Task3:
Automatic login to a website and data parsing using the “Beautiful Soup” package

By Jawad Khalil,

Why we need to do this task:

Suppose we need to extract the specific data from a specific page which we get after logging in to a specific website.

- ❑ One can use the extracted data for many purposes for example, in **software development, statistics and Insights**, etc.
- ❑ But, here we are going to only **test** the Python code to parse some data.

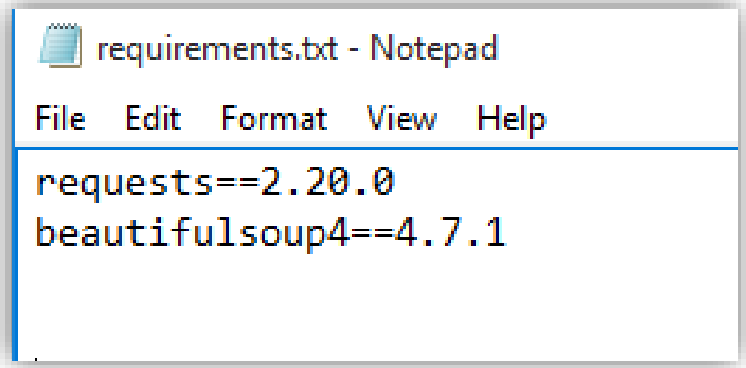
Before running the code:

- We will need a **website to log** in and parse the data. Here, we will parse the data from **https://the-internet.herokuapp.com** (**Note**: This is a dummy website and it needs a correct username and password to log in.)
- Before running the code, we must have installed the following two **packages**

1. **requests** (for pip users, run **"pip install requests"** in Terminal)
2. **beautifulsoup4** (for pip users, run **"pip install beautifulsoup4"** in Terminal.)

Alternatively, we can install the above packages with the **"requirement.txt"** file for this task by running the following command:

"pip install -r requirements.txt"



“requests”:

“requests” is an elegant and simple **HTTP library** for Python, built for human beings.

- **“requests”** allows us to send **HTTP** requests extremely easily
- The HTTP request returns a **Response Object** with all the response data (content, encoding, status, etc).
- It officially supports **Python 3.7+**.
- For the **documentation**, please visit:

https://requests.readthedocs.io/en/latest/

https://requests.readthedocs.io/en/latest/

“beautifulsoup4”:

BeautifulSoup is a library that makes it easy **to scrape** information from web pages. It creates a parse tree for parsed pages that can be used to extract data from **HTML**, which is useful for **web scraping**.

For the **documentation**, please visit:

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

“BeautifulSoup”:

- **“BeautifulSoup”** is a class in module **“bs4”**.
- It is a **data structure** representing a parsed **HTML** or **XML** document.
- Internally, this class defines the **basic interface** called by the tree builders when **converting an HTML/XML** document into a **data structure.**
- Most of the **methods** we call on a BeautifulSoup object are inherited from **PageElement** or **Tag**.

```
BeautifulSoup(markup='', features=None, builder=None, parse_only=None,
              from_encoding=None, exclude_encodings=None, element_classes=None, **kwargs)
```

Step1: Import the required packages

```
import requests
from bs4 import BeautifulSoup

```

The Difference between bs4 and beautifulsoup4:
"**bs4**" is technically a different package; however, it is a dummy package designed to install the correct package: "**beautifulsoup4**".

We can use either the **short name or long name** to install the “beautifulsoup4”.

Step2: Login Data

```
website_url = 'https://the-internet.herokuapp.com/'
login_url = "https://the-internet.herokuapp.com/authenticate"

login_data = {
    'username' : 'tomsmith' ,
    'password' : 'SuperSecretPassword!'
}

```

1. The "**login_url**" is the page we get after logging in to the required website.
Note: If one visits this URL by just copying it and pasting it in a browser he will get a "**Not Found**" page.

2. The "**login_data**" contains the required **username** and **password** that we need to pass to the above website to log in.

Step3: Start a session

```
with requests.session()as s:

```

By generating a “**session**” object upfront, we can reuse the session; this allows us to store **cookies**, for example, and **re-use** settings that will be utilized for all connections, such as headers and query parameters. Finally, sessions enable **connection pooling**, which allows us to reuse connections to the same host

Step4: Make a request to get a response

```
55    response = s.post(login_url, data=login_data)
56
```

The **post()** is a method that sends a **POST request** to a server.

The post() method is used when we need to send some data along with a request to **update** a response.

Here in this code, we are sending the "**login_data**" to a server to get some data back.

Step5: Call the BeautifulSoup() for the response and then inspect the HTML and text

```
soup = BeautifulSoup(response.content, "html.parser")
print("type(soup)=", type(soup))
```

Output: type(soup)= <class 'bs4.BeautifulSoup'>

Step5: Call the BeautifulSoup() for the response and then inspect the HTML and text

```
print("###### soup.prettify() started ##################################################################")
print(soup.prettify())
print("###### soup.prettify() ended ####################################################################")
```

Output:

```
###### soup.prettify() started ##################################################################
<!DOCTYPE html>
<!--[if IE 8]>         <html class="no-js lt-ie9" lang="en" > <![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js" lang="en">
 <!--<![endif]-->
 <html>
  <head>
   <script src="/js/vendor/298279967.js">
   </script>
   <meta charset="utf-8"/>
   <meta content="width=device-width" name="viewport"/>
   <title>
    The Internet
   </title>
   <link href="/css/app.css" rel="stylesheet"/>
   <link href="/css/font-awesome.css" rel="stylesheet"/>
   <script src="/js/vendor/jquery-1.11.3.min.js">
   </script>
   <script src="/js/vendor/jquery-ui-1.11.4/jquery-ui.js">
   </script>
   <script src="/js/foundation/foundation.js">
   </script>
   <script src="/js/foundation/foundation.alerts.js">
   </script>
   <script>
    $(document).foundation();
   </script>
  </head>
  <body>
   <div class="row">
    <div class="large-12 columns" id="flash-messages">
     <div class="flash success" data-alert="" id="flash">
      You logged into a secure area!
      <a class="close" href="#">
       ×
      </a>
     </div>
    </div>
   </div>
```

```
   <div class="row">
    <a href="https://github.com/tourdedave/the-internet">
     <img alt="Fork me on GitHub" src="/img/forkme_right_green_007200.png" style="position: absolute; top: 0; right: 0; border: 0;"/>
    </a>
    <div class="large-12 columns" id="content">
     <div class="example">
      <h2>
       <i class="icon-lock">
       </i>
       Secure Area
      </h2>
      <h4 class="subheader">
       Welcome to the Secure Area. When you are done click logout below.
      </h4>
      <a class="button secondary radius" href="/logout">
       <i class="icon-2x icon-signout">
        Logout
       </i>
      </a>
     </div>
    </div>
   </div>
   <div class="row" id="page-footer">
    <div class="large-4 large-centered columns">
     <hr/>
     <div style="text-align: center;">
      Powered by
      <a href="http://elementalselenium.com/" target="_blank">
       Elemental Selenium
      </a>
     </div>
    </div>
   </div>
  </body>
 </html>
</html>
###### soup.prettify() ended ####################################################################
```

Step5: Call the BeautifulSoup() for the response and then inspect the HTML and text

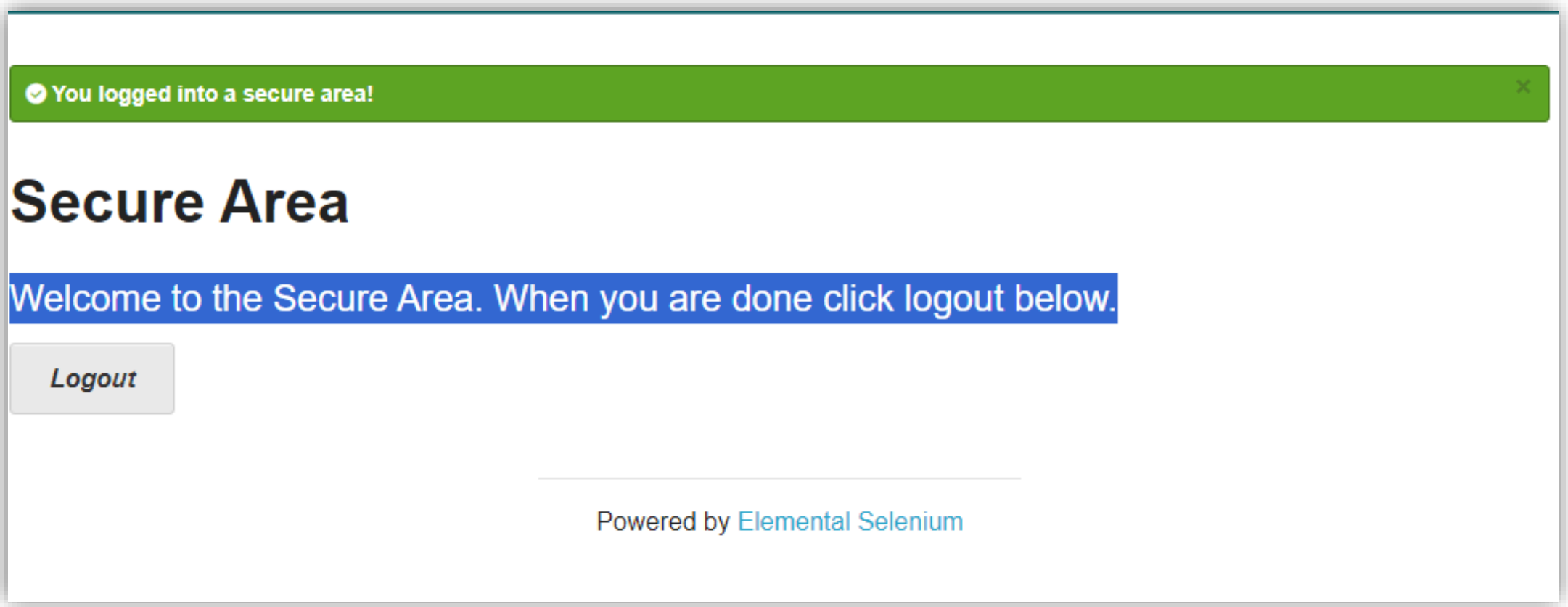
```
71
72    print("###### Soup.text started ##################################################################")
73    print(soup.text)
74    print("###### Soup.text ended ####################################################################")
75
```

Output:

```
Soup.text started ##########################################################################################

The Internet

    $(document).foundation();

        You logged into a secure area!
        ×

 Secure Area
Welcome to the Secure Area. When you are done click logout below.
 Logout

Powered by Elemental Selenium

Soup.text ended ############################################################################################
```

Step6: Get the required data by inspecting the HTML tree

The screenshot of the logged in page:



Task: We need to print the highlighted text

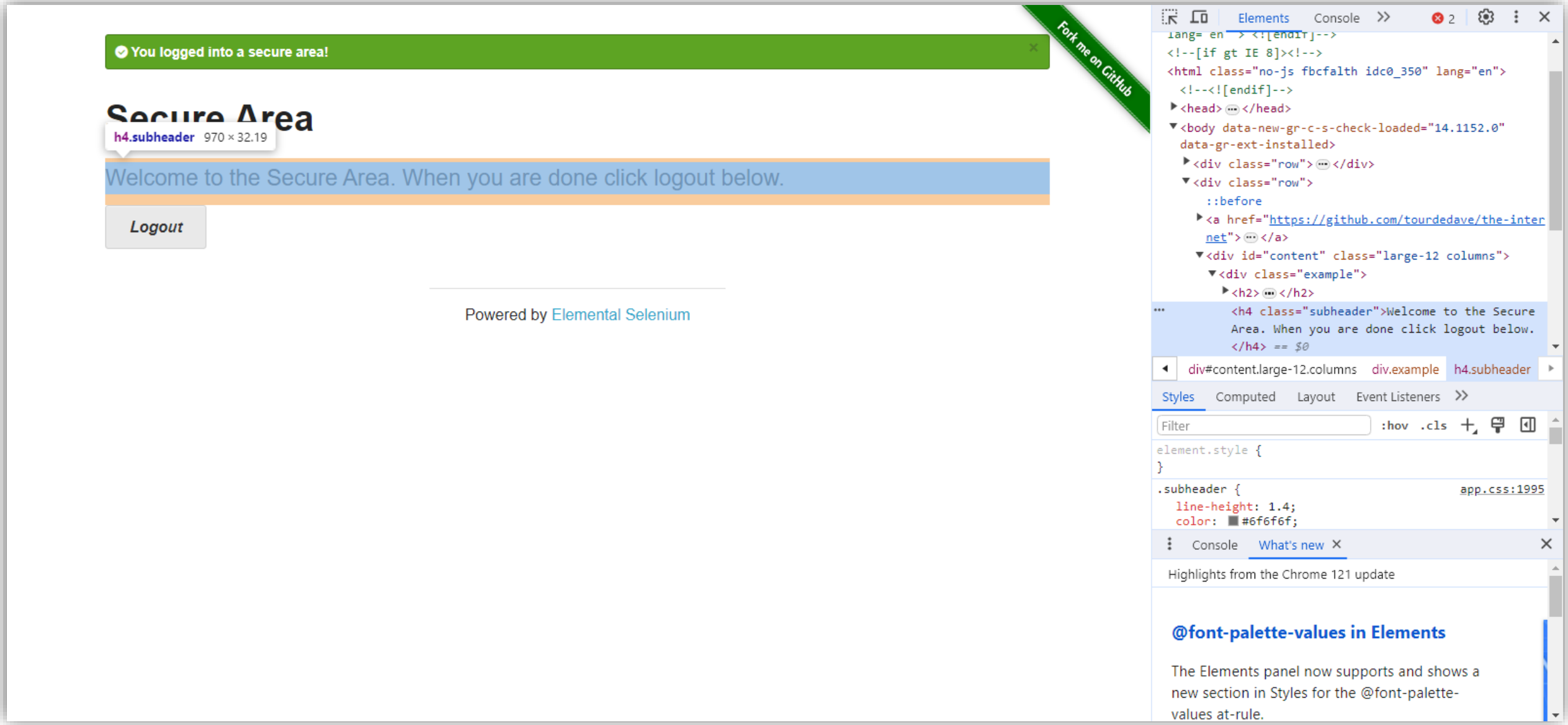
Step6: Get the required data by inspecting the HTML tree

To get the **position(element)** of the required data we need to inspect the login page. One can inspect the HTML tree in the terminal or output but…

```
  </head>
  <body>
   <div class="row">
    <div class="large-12 columns" id="flash-messages">
     <div class="flash success" data-alert="" id="flash">
      You logged into a secure area!
      <a class="close" href="#">
       ×
      </a>
     </div>
    </div>
   </div>
   <div class="row">
    <a href="https://github.com/tourdedave/the-internet">
     <img alt="Fork me on GitHub" src="/img/forkme_right_green_007200.png" style="position: absolut
    </a>
    <div class="large-12 columns" id="content">
     <div class="example">
      <h2>
       <i class="icon-lock">
       </i>
       Secure Area
      </h2>
      <h4 class="subheader">
       Welcome to the Secure Area. When you are done click logout below.
      </h4>
      <a class="button secondary radius" href="/logout">
       <i class="icon-2x icon-signout">
        Logout
       </i>
      </a>
     </div>
    </div>
   </div>
   <div class="row" id="page-footer">
    <div class="large-4 large-centered columns">
     <hr/>
     <div style="text-align: center;">
      Powered by
      <a href="http://elementalselenium.com/" target="_blank">
       Elemental Selenium
      </a>
     </div>
    </div>
   </div>
  </body>
 </html>
</html>
###### soup.prettify() ended ##################################################################
```

Step6: Get the required data by inspecting the HTML tree

the **easiest way** is to inspect the login page in a browser using the **developer's tools** (**inspect**)



Step6: Get the required data by inspecting the HTML tree

```
▼<div class="example">
  ▶<h2> ··· </h2>
***  <h4 class="subheader">Welcome to the Secure Area. When you are done click logout below.
     </h4> == $0
  ▶<a class="button secondary radius" href="/logout"> ··· </a>
```

One can clearly see that the required data is in the tag “**h4**” with **class=“subheader”**.
Now having this tag, we can write the **code** that will extract the required data.

Step6: Get the required data by inspecting the HTML tree

We can use **find()** or **find_all()** methods to find the required(unique) data from the html tree.

find(): It takes some arguments and returns the data according to the given criteria

```
find(name, attrs, recursive, string, **kwargs)
```

find_all(): It also takes some arguments and returns the data (**bs4.element.ResultSet or a list of results**)

The only **difference** between find() and find_all() is that find_all() returns a **list** containing a single result while find() just returns the **result.**

Step6: Get the required data by inspecting the HTML tree

Extracting the Required Data with find() :

```
required_data1 = soup.find("h4", class_="subheader").text
print('Required Data=', required_data1)

```

Output: Required Data= Welcome to the Secure Area. When you are done click logout below.

Here we used the find() function to look for the tag **“h4”** where **class=“subheader”**. **“.text”** returns only text from the tag “h4”.

```
▼<div class="example">
  ▶<h2> ··· </h2>
...  <h4 class="subheader">Welcome to the Secure Area. When you are done click logout below.
  </h4> == $0
  ▶<a class="button secondary radius" href="/logout"> ··· </a>
```

Step6: Get the required data by inspecting the HTML tree

Extracting the Required Data with find_all() :

Since there was only one “h4” tag with class=“subheader”, so we got the data by just using the **find().**

Now let’s assume there are more than one “h4” tag with class=“subheader”. If we use the find() function, it will return only the first “h4” tag with class=“subheader”. So, we can’t get the data from other same tags and classes with find() function.

In order to extract the data from the other same “h4” tag with same class we’ll use the **find_all()** function.

Let’s see an example in our case…

Step6: Get the required data by inspecting the HTML tree

Extracting the Required Data with find_all() :

```
required_data2 = soup.find_all("div", class_="row")
required_data2 = required_data2[1].find('h4', class_="subheader").text
print("Required Data=", required_data2)
```

```
<!DOCTYPE html>
<!--[if IE 8]>         <html class="no-js lt-ie9" lang="en" > <![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js rzvtxtnv idc0_350" lang="en">
  <!--<![endif]-->
▶ <head> ··· </head>
▼ <body data-new-gr-c-s-check-loaded="14.1152.0" data-gr-ext-installed>
  ▶ <div class="row"> ··· </div>
  ▼ <div class="row">
      ::before
    ▶ <a href="https://github.com/tourdedave/the-internet"> ··· </a>
    ▼ <div id="content" class="large-12 columns">
      ▼ <div class="example">
        ▶ <h2> ··· </h2>
···       <h4 class="subheader">Welcome to the Secure Area. When you are done click logout below.</h4> == $0
        ▶ <a class="button secondary radius" href="/logout"> ··· </a>
        </div>
      </div>
      ::after
    </div>
  ▶ <div id="page-footer" class="row"> ··· </div>
  </body>
▶ <grammarly-desktop-integration data-grammarly-shadow-root="true"> ··· </grammarly-desktop-integration>
</html>
```

Output:

Required Data= Welcome to the Secure Area. When you are done click logout below.

Step6: Get the required data by inspecting the HTML tree

Extracting the Required Data with **“attrs”(attribute)** search:

```
required_data3 = soup.find(attrs={"class": "subheader"}).text

```

Extracting the Required Data with **“tag”** and **“attrs”(attribute)** search:

```
required_data4 = soup.find("h4", attrs={"class":"subheader"}).text
```

```
required_data1 = soup.find("h4", class_="subheader").text
```

```
required_data2 = soup.find_all("div", class_="row")
required_data2 = required_data2[1].find('h4', class_="subheader").text
```

```

print(required_data1==required_data2==required_data3==required_data4)

```

Output: True

Thank you.