Selenium și BeautifulSoup - web scraping

Web Scraping este un proces prin care se poate accesa/seta informație existența pe o pagina web într-o formă cât mai convenabila. Trebuie specificat ca în ziua de azi paginile web sunt formate de o succesiune de tehnologii cum ar fi HTML, CSS, JS etc.
Astfel problema devine complexa deoarece pagina pe care noi putem sa o vedem într-un browser nu mai reprezintă ce văd în codul HTML, conținutul ar putea fi alterat de JS. Din acest motiv în unele cazuri exista necesitatea de a rula un browser virtual, în alte dăți avem nevoie de ceva cât mai rapid cum ar fi requests.
Prin urmare, se evidențiază două cazuri: extragerea de informații prin BeautifulSoup si extragerea de informații folosind Selenium.

BeautifulSoup

BeautifulSoup nu este un modul standard. De asemenea acesta folosește în fundal și un engine de parsare a codului html. Pentru aceasta parsare vom utiliza lxml. Vă rugăm sa instalati ambele module.

pip install BeautifulSoup4

pip install lxml

Utilizarea presupune trimiterea unei solicitări get() și parsarea șirului de caractere returnat.
Iată un program care arata o posibila utilizare a BeautifulSoup.

```
import requests
from bs4 import BeautifulSoup

r=requests.get("http://www.lpf.ro/clasament-liga-1")
s1=BeautifulSoup(r.text,'lxml')

for rand in s1.table.findAll('tr'):
    for camp in rand.findAll('td'):
            try:
                    print str(camp.stripped_strings.next()).ljust(16),
            except:
                    pass
    print


raw_input("\n\nApasa enter sa iesi")
```

Fig.1

```
#   Echipa           M   V   E   I   Golaveraj   Puncte
1   FCSB             7   4   2   1   10-5        38
2   F.C. Viitorul    7   2   3   2   7-6         35
3   Dinamo Bucuresti 7   3   3   1   9-6         33
4   CFR 1907 Cluj    7   3   2   2   8-7         33
5   CS U Craiova     7   1   3   3   5-10        28
6   Astra Giurgiu    7   1   1   5   7-12        26

Apasa enter sa iesi
```

Fig 2

În figura de mai sus se poate vedea extragerea primului tabel afisat pe pagina lpf.ro. Modulul request are rolul de a captura pagina get. BeautifulSoup va parsa textul primi în vederea gasirii de informație.

În html un tabel este format dintr-un tag de tip <table></table>, care are în interior taguri de tip <tr></tr> (tr provine de la table row adică un rând, o inregistrare în tabel). În interiorul tag-urilor <tr></tr> se regasesc intrari cu tag-ul <td></td> care include un câmp al tabelului.

Prin umare, după obtinerea unui tabel ne dorim sa găsim toate liniile din tabel. Facem asta prin apelul s1.table.findAll('tr'). De pe fiecare rând ne dorim sa extragem fiecare câmp și realizam asta cu o căutare pe fiecare rând aplicata tot prin metoda findAll astfel rand.findAll('td').

Dacă am afisa fiecare câmp dintr-un rând atunci un posibil rând ar fi:

```
<td>1</td>
<td><img alt="FCSB" src="http://www.lpf.ro/public/images/teams/20x20/steaua.png"
/><a class="clicklink" href="http://www.lpf.ro/club/steaua-bucuresti">FCSB</a></
td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>10-5</td>
<td>38</td>
```

Fig.3

Prin urmare trebuie sa aplicam anumite metode de a elimina tag-ul <td> dar și alte tag-uri cum ar fi <a> sau <img>. Fapt de implica multă munca manuala deoarece nu știm ce alte tag-uri sunt incluse în tag-ul <td>

Astfel ajungem sa folosim camp.stripped_strings.next() care va elimina text-ul neutilizat. Metoda stripped_strings returnează un generator care permite extragerea informației prin metoda .next().

O alta metoda de a extrage informația este folosind atributul text astfel: camp.text

Diferenta este data de alegerea unui encoding. Metoda camp.stripped_strings.next() nu formateaza dupa un encoding, pe cand metoda camp.text va folosi encoding-ul paginii web.

Următorul exemplu se dorește a fi mai explicit cu fiecare pas în parte.

Dar înainte de asta trebuie sa discutam despre sintaxa s.table.findAll(). Întrebarea este de ce s1.table?

BeautifulSoap permite o structura arborescenta la nivel de atribut.

Astfel variabila s reprezintă obiectul root al paginii parsate prin care putem extrage elementele. Astfel dacă dorim titlul paginii putem apela s1.title. Toate obiectele se regăsesc sub numele generic de descendenți.

```
>>> s1.title
<title>Clasament Liga 1</title>
>>> s1.descendants
<generator object descendants at 0x0000000005AD9168>
>>> len(list(s1.descendants))
6066
```

Fig.4

Spre exemplu pagina web lpf.ro are 6066 de descendenți. Totuși utilizarea unei structuri arborescente implica, din păcate, limitări date de duplicarea tipului de descendenți. Spre exemplu, pagina lpf.ro are mai multe tabele. Prin apelarea s1.table se va returna primul tabel.

Sa modificam și în primul program astfel încât sa extragem tabelul „Clasament Liga 1" deoarece în progrmaul anterior am extras „Clasament Liga 1 – Play-off". Astfel se poate vedea ca la aplicarea metodei findAll() indicam clasa „echipe".

```
import requests
from bs4 import BeautifulSoup

r=requests.get("http://www.lpf.ro/clasament-liga-1")
s1=BeautifulSoup(r.text,'lxml')
tabel=s1.findAll('div',{"class":"echipe"})
for rand in tabel[0].findAll("tr"):
    for camp in rand.findAll('td'):
            try:
                    print str(camp.stripped_strings.next()).ljust(12),
            except:
                    pass
    print


raw_input("\n\nApasa enter sa iesi")
```

Fig.7

#	Echipa	M	V	E	I	Golaveraj	Puncte
1	F.C. Viitorul	26	16	3	7	39-22	51
2	FCSB	26	13	8	5	34-22	47
3	Astra Giurgiu	26	13	5	8	32-28	44
4	CS U Craiova	26	13	4	9	36-26	43
5	CFR 1907 Cluj	26	14	7	5	42-23	43*
6	Dinamo Bucuresti	26	12	5	9	40-33	41
7	Gaz Metan Medias	26	10	9	7	36-27	39

Fig.8

Selenium și web Testing

Selenium este gândit într-un mod diferit deoarece permite rularea JavaScript-urilor. De asemenea, se poate integra cu diferite WebDriver cum ar fi :Firefox Driver (controlezi o sesiune web prin firefox), Chrome Driver, Internet Explorer Driver, Opera Driver, iOS Driver sau Android Driver. Astfel se poate vorbi de web automation în sensul ca se pot realiza teste ale interfețelor grafice plătite cum ar fi

pagini web embedded pentru managementul dispozitivelor (router wifi, switch, firewall, etc). Selelnium se împarte în doua sectiuni: Selenium și Selenium Framework. Selenium (numit și 1.0) reprezintă accesul prin Selenium la anumite pagini. Selenium Framework reprezintă folosirea unui framework în vederea automatizarii accesului prin mici variatiuni.
Se poate alege driver-ul care va ajuta cel mai bine. Codul rămâne neschimbat. Doar indicarea driver-ului se schimba.
Deoarece selenium nu este instalat default trebuie sa instalam urmatoarele:
pip install selenium
pip install chrome-driver
pip install chromedriver_installer
Apoi trebuie să verificam dacă în directorul C:\Python27\scripts exista fisierul chromedriver.exe deoarece aceasta cale ne este solicitata (calea completă pana la fisier).
Sa trecem la treaba și sa arătam un exemplu.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time


user="slenium.test@gmx.com"
passw="oooooo"

page= 'https://www.gmx.com/'


chromedriver = 'C:\\Python27\\scripts\\chromedriver.exe'
browser = webdriver.Chrome(chromedriver)
browser.get(page)
time.sleep(1)

fereastra=browser.find_element_by_xpath('/html/body/div[1]/div/div[1]/header/div/div[2]/a[2]')
fereastra.click()

time.sleep(1)

username=browser.find_element_by_xpath('/html/body/div[1]/div/div[1]/div/div/div[2]/form/div[1]/input')
username.send_keys(user)

password=browser.find_element_by_xpath('/html/body/div[1]/div/div[1]/div/div/div[2]/form/div[2]/input')
password.send_keys(passw)

login_attempt = browser.find_element_by_xpath('/html/body/div[1]/div/div[1]/div/div/div[2]/form/button')
login_attempt.submit()

time.sleep(1)

email = browser.find_element_by_name("core_mail")
email.click()

#browser.close()
```

Fig.1

Rularea va duce la deschiderea unei sesiuni, logarea și accesarea unei anumite sectiuni din site.
Obiectul browser are multiple metode interesante de discutat:

browser.get(„pagina”) - permite accesarea unei pagini.
browser.find_element_by_xpath(„xpath”) - permite accesarea unui tag dupa xpath. Returneaza un obiect similar cu browser.
browser.find_element_by_id(„id”) - permite accesarea unui tag după id. Returneaza un obiect similar cu browser.

obiect.send_keys(„sir”) - trimite un sir într-un form de post
obiect.click() - face click pe acel tag-urilor
obiect.submit() – legat de un post - trimite info
browser.close() - închide browser
obiect.text afiseaza textul asociat tag-ului.

Pagina prezentata nu se poate accesa prin BautifulSoup deoarece are anumite JS-uri active pentru logare și numai.