html_file, 'lxml')
6
• 7 print(soup.prettify())
8
```

```
<!DOCTYPE html>
<html class="no-js" lang="">
    <head>
        <title>
            Test - A Sample Website
        </title>
        <meta charset="utf-8"/>
        <link href="css/normalize.css" rel="stylesheet"/>
        <link href="css/main.css" rel="stylesheet"/>
```

Title tag

Test Website

Article 1 Headline

This is a summary of article 1

Article 2 Headline

This is a summary of article 2

Footer Information

```
1 from bs4 import BeautifulSoup
2 import requests
3
4 with open('simple.html') as html_file:
5     soup = BeautifulSoup(html_file, 'lxml')
6
7 match = soup.title| i
8 print(match)
9
```

```
<title>Test - A Sample Website</title>
```

Title Text

```
1 from bs4 import BeautifulSoup
2 import requests
3
4 with open('simple.html') as html_file:
5     soup = BeautifulSoup(html_file, 'lxml')
6
7 match = soup.title.text
8 print(match)
9 |
```

Test - A Sample Website

Try yourself

- Match =soup.div #first div
- Match = soup.find('div') #first div
- Match = soup.find('div', class_='footer_') # class is keyword in python
- Inspect the html page

Test Website

Article 1 Headline

This is a summary of article 1

Article 2 Headline

This is a summary of article 2

Footer Information

The screenshot shows the browser's developer tools with the 'Elements' tab selected. The left sidebar displays the DOM tree:

```
><head>...</head>
<body>
  <h1 id="site_title">Test Website</h1>
  <hr>
  <div class="article">
    <h2>
      <a href="article_1.html">Article 1 Headline</a> == $0
    </h2>
    <p>This is a summary of article 1</p>
  </div>
  <hr>
  <div class="article">
```

The right panel shows the corresponding HTML code:

```
<div class="article">
  <h2><a href="article_1.html">Article 1 Headline</a></h2>
  <p>This is a summary of article 1</p>
</div>
```

At the bottom, the status bar shows the element path: html.no-js body div.article h2 a.

```
1 from bs4 import BeautifulSoup
2 import requests
3
4 with open('simple.html') as html_file:
5     soup = BeautifulSoup(html_file, 'lxml')
6
7 article = soup.find('div', class_='article')
8 print(article)
9
```

Test Website

Article 1 Headline

This is a summary of article 1

Article 2 Headline

This is a summary of article 2

Footer Information

```
5 |     soup = BeautifulSoup(html_file, 'lxml')
6 |
7 | article = soup.find('div', class_='article')
8 | # print(article)
9 |
10| headline = article.h2.a.text
11| print(headline)
12|
13| summary = article.p.text
14| print(summary)
15|
```

The screenshot shows the 'Elements' tab of a browser's developer tools. The DOM tree on the left displays the following structure:

```
><head>...</head>
<body>
  <h1 id="site_title">Test Website</h1>
  <hr>
  <div class="article">
    <h2>
      <a href="article_1.html">Article 1 Headline</a> == $0
    </h2>
    <p>This is a summary of article 1</p>
  </div>
  <hr>
  <div class="article">
```

The 'article' element under the body has a blue border, indicating it is selected. The 'a' tag within the 'h2' element is also highlighted with a blue border. At the bottom of the developer tools, the 'Styles' tab is active, showing the CSS rule `element.style {`.

Article 1 Headline
This is a summary of article 1

All articles using find_all

Test Website

Article 1 Headline

This is a summary of article 1

Article 2 Headline

This is a summary of article 2

Footer Information

```
5     soup = BeautifulSoup(html_file, 'lxml')
6
7 for article in soup.find_all('div', class_='article'):
8     headline = article.h2.a.text
9     print(headline)
10
11    summary = article.p.text
12    print(summary)
13
14    print()
15
```

Article 1 Headline
This is a summary of article 1

Article 2 Headline
This is a summary of article 2

The screenshot shows the browser's developer tools with the 'Elements' tab selected. The DOM tree is displayed, starting with the <head> and <body> tags. Inside the body, there is a <h1> element with the id "site_title" containing the text "Test Website". Below it is an <hr> element. Then there are two <div> elements, both with the class "article". The first article contains an <h2> element with an anchor tag pointing to "article_1.html" and the text "Article 1 Headline". Below the h2 is a <p> element with the text "This is a summary of article 1". The second article follows a similar structure. At the bottom of the screenshot, the browser's address bar shows "article_1.html" and the status bar shows "html.no-js body div.article h2 a". The developer tools also show tabs for Styles, Computed, Event Listeners, DOM Breakpoints, and Properties.

Summary

- Introduced web scraping
- BeautifulSoup package installation
- Hands on exercise

Himanshu Patel, Instructor
Saskatchewan Polytechnic
email: patelh@saskpolytech.ca
Mining building, Saskatoon

