Webscraping with Python

Juan F. Imbet *Ph.D.*

Web scraping: Basics of HTML

HTML is a markup language that is used to create web pages. HTML is used to describe the structure of a web page. HTML is done using tags. Tags are used to describe the structure of a web page. Tags are enclosed in angle brackets. Tags can be nested inside other tags. Tags can have attributes. Attributes are used to describe the properties of a tag.

Web scraping: Basics of HTML (2)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>My First Heading</h1>
        <p>My first paragraph.</p>
    </body>
</html>
```

Web scraping: CSS

CSS is done using selectors. Selectors can be used to select elements by tag name, class, or id.

```
/* Select all elements */
* {
    color: red;
}
/* Select all h1 elements */
h1 {
    color: red;
}
/* Select all elements with class="example" */
.example {
    color: red;
}
/* Select the element with id="example" */
#example {
    color: red;
}
```

HTTP Requests

HTTP is a protocol that is used to send and receive data over the internet. HTTP is done using requests. Requests are sent to a server. Requests have a method and a URL. The method is used to specify the type of request. The URL is used to specify the location of the resource. The server responds with a response. The response has a status code and a body. The status code is used to specify the status of the request. The body is used to specify the data that was sent.

GET, POST, PUT, DELETE

- **GET** - Requests data from a specified resource
- **POST** - Submits data to be processed to a specified resource
- **PUT** - Updates a specified resource
- **DELETE** - Deletes a specified resource

Example: GET request

```
import requests
url = 'https://www.google.com'
response = requests.get(url)
print(response.status_code)
print(response.text)
```

```
200
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="es-419">
<head>...
```

Example Post request

```python
import requests
url = 'https://httpbin.org/post'
data = {'key1': 'value1', 'key2': 'value2'}
response = requests.post(url, data=data)
print(response.status_code)
print(response.text)
```

```
200
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "key1": "value1",
    "key2": "value2"
  },
...
}
```

Put and Delete requests

```
import requests
url = 'https://httpbin.org/put'
data = {'key1': 'value1', 'key2': 'value2'}
response = requests.put(url, data=data)
print(response.status_code)
print(response.text)
```

```
import requests
url = 'https://httpbin.org/delete'
response = requests.delete(url)
print(response.status_code)
print(response.text)
```

Good practice

Check the status code before parsing the response.

```
import requests
url = 'https://www.google.com'
response = requests.get(url)
if response.status_code == 200:
    print(response.text)
else:
    raise Exception(f"Error: {response.status_code}")
```

Headers

Not all websites allow you to communicate with them through a browserless environment. Some websites require you to specify a user agent. A user agent is a string that is used to identify the browser and operating system that is being used. A user agent can be specified using the headers argument.

```
import requests

url = 'http://sec.gov'
headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36'}
response = requests.get(url, headers=headers)
print(response.status_code)
```

Signing in

Some websites require you to sign in before you can access the data. You can sign in using the requests library. You can use the session object to keep track of the cookies. You can use the cookies argument to specify the cookies that you want to send.

```python
import requests

# some website that requires username and password
url = 'http://httpbin.org/basic-auth/user/passwd'

# create session object
session = requests.Session(
# sign in
session.auth = ('user', 'passwd')
# send request
response = session.get(url)
# print response
print(response.status_code)
print(response.text)
```

```
200
{
  "authenticated": true,
  "user": "user"
}
```

Ok, so we get all the HTML, now what?

We need to parse the HTML to extract the data that we want. We can use the BeautifulSoup library to parse the HTML.

```
import requests
from bs4 import BeautifulSoup as bs

url = 'https://www.google.com'
response = requests.get(url)
soup = bs(response.text, 'html.parser')
print(soup.prettify())
```

HTML as a tree

```
<div id="guser" width="100%">
    <nobr>
     <span class="gbi" id="gbn">
     </span>
     <span class="gbf" id="gbf">
     </span>
     <span id="gbe">
     </span>
     <a class="gb4" href="http://www.google.com.co/history/optout?hl=es-419">
      ...
```

My take on how to traverse the tree

Use the `find_all` method, and specify the tag name, class, or id.

```
soup = bs(response.text, 'html.parser')
soup.find_all('a')
soup.find_all('a', class_='gb4')
soup.find_all('a', id='gb_70')
```

Recall that `find_all` always returns a list, even if there is only one element.