Section 13

Web Scraping

Mohammed Asir Shahid

2021-08-05

Contents

1 The webbrowser Module

```
import webbrowser

webbrowser.open("https://asir.dev")
```

Let's create a program that can open a given address on maps.

```
import webbrowser, sys, pyperclip

sys.argv # ["mapit.py", "870", "Valencia", "St."]

# Check if command line arguments were passed
```

```
if len(sys.argv) > 1:
    # ["mapit.py", "870", "Valencia", "St."] -> 870 Valencia St.
    address=" ".join(sys.argv[1:])
else:
    address=pyperclip.paste()

webbrowser.open("https://www.google.com/maps/place/%s" % (address))
```

2 Downloading from the Web with the Requests Module

The requests module lets you easily download files from the web without having to worry about complicated network issues. The requests module is a third party module which we'll need to install on our own.

```
pip install requests
```

We can pass a URL to the requests.get() function in order to get the file. We can check the status code to see if it downloaded properly, if so then we'll get the status code 200. We can print out the file using .text. We can also see if there is an issue by calling the $\text{raise}_{\text{forstatus}}()$ method which will raise an error if we ran into any problems.

```
import requests

res=requests.get("http://automatetheboringstuff.com/files/rj.txt")

print(res.status_code)

print(len(res.text))
print(res.text[:500])

print(res.raise_for_status())

badRes=requests.get("http://automatetheboringstuff.com/files/rjuliet.txt")

print(badRes.raise_for_status())
```

2.1 Write-binary mode: open(filename, "wb")

We can save a web page to a file using the open function. However, we must do somethings differently.

```
import requests

res=requests.get("http://automatetheboringstuff.com/files/rj.txt")
playFile=open("RomeoAndJuliet.txt","wb")

for chunk in res.iter_content(100000):
    playFile.write(chunk)

playFile.close()
```

Request module functions can be useful, but they are somewhat limited. You can only use it when you have the exact URL that you need to download. Selenium lets your Python scripts control the web browser directly.

3 Parsing HTML with the Beautiful Soup Module

Here we will learn how to write programs that pull information off of web pages. This is known as web scraping. We have a third party module called beautifulsoup which makes parsing through websites HTML much easier.

```
pip install beautifulsoup4
```

```
import bs4
```

Let's try to parse through an eBay page and scrape the price information from that page.

```
import bs4,requests

url="https://www.ebay.com/itm/313628358864"
```

```
def geteBayPrice(productURL):
    response = requests.get(url)

    response.raise_for_status()

    soup=bs4.BeautifulSoup(response.text,"html.parser")

    elems=soup.select("#prcIsum")

    return elems[0].text.strip("US").strip()

url="https://www.ebay.com/itm/313628358864"

price=geteBayPrice(url)

print("The price is %s" % (price))


The price is $20.51
```

This code is pretty reliant on the website's CSS staying the same.

```
import bs4,requests


def getTemperature(url):
    response = requests.get(url)

    response.raise_for_status()

    soup=bs4.BeautifulSoup(response.text,"html.parser")

    elems=soup.select("#current_conditions-summary > p.myforecast-current-lrg")


    return elems[0].text.strip()

url="https://forecast.weather.gov/MapClick.php?lat=40.8667&lon=-73.0343"
```

```
temp=getTemperature(url)

print("The temperature is %s" % (temp))
```

```
The temperature is 76°F
```

4 Controlling the Browser with the Selenium Module

We have learned how to download webpages and parse their HTML. However, sometimes the webpages we want to access require logging in or relying on Javascript. In these cases, we can use Selenium which will open a programatically controlled webbrowser. It's a third party module that we need to install ourselves.

```
pip install selenium
```

```
from selenium import webdriver

browser=webdriver.Firefox()

browser.get("https://automatetheboringstuff.com")

elem=browser.find_element_by_css_selector(".main > div:nth-child(1) > ul:nth-child(21)

elem.click()

elems=browser.find_elements_by_css_selector('p')
print(len(elems))
```

```
109
```

Selenium has several methods of getting web elements from the page. The most commonly used ones are $\text{find}_{\text{element by css selector}}$ and $\text{find}_{\text{elements by css selector}}$. However, we can also find by class name, id, link text, partial text, and tag name.

```
from selenium import webdriver

browser=webdriver.Firefox()

browser.get("https://asir.dev")

postElem=browser.find_element_by_css_selector("#blog-link")

postElem.click()

browser.back()
browser.forward()
browser.refresh()
#browser.quit()

searchElem=browser.find_element_by_css_selector("#search-box")

search="Post"

searchElem.send_keys(search)
searchElem.submit()
```

Now we can take a look at how Python scripts can use Selenium to read webpages.

```
from selenium import webdriver

browser=webdriver.Firefox()

browser.get("https://automatetheboringstuff.com")
elem=browser.find_element_by_css_selector(".main > div:nth-child(1) > blockquote:nth-ch
print(elem.text)

elem=browser.find_element_by_css_selector("html")
print(elem.text)
```