UNIVERZA V LJUBLJANI
NARAVOSLOVNOTEHNIŠKA FAKULTETA
ODDELEK ZA TEKSTILSTVO, GRAFIKO IN OBLIKOVANJE

Spletno luščenje podatkov z uporabo Pythona in knjižnice BeautifulSoup

Seminarska naloga

Ime in priimek: Alena Selimović
Predmet: interaktivni mediji
Smer in letnik študija: GMT-I, 2. letnik
Šolsko leto: 2016/2017

Ljubljana, 17.2.2017

Kazalo

1. UVOD

Za temo seminarske naloge sem si izbrala področje spletnega luščenja podatkov. Najprej sem na kratko opredelila pojme semantični splet, HTML in XML, saj je poznavanje le teh pomembno za razumevanje postopka pridobivanja podatkov iz spletnih dokumentov. V nadaljevanju sem razložila kaj sploh je luščenje podatkov in nato na kratko predstavila DOM, ki se navezuje na eno od tehnik pridobivanja podatkov s spleta. Dotaknila sem se tudi zakonodaje spletnega luščenja podatkov, saj se mi zdi pomembno poznati pravila, ki določajo, v katerih primerih je luščenje podatkov dovoljeno in pod kakšnimi pogoji.

Sledi predstavitev modula urllib in knjižnice BeautifulSoup, ki sem ju uporabila pri izdelavi dveh programov, in sicer za pridobitev spletnega naslova in HTML zapisa spletne strani, ki sem jo uporabila za luščenje podatkov.

V zadnjem delu seminarske naloge sem opredelila cilje pri izdelavi obeh programov, postopoma razložila korake izdelave in nato priložila še celotno kodo obeh programov. Na koncu sem navedla eno od omejitev knjižnice BeautifulSoup, na katero sem naletela med pisanjem programov.

2. SEMANTIČNI SPLET

Svetovni splet (World Wide Web) je množica dokumentov in drugih virov, ki so med seboj povezani s hiperpovezavami (hyperlinks) in spletnimi naslovi oz. url-ji. Vloga računalnikov je dostavljanje in predstavitev teh virov, kar nam omogoča dostop do številnih podatkov in informacij, ki se nahajajo na spletu. Podatki so predstavljeni v obliki ljudem berljivih dokumentov, grafičnih predstavitev, ipd. Vendar pa računalniki sami ne poznajo pomena teh virov. Rezultat našega iskanja po spletu ni odgovor na to, kar iščemo, ampak seznam različnih povezav, na katerih se odgovor lahko nahaja. Če pa bi računalniki poznali pomen virov na spletu, bi lahko lažje in bolj pravilno posredovali želene informacije.

Izraz semantični splet se je pojavil že konec 90-ih let prejšnjega stoletja, in sicer s strani organizacije W3C (World Wide Web Consortium). Osnovna ideja semantičnega spleta je izboljšati trenutni svetovni splet tako, da bodo dokumenti povezani na način, ki je razumljiv tudi strojem. Semantični splet bi predstavljal globalno podatkovno bazo, podatke iz te baze pa bi lahko uporabljali tako ljudje kot tudi računalniki - ti bi z lažjim in hitrejšim procesiranjem, bolj natančno interpretacijo in lažjim povezovanjem podatkov na spletu, sodelovali in pomagali ljudem pri iskanju zahtevanih informacij.

Razvoj semantičnega spleta za zdaj poteka počasneje kot so si obetali. Največji problem pri razvoju semantičnega spleta leži v razvoju strojnega učenja, kot je npr. področje računalniškega vida in razvoju umetne inteligence.

3. HTML IN XML

3.1. HTML

HTML (Hypertext Markup Language) je hipertekstovni označevalni jezik - je format podatkov, ki se uporablja za ustvarjanje HTML dokumentov in določa skladnjo označevanja metabesednih elementov. Celoten nabor značk v HTML besedilu nam omogoča:

- objavljanje elektronskih dokumentov z naslovi, podnaslovi, tabelami, seznami, fotografijami, multimedijsko vsebino, ipd.,
- pridobivanje spletnih informacij prek spletnih povezav,

- postavljanje vnosnih polj oz. obrazcev za upravljanje z oddaljenimi storitvami in uporabo pri naročilih.

HTML je bil razvit leta 1991 z namenom, da bi lahko elektronsko povezali dokumente s hiperpovezavami in tako dobili mrežo med seboj povezanih dokumentov. Razvil ga je fizik Tim Berners-Lee, ki je takrat delal pri CERN-u. Vizija pri nadaljnjem razvoju HTML jezika je bila, da bi lahko vse naprave učinkovito uporabljale standard za pridobivanje informacij na spletu - računalniki z različnimi operacijskimi sistemi, z različnimi arhitekturami, mobilni telefoni, tablični računalniki in danes vedno bolj tudi druge pametne naprave.

V osnovi je na sintakso HTML jezika močno vplival jezik SGML (standardizirani splošni označevalni jezik), kasneje pa se je HTML razvijal ločeno. Po prvi verziji HTML so sledile druge - verzija HTML 2.0 je standardizirala tabele in obrazce, ter ga nadgradila z izboljšavami. Nato so sledile verzije 3.2, 4.0 in 4.01, ki so vključevale še dodaten nabor značk. Danes se uporablja predvsem HTML 5.0, ki poleg zgradbe dokumenta določa tudi semantični pomen posameznih značk in delov dokumenta in je zmanjšal potrebo po zunanjih vtičnikih (kot je npr. Flash).

3.2. XML

XML (Extensible Markup Language) je razširljivi označevalni jezik za dokumente, ki vsebujejo strukturirane informacije. Definira standardni način za dodajanje oznak k dokumentom; je meta jezik za opisovanje označevalnih jezikov. Pri XML specifikaciji nabor in semantika značk nista fiksna, kot pri HTML jeziku. Pri HTML specifikaciji je nabor značk odvisen od proizvajalcev spletnih brskalnikov, pomembna pa je tudi združljivost s starejšimi sistemi, zato je vpeljava novih značk počasen in nadzorovan proces. Pri XML specifikaciji pa tega problema ni, saj ne določa ne nabora ne semantike značk. XML jezik je možno tudi razširiti, saj si lahko sami izmislimo imena značk. Ker pri XML specifikaciji ni določenega nabora značk, je semantika posledično odvisna od aplikacije, ki procesira XML dokument ali od slogovnih datotek.

Najbolj znane XML sheme so:
- KML (Keyhole Markup Language), ki se uporablja pri geografskem prikazu podatkov,

- XUL (XML User Interface Language), ki ga uporablja predvsem fundacija Mozilla za izgradnjo uporabniških vmesnikov,
- RDF (Resource Description Framework), ki se uporablja za opis meta podatkov,
- SVG, ki opisuje strukturo vektorskih slik.

4. LUŠČENJE PODATKOV

Na spletu se nahaja velika količina podatkov, ki so shranjeni v podatkovnih bazah. Viri na spletu so večinoma v obliki HTML in XML dokumentov in predstavljajo delno strukturiran vir podatkov. Te dokumente računalnik razume in je sposoben iz njih programsko pridobiti podatke. V nekaterih primerih pa so podatki, ki so namenjeni uporabnikom in deljenju med njimi, prikazani na načine, ki računalnikom omejujejo njihovo procesiranje (npr. v obliki fotografij in grafik) ali pa so spletne strani namenoma izdelane na tak način, da je do njih računalniško težko dostopati.

Spletno luščenje je proces, ki obsega iskanje in pridobivanje podatkov v delno strukturiranih dokumentih na spletu. Pridobljene podatke lahko nato uporabimo v druge namene. Računalnik omogoča hiter dostop do podatkov in lahko obdela veliko količino le teh, s programi za luščenje podatkov pa je možno te podatke tudi povezati med seboj in s tem pridobiti nove informacije, ki bi bile brez pomoči računalnika težko dostopne. Z integracijo podatkov lahko tako na enem mestu dobimo podatke kot so npr. ime, telefonska številka, naslov, zaposlitev, mnenje uporabnika, ipd. Računalnik olajša delo dolgotrajnega iskanja, prihrani čas in ponudi bolj uporabne informacije.

Problem, ki se pojavlja pri luščenju podatkov s spleta je predvsem ta, da so spletni brskalniki namenjeni prikazovanju dokumentov ljudem. Zato so lahko dokumenti, ki so za ljudi razumljivi, za računalnik včasih velika neznanka. Poleg tega so brskalniki pri prikazovanju dokumentov, ki se ne držijo konvencij in pravil pisanja, popustljivi. Za računalnik pa to predstavlja problem, s tem pa spletno luščenje pogosto postane težavno ali celo nemogoče.

Nameni luščenja so različni. Večinoma se luščenje uporablja pri analizi različnih podatkov in raziskavah. Pogosta je uporaba luščenja pri izdelavi spletnih pajkov, ki se sprehajajo po spletu in ga indeksirajo. Včasih je izdelava brskalnikov potekala ročno, zdaj pa s pomočjo spletnega luščenja računalniki sami indeksirajo splet in s pomočjo algoritmov uporabnikom ponudijo

možnost iskanja po njem. Spletno luščenje se s simulacijo uporabnika uporablja tudi kot eden izmed načinov avtomatičnega testiranja spletnih aplikacij.

4.1. DOM in spletno luščenje podatkov

Document Object Model (DOM) je specifikacija, ki neodvisno od programskega jezika in platforme definira, kako so podatki prikazani, kako do njih dostopati in kako z njimi manipulirati. Nastal je kot specifikacija, ki dovoli programom, napisanih v jezikih JavaScript in Java, prenosljivost med spletnimi brskalniki.
DOM je načrtovan v tradicionalno objektno usmerjenem smislu. To pomeni, da so dokumenti obravnavani kot objekti, model DOM pa ne definira samo strukture dokumenta, ampak tudi logiko dokumenta in njegovih gradnikov. DOM specificira:
- vmesnike in objekte, ki predstavljajo dokument in ga spreminjajo,
- obnašanje in lastnosti objektov in vmesnikov (semantiko),
- razmerje in sodelovanje med objekti in vmesniki.

S specifikacijo DOM lahko spreminjamo strukturo dokumenta, mu dodajamo ali brišemo vsebino. Dokumenti vmesnika DOM imajo logično strukturo, ki je podobna drevesni strukturi - vsak dokument vsebuje korensko vozlišče (služi kot koren drevesa elementov v dokumentu), korensko vozlišče vsebuje druga vozlišča (nodes), ta pa lahko vsebujejo še dodatna vozlišča. DOM s tako drevesno strukturo združuje množico gradnikov (objektov) in opisuje način sprehajanja po dokumentu.

Pri luščenju podatkov iz ciljne spletne strani lahko s pomočjo drevesne strukture DOM programsko pridemo do dela spletne strani, ki nas zanima, ter iz njega izluščimo želene oz. potrebne podatke.

4.2. Pravni vidik luščenja podatkov

Zakonodaja glede luščenja podatkov tako v tujini kot v Sloveniji, ni povsem jasna. Spletne strani pogosto ne dovoljujejo luščenja podatkov ali pa ga omejijo in definirajo pogoje uporabe pridobljenih podatkov. Spletne strani navadno vsebujejo robots.txt datoteko, kjer so navedene omejitve dostopa do podatkov. Nekatere rešitve, ki se uporabljajo za preprečevanje luščenja podatkov so:

- blokiranje IP naslova,
- onemogočitev spletnih API-jev,
- blokiranje na podlagi spremljanja prevelike uporabe spletne strani,
- detekcija s pastmi za robote, t.i. honeypot - npr. namenske strani, do katerih roboti ne smejo dostopati, če tega pravila ne upoštevajo in kljub temu dostopajo do podatkov na strani, pa se pri tem identificirajo in pustijo svoj IP naslov, na podlagi katerega jih je mogoče v prihodnje blokirati.

Pomembno je tudi v kakšen namen se uporabijo pridobljeni podatki. Lastniki spletnih strani kot so npr. spletni telefonski imeniki, imeniki podjetij, ipd. ne želijo, da bi se podatki, pridobljeni z njihovih strani, uporabljali za analizo uporabnikov in razkrivanje informacij o njih, saj bi to sprožilo nezadovoljstvo z njihovo storitvijo in lahko tudi zahteve za umik nekaterih podatkov. Neprimerno je tudi npr. pridobivanje e-poštnih naslovov uporabnikov in nato pošiljanje neželene e-pošte oz. spama na njihove e-poštne naslove.

Pri pridobivanju podatkov sta zato vedno potrebna poznavanje pogojev in previdnost, pri prikazu podatkov pa je potrebno jasno navesti, da podatki niso naši in navesti njihov vir.

5. MODUL URLLIB

Modul definira določeno število funkcij in razredov, ki se večkrat uporabljajo pri pisanju različnih programov. Na ta način jih ni potrebno vsakič znova pisati od začetka, ampak lahko v Python preprosto uvozimo modul, ki te funkcije že vsebuje.
Urllib je modul, ki se uporablja za pridobivanje podatkov s spletnih strani. V starejših verzijah Pythona se je uporabljal modul urllib2, v Pythonu 3 pa je bil ta modul razdeljen na tri dele, imenovane:
- urllib.request
- urllib.parse
- urllib.error.

Modul urllib je del Pythona, uvozimo ga z ukazom import urllib.request oz. v starejših verzijah Pythona z ukazom import urllib2.

Ko se v Pythonu želimo povezati na nek spletni strežnik lahko uporabimo funkcijo urlopen(). Ta nam omogoča, da se povežemo na nek spletni naslov oz. url (torej na strežnik) ter z njega dobimo HTML zapis spletne strani, ki se nahaja na tem spletnem naslovu.

Funkcija urlopen() znotraj oklepajev kot argument sprejme naslov spletnega mesta oz. url (Uniform Resource Locator). V starejših verzijah se uporablja ukaz urllib2.urlopen(), v Pythonu 3 pa urllib.request.urlopen(). Funkcija urlopen() lahko pridobi url naslove, ki uporabljajo različne protokole (ti določajo način, kako se sporazumevata spletni strežnik in brskalnik) - FTP (File Transfer Protocol), HTTP (Hyper-Text Transfer Protocol) in HTTPS (Hyper-Text Transfer Protocol Secure). Najbolj običajen je protokol HTTP.

HTTP je protokol za prenos hiperteksta in temelji na ciklu poizvedbe in odziva (request-response cycle) - uporabnik poda zahtevo za dostop do določenega strežnika, strežnik pa se odzove in prikaže želeno spletno stran. Ukaz urllib.request posnema ta cikel, tako da z njegovo uporabo podamo zahtevo po določenem url naslovu, ko pokličemo funkcijo urlopen() pa kot odziv dobimo želeni url. Odziv, ki ga vrne funkcija urlopen(), je objekt. Pri nadaljnji uporabi lahko na njem kličemo druge funkcije.

6. KNJIŽNICA BEAUTIFULSOUP

Knjižnica je skupek kode, ki je bila ustvarjena za določen namen. Knjižnica BeautifulSoup je bila ustvarjena z namenom luščenja podatkov iz HTML in XML dokumentov. Avtor knjižnice je Leonard Richardson.
Knjižnico BeautifulSoup je potrebno najprej namestiti na računalnik in jo nato uvoziti v Python z ukazom:
from bs4 import BeautifulSoup

Pri uporabi Pythona 3 se uporablja knjižnico BeautifulSoup 4 (bs4), pri starejših verzijah Pythona pa starejše verzije knjižnice BeautifulSoup.

BeautifulSoup omogoča:
- uporabo že napisanih funkcij in metod za navigacijo, iskanje in spreminjanje HTML zapisa ter pridobivanje natančno določenih podatkov (razčlenitev HTML-ja oz. HTML parsing) - na

ta način v program napišemo veliko manj kode, kot če bi se lotili luščenja podatkov brez uporabe knjižnice in tako prihranimo precej časa,
- avtomatično pretvori dokumente, ki jih želimo pridobiti, v Unicode zapis, tako da nam ni potrebno skrbeti glede kodiranja.

Knjižnica BeautifulSoup torej omogoči, da iz HTML dokumenta pridobimo točno določene podatke, ki nas zanimajo. To so lahko npr. vse povezave v dokumentu, vsi naslovi, vsi naslovi tabel itd., ali pa natančno določene povezave, naslovi, slike, tekst, itd.

Zanimivost: Ime BeautifulSoup je referenca na otroško pravljico, v kateri lačni tujci prepričajo prebivalce mesta, naj vsak prispeva majhen delež svoje hrane, da bi iz njih skupaj pripravili obrok, ki bi lahko nahranil vse. Konvencije in pravila pisanja HTML dokumentov pogosto niso upoštevana, npr. ni indentacije, pojavljajo se izpuščena podpičja, ipd. (brskalniki dokumente kljub temu večinoma pravilno razumejo). BeautifulSoup pa vzame tak neurejen HTML in ga spremeni v bolj primeren zapis ter s tem omogoči lažjo navigacijo po dokumentu in posledično lažje luščenje podatkov.

7. PRIMER

7.1. Cilj programov

Za izvedbo naloge sem ustvarila enostavno spletno stran, ki ne vsebuje robots.txt datoteke.
Povezava do strani: http://www.py-scraping.eu/

Na strani se nahaja seznam 28 robotov v tabeli, vsaka vrstica tabele pa vsebuje tudi povezavo do profilne strani robota, kjer se nahajajo še drugi podatki o posameznem robotu.

a) Cilj prvega programa je, da z vsake profilne strani pridobi dva podatka - ime robota in kdo ga je ustvaril, nato pa te podatke shrani v .csv datoteko, ki bo shranjena v mapi, kjer se nahaja program.
b) Cilj drugega programa je, da s strani s slikami robotov pridobi vse slike in jih shrani v mapo, kjer se nahaja program, datoteke slik pa naj imajo enako ime kot alternativni tekst v njihovem HTML zapisu.

Oba programa sta napisana v Pythonu 3 in uporabljata modul urllib.request ter knjižnico BeautifulSoup 4. V starejših verzijah Pythona bi se sintaksa nekoliko razlikovala.

7.2. Izvedba naloge

a) V Python sem iz modula urllib.request uvozila funkcijo urlopen(). Nato sem v spremenljivko z imenom url shranila URL naslov spletne strani (v tem primeru http://www.py-scraping.eu/). V naslednji vrstici sem s funkcijo urlopen(), ki kot argument sprejme spremenljivko url, poklicala ta URL in s funkcijo read() pridobila HTML zapis s tega spletnega naslova. HTML zapis strani sem shranila v spremenljivko z imenom odziv. Da se je koda pravilno izvedla, sem preverila z ukazom print, ki je v terminalu izpisal HTML kodo strani (index.html).

```
from urllib.request import urlopen
url = "http://www.py-scraping.eu/"
odziv = urlopen(url).read()
print(odziv)
```

Nato sem na računalnik namestila knjižnico BeautifulSoup in jo uvozila v Python. V tem primeru sem uporabila še ključno besedo "as", ki omogoča, da sem se v nadaljevanju na knjižnico BeautifulSoup sklicevala s "Soup" (predvsem zato, da sem zmanjšala možnost sintaktičnih napak pri pisanju).
Nato sem spremenljivko odziv (kamor sem prej shranila HTML zapis strani) kot argument vnesla v metodo Soup() in vse skupaj shranila v spremenljivko z imenom soup. Uporabila sem še funkcijo prettify(), ki omogoča, da se v terminalu prikaže izpis HTML z vsemi indentacijami - boljša čitljivost HTML zapisa.

S tem, ko sem pridobila HTML zapis spletne strani v knjižnici BeautifulSoup, je postala navigacija po HTML-ju veliko bolj enostavna. Če npr. napišem soup.html.head.title.string, to pomeni, da navigiram po HTML strukturi preko značke <html> v značko <head>, kjer se nahaja značka <title> in pridobim vsebino le te. V terminalu se mi je izpisal naslov strani, kar pomeni, da koda pravilno deluje.

```
from urllib.request import urlopen
from bs4 import BeautifulSoup as Soup
```

```
url = "http://www.py-scraping.eu/"
odziv = urlopen(url).read()
soup = Soup(odziv, "html.parser")

print(soup.prettify())
print(soup.html.head.title.string)
```

HTML zapis strani vključuje tudi nekatere podatke, ki jih za izvedbo naloge nisem potrebovala - cilj je bil namreč iz HTML-ja pridobiti samo dve točno določeni vrsti podatkov. Iz HTML zapisa je razvidno, da se vsak izmed podatkov o robotu nahaja v svoji znački <td> (table data). Pri vsakem robotu je možnost klika na povezavo "Profil" - v HTML zapisu to izgleda kot:

<td>Profil</td>

Kar sem želela doseči od Python programa je, da sam "klikne" na vsako izmed povezav "Profil" in s profilne strani vsakega robota pridobi dva podatka - ime robota in kdo ga je ustvaril.

Najprej sem z metodo findAll() poiskala vse povezave <a> v HTML dokumentu in jih izpisala z ukazom print. Ukaz print je znotraj for zanke, zato da se izvede za vsako od najdenih povezav posebej, tako da je rezultat kode izpis vseh povezav v terminalu.

```
for link in soup.findAll("a"):
    print(link)
```

V terminalu se vidi, da je večina povezav takih, kot sem jih želela (takih, ki vodijo do profilne strani robotov), ne pa vse (v HTML dokumentu se nahaja še ena povezava, in sicer v naslovu "Spletna stran za vajo spletnega luščenja podatkov"). To pomeni, da sem morala s pogojnim stavkom zožiti izbor povezav. V tem primeru bi to lahko dosegla na dva načina:

- izbor bi lahko zožila le na tiste povezave, ki vsebujejo niz "Profil":

```
if link.string == "Profil" {koda...}
```

- ali pa na vse tiste povezave, ki ne vsebujejo niza "Spletna stran za vajo spletnega luščenja podatkov":

```
if link.string != "Spletna stran za vajo spletnega luščenja podatkov"
{koda...}
```

Ker gre za primerjavo dveh vrednosti, je v prvem primeru uporabljen dvojni enačaj ==, v drugem pa negacija !=.

Koda:

```
for link in soup.findAll("a"):
    if link.string == "Profil":
        print(link)
```

Znotraj href atributa je del URL-ja do profilnih strani. Da Python povezave prikaže v celoti (http://www.py-scraping.eu/robot-1.html) in ne kot HTML zapis povezav (Profil), je potrebno dodati še manjkajoči del URL-ja. To sem dosegla s konkatenacijo, tako da sem url naslovu dodala še link["href"] in oboje skupaj shranila v spremenljivko url_robota.

```
for link in soup.findAll("a"):
    if link.string == "Profil":
        url_robota = "http://www.py-scraping.eu/" + link["href"]
        print(url_robota)
```

Tako sem v terminalu dobila izpis vseh povezav, ki vodijo do profilnih strani posameznih robotov.

Cilj naslednjega koraka je bil, da program odpre vsako od teh povezav (kot bi uporabnik kliknil nanjo). V spremenljivki html_robota sem poklicala funkcijo urlopen() z argumentom url_robota (= spremenljivka iz prejšnjega koraka). Podobno kot na začetku sem s tem dobila HTML zapis, le da tokrat ne za osnovno spletno stran (index.html), ampak za vsako od profilnih strani, do katerih vodijo povezave. Da je program res pridobil HTML za vsako profilno stran sem dosegla z uporabo for zanke, tako da se spremenljivka html_robota nahaja znotraj zanke, kar pomeni, da se bo zanka izvedla za vsako od prej pridobljenih povezav.

```
for link in soup.findAll("a"):
    if link.string == "Profil":
        url_robota = " http://www.py-scraping.eu/" + link["href"]
        html_robota = urlopen(url_robota).read()
```

Do te točke je program torej pridobil HTML osnovne strani, nato v tem HTML-ju našel zgolj želene povezave in potem pridobil HTML za vsako od profilnih strani. To je tako, kot če bi

uporabnik kliknil na vsako izmed povezav "Profil" in nato kopiral vsebino vsake profilne strani.

V naslednjem koraku pa sem želela, da program poišče le imena robotov in njihovih ustvarjalcev.
Iz HTML zapisa je mogoče razbrati, da se podatek o imenu robota nahaja znotraj <h1> značke, podatek o tem, kdo je robota ustvaril, pa znotraj značke. Vendar pa so na strani še drugi h1 in span elementi (h1 v naslovu strani, span pa pri zapisu igralca, ki je igral robota oz. pri imenu igralca, ki je robotu dal glas). V tem primeru sem izbor zožila z navedbo imena razreda h1 oz. span elementa.
Vsak h1 element, ki vsebuje ime robota ima razred z imenom "robot-name" (<h1 class="robot-name">...</h1>), vsak span element, ki vsebuje ime ustvarjalca robota, pa razred "creator" (...). Po razredih je mogoče ločiti želene h1 in span elemente od preostalih h1 in span elementov.

Koda torej poleg funkcije find("h1") oz. find("span") vsebuje še attrs={"class": "robot-name"} oz. attrs={"class": "creator"}, kar natančno določi atribute želenih <h1> in značk. Da pa se v terminalu ne prikaže HTML zapis npr. <h1 class="robot-name">R2D2</h1> in George Lucas, je bilo potrebno uporabiti še .string, ki izpiše zgolj vsebino značk.

Pod obstoječo kodo (še vedno znotraj for zanke in pogojnega stavka) sem dodala:

```
robot_soup = Soup(html_robota, "html.parser")
imena = robot_soup.find("h1", attrs={"class": "robot-name"}).string
ustvarjalci = robot_soup.find("span", attrs={"class": "creator"}).string

print(imena + " - " + ustvarjalci)
```

Rezultat kode je izpis vseh imen robotov in imen njihovih ustvarjalcev, med imeni robotov in imeni ustvarjalcev pa stoji pomišljaj. Ukaz print mora biti znotraj for zanke, zato da se izpiše za vsako ime robota in ime ustvarjalca posebej.

V zadnjem koraku sem želela, da bi program vsa zgornja imena shranil kot .csv (Comma Separated Values) datoteko. To bi prišlo prav pri pridobljenih podatkih v večjem obsegu, ki bi jih želeli potem še urejati, spreminjati, pošiljati naprej, ipd.

Na vrh kode (pod uvozom knjižnic urllib in BeautifulSoup) sem najprej dodala vrstico:

```
csv_dat = open("seznam_robotov.csv", "w")
```

Ta ustvari novo .csv datoteko z imenom "seznam_robotov" in jo shrani v spremenljivko csv_dat, "w" pa določa način, v katerem je zapisana datoteka (file mode) in pomeni samo za pisanje (only writing) - to pomeni, da bo program ustvaril novo "seznam_robotov.csv" datoteko, če ta še ne obstaja oz. izbrisal to datoteko in jo na novo ustvaril, če že predhodno obstaja.

Nato sem pod vso obstoječo kodo dodala še dve vrstici:

`csv_dat.write(imena + " - " + ustvarjalci + "\n")` - ta vrstica se mora nahajati znotraj for zanke in pogojnega stavka, saj se mora izvesti za vsako ime robota in njegovega ustvarjalca posebej. Zapis s konkatenacijo omogoči, da v .csv datoteki med imeni robotov in ustvarjalcev stojijo pomišljaji, "\n" pa ustvari novo vrstico za vsakega robota posebej (drugače imena ne bi bila zapisana v stolpcu, ampak bi se nadaljevala v eni sami dolgi vrsti).

`csv_dat.close()` - se nahaja izven for zanke, saj zaključi in ustvarjeno .csv datoteko shrani v mapo, kjer se nahaja Python program (to se izvede samo enkrat).

a) Cel program:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup as Soup

csv_dat = open("seznam_robotov.csv", "w")

url = "http://www.py-scraping.eu/"
odziv = urlopen(url).read()

# print(odziv)
```

```
soup = Soup(odziv, "html.parser")
# print(soup.prettify())
# print(soup.html.head.title.string)

for link in soup.findAll("a"):
    if link.string == "Profil":
        url_robota = "http://www.py-scraping.eu/" + link["href"]
        html_robota = urlopen(url_robota).read()
        robot_soup = Soup(html_robota, "html.parser")
        imena = robot_soup.find("h1", attrs={"class": "robot-name"}).string
        ustvarjalci = robot_soup.find("span", attrs={"class": "creator"}).string
        print(imena + " - " + ustvarjalci)
        csv_dat.write(imena + " - " + ustvarjalci + "\n")

csv_dat.close()
```

Rezultat programa so izpisana imena robotov in njihovih ustvarjalcev v terminalu ter ustvarjena .csv datoteka s temi imeni, ki jo je možno odpreti v Excellu in nadalje urejati.

b) V drugem programu sem za osnovo vzela url strani:
http://www.py-scraping.eu/photos.html
Cilj programa je, da s strani pridobi vse slike in jih shrani v mapo, kjer se nahaja program ter datoteke slik poimenuje enako kot se glasi vsebina alt atributa v njihovem HTML zapisu.

Najprej sem na enak način kot v prvem programu, v Python uvozila modul urllib in knjižnico BeautifulSoup. Nato sem ustvarila funkcijo z imenom slike, ki ima parameter url. Znotraj funkcije se odvijejo trije koraki - v spremenljivko url se shrani url strani (ki ga vnesemo kot argument, ko pokličemo funkcijo), nato se v spremenljivko z imenom html shrani HTML zapis strani s tega url-ja, nazadnje pa z ukazom return pridobimo ta HTML. Funkcijo sem ustvarila zato, ker sem program preizkušala še z drugimi spletnimi stranmi in mi tako ni bilo potrebno vsakič znova pisati iste kode. Prednost funkcije je, da samo enkrat zapišemo kaj naj se zgodi znotraj funkcije, nato pa lahko funkcijo pokličemo večkrat tekom programa in ji pri tem damo poljuben argument.

```
import urllib
import urllib.request
from bs4 import BeautifulSoup as Soup
```

```
def slike(url):
    url = urllib.request.urlopen(url)
    html = Soup(url, "html.parser")
    return html
```

Nato sem poklicala funkcijo "slike" in kot argument vnesla url strani s slikami robotov ter jo shranila v spremenljivko soup. Potem sem na enak način kot sem v prvem programu poiskala vse povezave, tu poiskala vse slike, ki se nahajajo na url-ju, ki sem ga vnesla. Da sem slike dobila kot povezave, na katere je mogoče klikniti z miško, ne pa kot HTML zapis (), sem s funkcijo get() pridobila vire (src oz. source) slik in jih shranila v spremenljivko z imenom src. Ponovno sem uporabila for zanko, zato da se ukaz izvede za vsako sliko posebej.

```
soup = slike("http://www.py-scraping.eu/photos.html")
for img in soup.findAll("img"):
    src =(img.get("src"))
```

V naslednjem koraku sem s funkcijo get() pridobila še alt atribut slik, zato da se imena datotek slik ujemajo z alternativnim tekstom slik v HTML-ju:

```
ime_dat = img.get("alt")
```

V mojem primeru tu ni bilo nobenih težav, saj imajo vse slike v HTML-ju določen alternativni tekst, če pa ga katera od slik ne bi imela, potem bi morala določiti kaj se zgodi, v primeru, da slika nima alt teksta. Ena od možnosti bi bila npr. da bi na začetku programa uvedla števec i = 1, potem pa znotraj for zanke s pogojnim stavkom določila, da je v primeru, ko slika nima alt teksta, ime datoteke enako nizu od i (ki se pri vsaki naslednji sliki poveča za 1 = inkrementacija), v primeru, da slika ima alt tekst, pa je ime datoteke enako kot alt tekst:

```
alt = img.get("alt")
    if len(alt)==0:
        ime_dat=str(i)
        i = i + 1
    else:
        ime_dat=alt
```

Vendar pa kot rečeno, imajo v mojem primeru vse slike alt tekst, tako da števca in pogojnega stavka nisem potrebovala.

V zadnjem koraku sem s funkcijo open() ustvarila novo datoteko za vsako sliko - datoteka ima enako ime kot je alt tekst slike in končnico .jpg (vse slike, ki se nahajajo na spletni strani so v formatu .jpg), "wb" pa je način, v katerem je shranjena datoteka (file mode) in pomeni "write binary" - datoteka je odprta za pisanje v binarnem načinu (če bi želeli datoteko zgolj brati, bi uporabili način "rb" oz. read binary) - wb in rb način se uporabljata za datoteke, ki niso tekstovne. Na koncu sem zaprla ustvarjene datoteke. Tudi te tri vrstice kode morajo biti znotraj for zanke, zato da se izvedejo za vsako sliko posebej in je rezultat programa shranjenih 28 slik robotov, ki se nahajajo v mapi, kjer je tudi program.

```
slika_dat = open(ime_dat + ".jpg", "wb")
slika_dat.write(urllib.request.urlopen(src).read())
slika_dat.close()
```

b) Cel program:

```
import urllib
import urllib.request
from bs4 import BeautifulSoup as Soup

def slike(url):
    url = urllib.request.urlopen(url)
    html = Soup(url, "html.parser")
    return html

soup = slike("http://www.py-scraping.eu/photos.html")
for img in soup.findAll("img"):
    src =(img.get("src"))
    ime_dat = img.get("alt")
    slika_dat = open(ime_dat + ".jpg", "wb")
    slika_dat.write(urllib.request.urlopen(src).read())
    slika_dat.close()
```

7.3. Omejitev knjižnice BeautifulSoup

Ena od omejitev knjižnice BeautifulSoup, na katero sem naletela med pisanjem programov v Pythonu je, da ne podpira pridobivanja podatkov, ki so zapisani v JavaScriptu.
Na spletni strani o robotih je bil podnaslov sprva zapisan v znački kot:
Seznam robotov in androidov iz filmov, serij, knjig, resničnih... .

V tem primeru bi s knjižnico BeautifulSoup lahko brez težav pridobila ta element (ki ima id z imenom "podnaslov"). Ko pa sem podnaslov v HTML-ju spremenila v:
Podnaslov
in v HTML dodala JavaScript:
<script>
document.getElementById("podnaslov").innerHTML = "Seznam robotov in androidov iz filmov, serij, knjig, resničnih...";
</script>

in potem v Python zapisala:

```
from bs4 import BeautifulSoup as Soup
from urllib.request import urlopen

url = "http://www.py-scraping.eu/"
odziv = urlopen(url).read()
soup = Soup(odziv, "html.parser")

podnaslov = soup.find("p", attrs={"id": "podnaslov"}).string
print(podnaslov)
```

je bil rezultat programa pridobljena vsebina HTML zapisa, torej "Podnaslov", ne pa JavaScript zapisa, ki je v resnici zapisan in ga vidimo, ko odpremo spletno stran -> " Seznam robotov in androidov iz filmov, serij, knjig, resničnih...".

Med iskanjem po spletu sem izvedela, da knjižnica BeautifulSoup ni dovolj, da bi s spleta lahko pridobila JavaScript zapis in da so za to potrebne močnejše knjižnice, npr. Selenium ali

PyQT. Knjižnica BeautifulSoup je torej primerna za luščenje podatkov iz HTML zapisa, če želimo pridobiti tudi vsebino iz JavaScripta, pa je potrebna še uporaba katere druge knjižnice.

8. ZAKLJUČEK

Ko sem začela pisati seminarsko nalogo nisem vedela praktično ničesar o luščenju podatkov s spleta, imela pa sem solidno znanje HTML-ja, CSS-ja in JavaScripta, zato se mi je zdelo smiselno, da to znanje povežem še s programiranjem v Pythonu in se naučim osnov spletnega luščenja podatkov. Zelo zanimiv se mi zdi pravni vidik luščenja podatkov s spleta. Na internetu sem našla številne tutoriale, ki opisujejo načine, kako z Googla pridobiti vse slike določenega iskalnega niza. Vendar pa so vsi tutoriali opozarjali, da Google ne podpira takega luščenja podatkov in da lahko v primeru, da z modulom urllib.request dostopaš do podatkov, blokira tvoj IP naslov in se na ta način zaščiti. Večina tutorialov zato prikazuje kako s knjižnico Selenium preslepiti to pravilo in brskalniku dati vtis, da gre za uporabnika, ki odpira in shranjuje slike, ne pa za računalniški program.

V svojem primeru sem se odločila, da kar sama izdelam preprosto spletno stran, ki ne vsebuje robots.txt datoteke, poleg tega pa sem tako lahko sproti spreminjala in prilagajala HTML zapis in preizkušala kako na različne načine s strani pridobiti različne podatke. Zanimivo se mi je zdelo, ko sem naslov v HTML-ju zamenjala z JavaScript zapisom in nato te vsebine nisem mogla pridobiti le z uporabo knjižnice BeautifulSoup. V prihodnje se bom zato seznanila še s knjižnico Selenium ali pyQT in se naučila še osnov pridobivanja podatkov iz JavaScripta. To bi bilo uporabno, saj večina spletnih strani danes vsebuje elemente JavaScripta ali pa knjižnic JQuery, AngularJS, itd.

Pri pisanju programa, ki s spletne strani pridobi in shrani vse slike robotov, sem imela nekaj težav. Sprva sem stran namreč oblikovala tako, da se je slika posameznega robota nahajala na njegovi profilni strani. Tako je moral program najprej pridobiti HTML naslov vsake profilne strani (kot pri prvem programu pri pridobivanju imen robotov), nato pa na vsaki strani identificirati <img> značko, pridobiti url slike in jo nato shraniti. Vendar pa je moj program v mapo vsakič shranil samo 22 slik, ne pa vseh 28-ih slik (toliko jih je vse skupaj). Po internetu in forumih sem iskala kakšen bi lahko bil razlog za to težavo, vendar rešitve nisem našla. Koda je bila namreč znotraj for zanke, tako da bi morala delovati za vse slike enako, poleg tega pa so vse slike na spletni strani tudi v istem formatu (jpg). Zato mi ni čisto jasno zakaj bi program deloval samo za 22 slik, ne pa za vse. Ker rešitve nisem našla, sem spremenila HTML in vse slike robotov prestavila na skupno stran. Nato sem temu prilagodila program, ki je potem deloval brez težav.

VIRI

(obiskani med 16.2. in 23.2.2017)

http://mafalda.informatika.uni-mb.si/2007-08/szizr/predavanja/04-semantic%20web.pdf

http://www.lavbic.net/delo-in-raziskovanje/semanticni-splet/

https://docs.python.org/3/howto/urllib2.html

https://docs.python.org/2/library/urllib.html

https://www.coursera.org/learn/python-network-data/lecture/Dl9XG/parsing-html-with-beautifulsoup

https://www.coursera.org/learn/html/lecture/Lhlau/01-02-the-evolution-of-html

https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

https://www.w3.org/2002/07/26-dom-article.html

https://en.wikipedia.org/wiki/Document_Object_Model

https://tools.ietf.org/html/rfc1866

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

https://www.becontentking.com/academy/robotstxt/

http://web.stanford.edu/~zlotnick/TextAsData/Web_Scraping_with_Beautiful_Soup.html

https://www.analyticsvidhya.com/blog/2015/10/beginner-guide-web-scraping-beautiful-soup-python/

http://altitudelabs.com/blog/web-scraping-with-python-and-beautiful-soup/

https://www.w3.org/XML/

https://en.wikipedia.org/wiki/XML