Primjena Python biblioteke Beautiful Soup

Matuzić, Luka

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski dvopredmetni studij informatike i filozofije

Luka Matuzić

Primjena Python biblioteke Beautiful Soup

Završni rad

Mentorica: izv. prof. dr. sc. Ana Meštrović

Rijeka, 2017.

Sadržaj

Sažetak

Beautiful Soup je Python biblioteka za dohvaćanje podataka iz HTML i XML datoteka. Radi u kombinaciji s parserom kako bi korisniku pružio razne načine navigacije, traženja i izmjene samog stabla parsiranja. Programerima obično štedi sate ili dane rada. Način rada koji opisuje rad same biblioteke Beautiful Soup naziva se web struganje *(eng. web scraping),* čime čitavi internet postaje baza podataka iz kojeg se mogu dohvatiti željeni podaci. U završnom radu demonstrirat ću detaljan prikaz rada Pythonove biblioteke Beautiful Soup na specifičnom primjeru. S unaprijed odabranim informacijama koje želim dohvatiti sa već odabrane web stranice ispisati ću podatke u .csv datoteci unutar Excela u svrhu prikaza dohvaćenih podataka unutar tablice. Tablica se sastoji od 3 zaglavlja *(eng. headera)* koji predstavljaju te podatke te svakom izmjenom podataka na samoj web stranici, moj web strugač će, uz pomoć Beautiful Soupa, automatski ažurirati i ispisati nove podatke unutar tablice u Excelu. Dakle, može se primijeniti za buduću upotrebu te može služiti kao predložak *(eng. template)* za svaki budući strugač podataka s bilo koje web stranice.

Ključne riječi: Python, Beautiful Soup, HTML, parsiranje, stablo parsiranja, web struganje, internet (web)

1. Uvod

U završnom radu baviti ću se Python bibliotekom Beauful Soup koja se koristi za web struganje, odnosno dohvaćanje potrebnih podataka s weba. Najviše se koristi kako bi dohvatili određene, nama bitne, podatke s web stranice koja nije previše pregledna i do čijih podataka ne možemo doći na jednostavan način. Beautiful Soup mnogim programerima skraćuje potrošeno vrijeme i novac koji bi se mogli utrošiti u neke bitnije stvari od ručnog dohvaćanja podataka s određene web stranice. Ova Python biblioteka omogućuje nekoliko jednostavnih metoda za navigaciju, pretraživanje i modificiranje stabla parsiranja, o čemu ću detaljnije govoriti u nastavku ovog rada. Za sada, to stablo neka se odnosi na raščlambu određenog dokumenta ili web stranice te dohvaćanja onih informacija koje tražimo. Beautiful Soup parsira, odnosno raščlanjuje bilo što što joj se da te u isto vrijeme obuhvaća sve ono što se u stablu parsiranja i nalazi. Laički rečeno, može joj se „reći“ da nađe, odnosno vrati sve linkove s neke određene web stranice, da nađe sve linkove npr. neke klase eksternalnog linka, da vrati sve linkove čija URL adresa u sebi sadrži npr. „facebook.com“, da prikaže sva zaglavlja neke tablice koje imaju podebljani *(eng. bold)* tekst te da taj tekst posebno zapiše kao .csv datoteka u Excelu i sl. [3]. Za pronalazak svih tih podataka, tj. za njihovo dohvaćanje, trebalo bi puno vremena i truda koje ova Pythonova biblioteka obavi za svega nekoliko minuta, ako se s njom zna pravilno služiti i ako znamo točno koje podatke i informacije želimo dohvatiti.

2. Python

Budući da je Beautiful Soup jedna od biblioteka programskog jezika Python, sada ću navesti nekoliko bitnijih informacija o njemu. Python je široko korišten programski jezik za programiranje opće namjene kojeg je stvorio Guido van Rossum i prvi put je objavljen 1991. godine. Python ima filozofiju dizajna koja naglašava čitljivost kôdova (naročito korištenjem razmaka, odnosno uvlačenja, kako bi se odredili kôdni blokovi, za razliku od vitičastih zagrada ili nekih ključnih riječi) te sintaksu koja programerima omogućava da izraze ono što su htjeli napraviti u puno manje linija kôda nego što bi to bilo npr. u C++-u ili u Javi. Dakle, ovaj jezik je konstruiran kako bi se omogućilo „čisto“ pisanje kôdova i programa malih i velikih razmjera [9].

Jedna od bitnijih značajki Pythona kao programerskog jezika je njegov dinamički sustav te automatsko upravljanje memorijom. Isto tako, podržava i višestruke programerske paradigme uključujući objektno orijentirano programiranje, imperativno i funkcijsko programiranje te razne proceduralne stilove [9].

Python je dostupan za mnoge operacijske sustave što mu omogućuje da se može raditi na doista širokom izboru raznih sustava.

2.1. Povijest

Python je osmišljen u kasnim 1980.-ima, a njegova implementacija počela je u prosincu 1989. godine od strane Guida van Rossuma u „Centrum Wiskunde & Informatica“ u Nizozemskoj i to kao nasljednik programerskog jezika ABC koji je također osmišljen i implementiran na istom institutu u Nizozemskoj.

Sam Guido van Rossum je rekao da je Python nastao kao hobi koji ga je trebao okupirati tijekom božićnih praznika kada nije mogao raditi u uredu jer je bio zatvoren za vrijeme blagdana. Htio je napraviti programski jezik koji bi se sviđao Unix/C hakerima, a samo ime Python je poteklo od njegove najdraže humoristične serije „Monty Python's Flying Circus“ [9].

2.2. Uporedba s ostalim programskim jezicima

Unutar IT zajednice česte su kritike Pythona na račun njegove brzine, odnosno njegove sporosti. Budući da je Python interpreterski jezik, programi napisani u njemu vrše se malo sporije za usporedbu od kompajlerskih *(eng. compile)* jezika, kao što su C, C++ i slični. Međutim, unatoč toj brzinskoj manjkavosti, u industriji se Python poprilično koristi ponajviše kao razvojni program za krajnjeg korisnika *(eng. back end)* [9], što znači da se koristi za rad na arhitekturi i samoj logici sustava. Laički rečeno, razvojni program za krajnjeg korisnika predstavlja ono što se ne vidi na ekranu, a esencijalno je za rad bilo čega što se koristi, bila to neka određena web stranica, aplikacija, igra i sl.

Python se često uspoređuje s Javom. Oba su prevoditeljski *(eng. interpreter)* jezici te nemaju gotovo nikakvu podršku za višejezgreno izvođenje programa, pošto i Python i Java koriste samo jednu procesorsku jezgru. Java je kao jezik puno primjenjenija u izradi mobilnih aplikacija i interaktivnog web sadržaja, dok je Python gospodar PC svijeta. Što se tiče brzine izvođenja programa, Java i Python su približno jednaki [9].

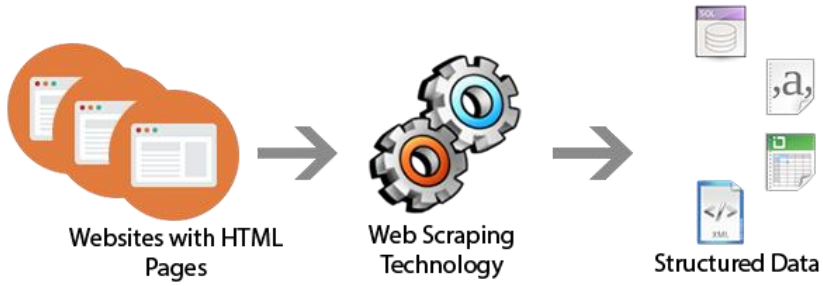
3. Web struganje

U ovom poglavlju detaljnije ću objasniti kako web struganje funkcionira. Softveri koji koriste web struganja mogu pristupiti WWW-u koristeći HTTP protokol ili kroz određeni web preglednik *(eng. browser)*. Web struganje se može činiti ručno *(eng. manual)*, odnosno od strane osobe koja koristi prikladni softver, ali pod tim terminom se obično misli na automatizirane i implementirane procese koristeći botove i web puzavce *(eng. web crawler)*.

Bot je softverska aplikacija koja izvršava automatizirane zadatke ili skripte na internetu, dok je web puzavac (ponekad se naziva spider ili pauk) bot koji automatski pretražuje WWW i to ponajviše u svrhu web indeksiranja. Web puzavci mogu kopirati sve stranice koje posjete za kasnije procesuiranje određenog pretraživača *(eng. search enginea)* koji indeksira te posjećene stranice koje puzavci skidaju s weba, tako da korisnici mogu što učinkovitije i brže pretraživati ono što žele. Zanimljivo je da se više od polovice internet prometa sastoji od botova i web puzavaca.

Web struganje uključuje dohvat (*engl. fetching*) i izvod *(eng. extracting)* podataka i to u HTML obliku. Dohvat se odnosi na skidanje web stranice koje čini web preglednik kada se ode na određenu web stranicu i tu se u priču ubacuje web puzanje koji zapravo postaje glavna komponenta samog web struganja. Nakon što je web stranica dohvaćena, nastupa izvod. Sadržaj web stranica mogu biti parsirani (raščlanjeni), pretraženi, reformatirani, njihovi podaci kopirani u razne tablice, strukturirane liste i sl.

Web strugači uobičajeno uzmu neke podatke s neke web stranice kako bi se ti podaci mogli koristiti negdje drugdje u neku potpuno drugu svrhu. Ti podaci su najčešće u raznim skriptnim programima u HTML obliku te se njima upravlja prema želji. Npr. struganje kontakta *(eng. contact scraping),* koji se odnosi na pretraživanje i kopiranje imena ljudi i njihovih telefonskih brojeva, raznih tvrtki i njihovih URL adresa, itd.



Slika 1 – Grafički prikaz osnovnih koraka u postupku web struganja [5]

3.1. Tehnike web struganja

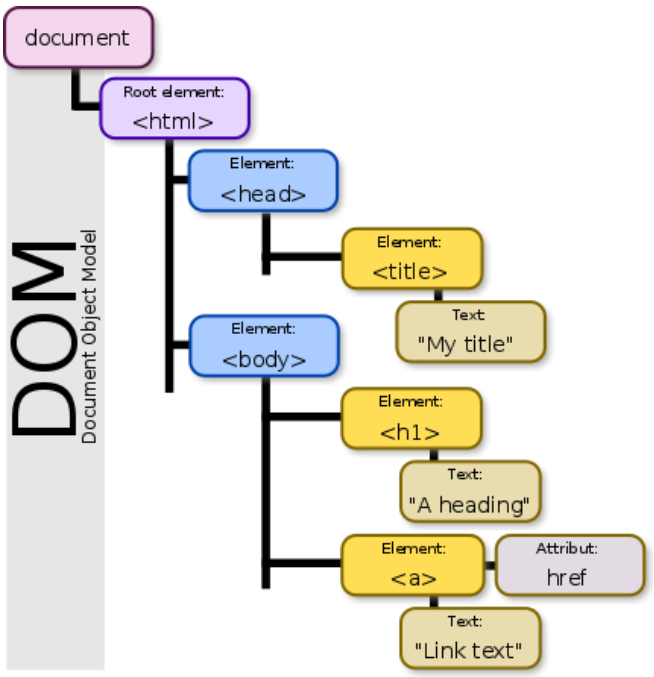
1. Kopiraj – zalijepi *(eng. copy – paste)* od strane čovjeka → ponekad ni najbolje tehnologije web struganja ne mogu zamijeniti ručno kopiranje podataka te je to ponekad jedini način kopiranja podataka. Razlog tome je to što neke web stranice imaju postavljene barijere kreirane specifično za zaustavljanje bilo kakvih automatskih dohvaćanja podataka te samim time botovi i web puzavci ne mogu obavljati ono za što su programirani

2. Podudaranje uzoraka teksta → jednostavan, ali veoma učinkovit način dohvaćanja informacija i podataka s weba, a odnosi se na tekst koji odgovara regularnim izrazima *(eng. regular expression)*, odnosno slijedovima znakova koji definiraju određeni traženi uzorak

3. HTTP programiranje → statične i dinamične web stranice i njihovi podaci mogu biti dohvaćeni postavljanjem HTTP zahtjeva na udaljeni web poslužitelj

4. HTML parsiranje → mnoge web stranice imaju velike zbirke stranica dinamički generiranih iz temeljnog strukturiranog izvora kao što je baza podataka. Podaci iste kategorije obično su kodirani na slične stranice zajedničkim skriptom ili predloškom. U podatkovnom rudarenjenju *(eng. data miningu)* (odnosi se na sortiranje, organizaciju i grupiranje velikog broja podataka i izvlačenje relevantnih podataka), program koji detektira takve predloške u određenom izvoru informacija, „izvlači" svoj sadržaj i prevodi ga u relacijski oblik, naziva se omotač *(eng. wrapper)*. Algoritmi omotača pretpostavljaju da ulazne stranice sustava indukcijskog omotača (indukcijskog u smislu uvođenja u neki skup podataka) omogućuju zajednički predložak i da se mogu lako identificirati u smislu zajedničke sheme URL-a. Štoviše, neki polu-strukturirani jezici za upite podataka, kao što su XQuery i HTQL, mogu se koristiti za analiziranje HTML stranica te za preuzimanje i promjenu sadržaja same web stranice

5. DOM parsiranje → DOM ili Document Object Model je sučelje za aplikacijsko programiranje koji se temelji na različitim platformama i jezicima koji obrađuju HTML, XHTML ili XML dokument kao strukturu stabla, pri čemu je svaki čvor objekt koji predstavlja dio dokumenta. Predmeti se mogu programirati i sve vidljive promjene koje se javljaju kao rezultat mogu se odraziti na prikazu dokumenta. Ugrađivanjem punopravnog web preglednika, kao što je Google Chrome ili Mozilla, programi mogu dohvatiti dinamički sadržaj koji generiraju skripte na strani klijenta. Te kontrole preglednika također analiziraju web stranice u DOM stablo, na temelju kojih programe mogu dohvatiti dijelove stranica [10].



Slika 2 - DOM stablo [4]

6. Vertikalna agregacija → odnosi se na platforme koje stvaraju i prate mnoštvo botova za određene vertikale bez "čovjeka u petlji" (bez izravne ljudske uključenosti), a nema posla koji se odnosi na određeno ciljanu web-lokaciju. Priprema uključuje uspostavu baze znanja za cijelu vertikalnu platformu, a zatim platforma automatski stvara botove. Kvaliteta platforme mjeri se kvalitetom informacija koje prima (obično broj polja) i njezine skalabilnosti (koliko brzo može doći do stotina ili tisuća web stranica) [11].

4. Beautiful Soup

Beautiful Soup je Python biblioteka za dohvaćanje podataka iz HTML i XML datoteka. Radi u kombinaciji s parserom kako bi korisniku pružio idiomatske načine navigiranja, traženja i izmjene samog parserskog stabla. Programerima obično štedi sate ili dane rada [3].

Web struganje je vrlo moćan alat za dohvaćanje željenih podataka s određene web stranice. S web struganjem cijeli internet postaje baza podataka. U daljnem dijelu ovog završnog rada prikazati ću kako analizirati web stranicu u podatkovnu .csv datoteku pomoću Python paketa pod nazivom BeautifulSoup.

Za ovaj rad koristio sam primjer Phuca Duonga [13], višeg softverskog inženjera *(eng. senior software engineer)* koji radi u centru za podučavanje analize podataka *(eng. data science)* pod nazivom Data Science Dojo [14].

Koristio sam najpopularniju Pythonovu platformu za analizu podataka pod nazivom Anaconda i Sublime Text 3, a to je sofisticirani uređivač teksta *(eng. text editor)* za kôd. Također, web stranica koju sam iskoristio za prikaz rada ove biblioteke je NewEgg.com, a ta stranica se bavi prodajom djelova za računala, laptope i razne druge elektronske uređaje.

4.1. Rad s Beautiful Soupom

Za početak, napravio sam instalaciju Anaconde kako bi preko nje u sučelju *command prompt* mogao raditi s Pythonom te sam instalirao Sublime Text 3 kako bi preko uređivača teksta provjerio što sam radio i da li je to što sam radio bilo u redu.

Kako bi uopće mogao raditi s Beautiful Soupom, prvo treba instalirati paket koji podržava tu biblioteku. Taj paket naziva se bs4, a za samu instalaciju, koristi se pip, odnosno sustav za upravljanje paketima koji se koristi za instalaciju i upravljanje programskim paketima napisanima u Pythonu.

Dakle, naredba glasi ovako:

```
C: \Users\Bubara\Desktop\webscrape> pip install bs4
```

Dakle, pip omogućava pristup samoj biblioteci Beautiful Soup i potrebna je cijela, tj. apsolutna putanja kako bi ispravno radio zato što pip nije jedna od Pythonovih sintaksa nego je to alat naredbenog retka *(eng. command line tool)* pomoću kojeg se omogućuje korištenje Pythonove sintakse. Tijekom izrade ovog završnog rada, koristio sam trenutno najnoviju verziju Beautiful Soupa, a to je Beautiful Soup 4 (bs4). Nadalje, kako bi počeo koristiti samu biblioteku, potrebno ju je uvesti *(eng. import)* u sučelje *command prompt* unutar kojega je već instaliran paket koji podržava biblioteku Beautiful Soup.

Središnja uloga biblioteke Beautiful Soup je parsiranje HTML datoteka te je to zapravo sve što ona čini. Kako bi uopće mogli dohvatiti nešto s weba, potreban je web klijent. Unutar Pythona, to se čini pomoću paketa pod nazivom `urllib`, a unutar njega nalazi se modul `request`, a unutar tog modula je funkcija koja se naziva `urlopen` pomoću koje se dolazi do samih podataka na webu. Linija kôda pomoću koje se koristi `request` modul i `urlopen` kako bi se nešto dohvatilo s weba:

```
>>> from urllib.request import urlopen as uReq
```

Kako ne bi morao svaki puta pisati `urlopen`, tu funkciju sam stavio u varijablu `uReq` radi bržeg i lakšeg pisanja kôda. Nadalje slijedi samo uvoženje biblioteke Beautiful Soup unutar paketa bs4, a to se radi na sljedeći način:

```
>>> from bs4 import BeautifulSoup as soup
```

U ovom slučaju sam, kao i u prethodnom, biblioteci BeautifulSoup dao ime `soup` kako ne bi morao svakog puta pisati cijelo ime. Ono što ova linija kôda radi je to da Beautiful Soup parsira HTML tekst, a `urllib` dohvaća samu web stranicu na kojoj se taj HTML tekst i nalazi. U ovom slučaju, to će biti web stranica NewEgg.com na kojoj se nalaze razni elektronski uređaji, računala, laptopi i sl. Recimo da nas na toj stranici zanimaju grafičke kartice. Pomoću web strugača kojeg ću u nastavku dovršavati, moći ću skupiti podatke svih tih grafičkih kartica, bila to njihova procjena *(eng. rating)*, cijena, puštanje novih grafičkih kartica u prodaju kroz par tjedana, tada sam u mogućnosti za par tjedana ponovno pokrenuti svoj strugač te će se podaci sa web stranice NewEgg.com o grafičkim karticama automatski ažurirati *(eng. update)* u bilo koji skup podataka koji želim, bila to npr. baza podataka, .csv datoteka, Excel datoteka i sl.

Budući da želim dohvatiti podatke s web stranice NewEgg.com, potrebna mi je URL adresa te web stranice koju ću kopirati ili u svoj naredbeni prozor *(eng. command window)* s Anacondom ili u SublimeText-u 3.

To se čini na sljedeći način:

```
>>> my_url = 'https://www.newegg.com/Video-Cards-
VideDevices/Category/ID-38?Tpk=graphic%20cards '
```

Sada se vraćam na web klijent te dohvaćam podatke s gore navedene web stranice:

```
>>> uReq (my_url)
```

Ova linija kôda na neki način „otvara“ vezu s web stranicom i dohvaća njezine podatke na način da skida *(eng. download)* čitavu web stranicu. Kako ne bih konstantno ponavljao isti zahtjev, stavljam ga u varijablu naziva `uClient`:

```
>>> uClient = uReq (my_url)
```

Sada kada se sadržaj web stranice skinuo, možemo reći da on trenutno„visi u zraku“ te trebamo nešto čime ćemo taj sadržaj moći pročitati. Obzirom da se Beautiful Soup bavi parsiranjem HTML teksta web stranica, trebamo neki način da pročitamo taj HTML, a za to služi naredba `.read ( )`.

```
>>> page_html = uClient.read ()
```

Pošto radimo s web klijentom i s otvorenom internet vezom, poželjno je uvijek zatvoriti radnju kako ne bi došlo do curenja podataka i memorije.

```
>>> uClient.close ()
```

S ovime što sam do sada napravio, dohvaćamo jednu veliku spojenu masu HTML teksta koji nema nekog prevelikog smisla nekome tko uopće ne razumije HTML te ga sada trebamo parsirati. Time se vraćam na poziv funkcije kojeg sam nazvao `soup` i koja predstavlja samu biblioteku Beautiful Soup i pomoću koje će se vršiti parsiranje HTML teksta.

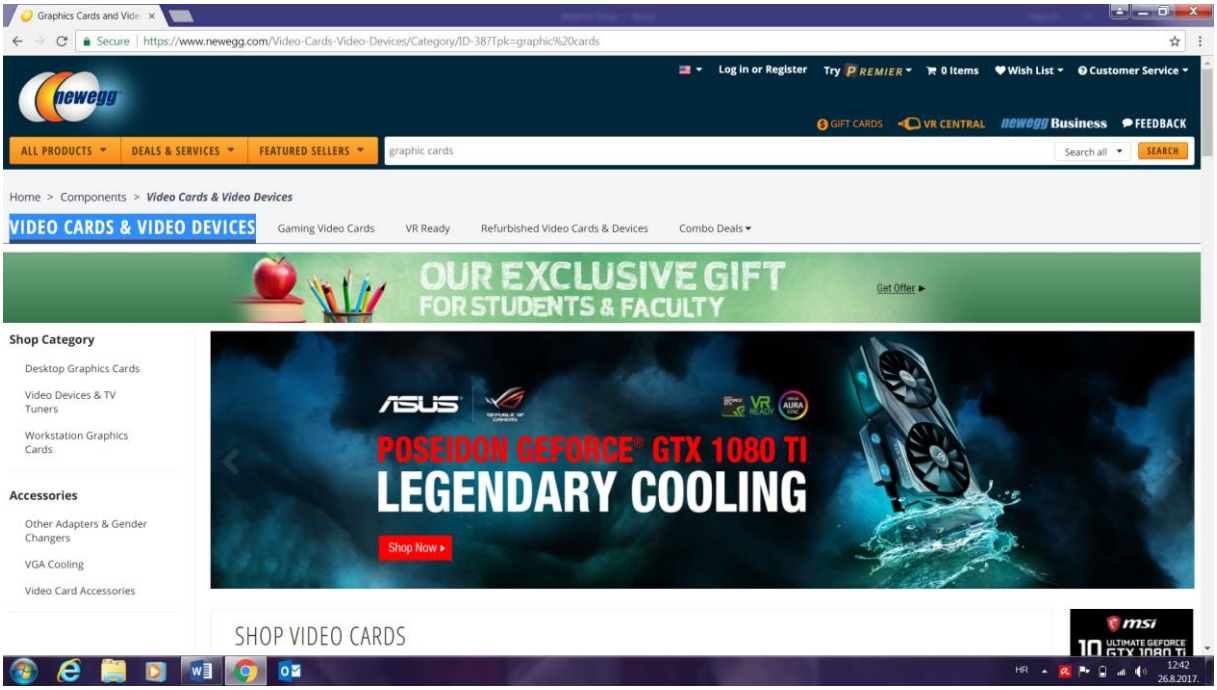
```
(>>> from bs4 import BeautifulSoup as soup)

>>> page_soup = soup (page_html, “html.parser“)
```

Dio koji se odnosi na **“**`html.parser`**“** označava način parsiranja, pošto to može biti npr. XML datoteka ili nešto drugo, no u ovom slučaju koristi se **“**`html.parser`**“** zato što parsiramo HTML datoteku, odnosno HTML tekst te ga pohranjujem u varijablu `page_soup` kako se ne bi izgubilo ono što se parsira.

U svrhu testiranja dosadašnjeg dijela kôda, ovo se dobije kada želimo dohvatiti zaglavlje web stranice NewEgg.com dosadašnjim naredbama, odnosno `<h1>` oznaku u HTML-u.

```
<h1 class="page-title-text"> Video cards &amp;amp; Video Devices </h1>
```

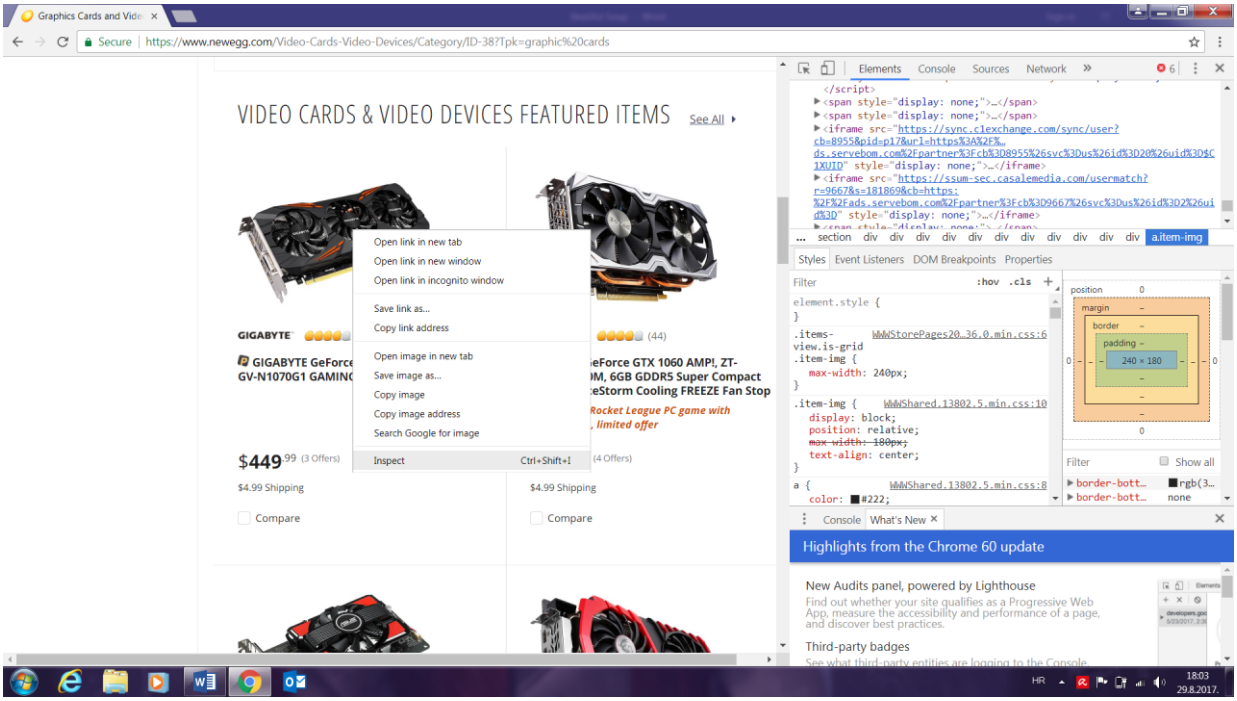


Slika 3 - header web stranice NewEgg.com

Sada se pomoću varijable `page_soup` može pristupiti svakom dijelu naslovne stranice NewEgg.com, npr. pomoću linije `page_soup.p` može se pristupiti paragrafu naslovne stranice na kojoj se trenutno nalazi, može se pristupiti raznim linkovima koji se nalaze na toj stranici pomoću `page_soup.a`, itd. Već tu se vidi koliko je Beautiful Soup moćan alat i do koliko raznih podataka i informacija možemo doći s doslovno par linija kôda.

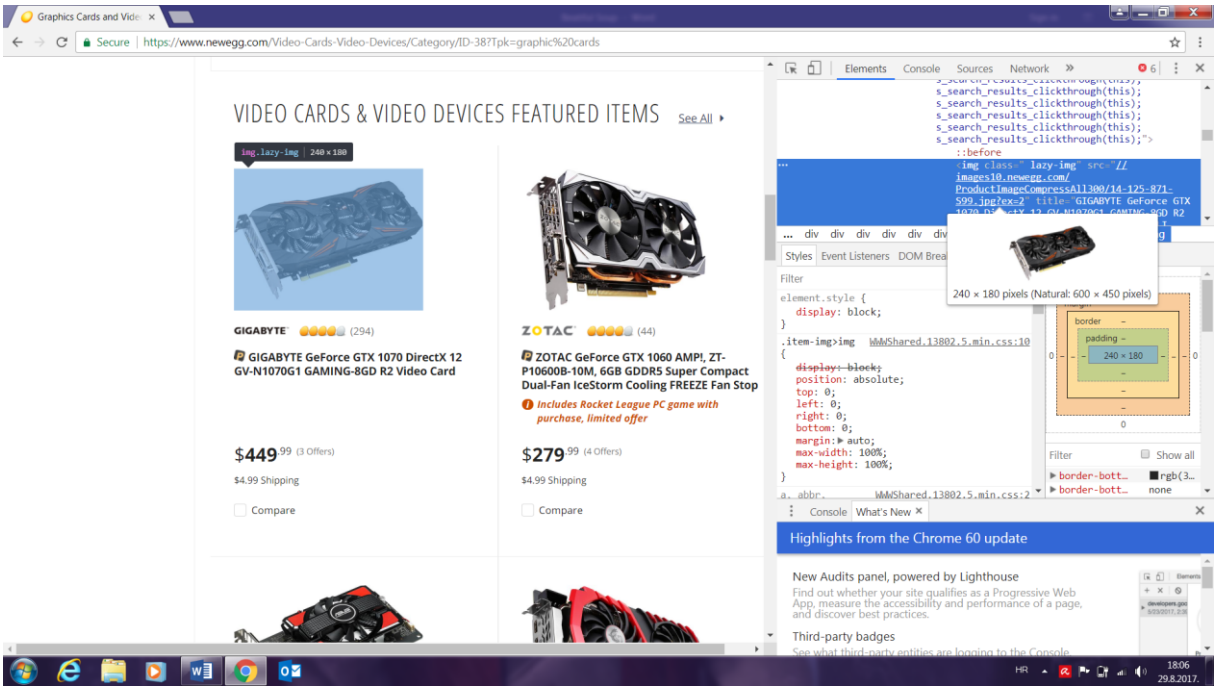
Pošto sam uzeo primjer sa grafičkim karticama, sada ću prikazati kako pretvoriti svaku grafičku karticu koju vidim na stranici NewEgg.com u običnu stavku (red) unutar tablice u .csv datoteci u Excelu. Ono što zapravo treba učiniti jest to da tip podataka koji trenutno imam (a to je Beautiful Soup tip podataka) „križam“ s elementima prije navedenih DOM elemenata. Jednostavnije rečeno, ono što treba napraviti jest parsirati elemente web stranice na kojoj se nalazim u DOM stablo (raščlaniti ih) kako bi ih lakše mogao dohvatiti.

Razmotrimo jedan od elemenata kojeg promatram (jednu grafičku karticu) desnim klikom na njega te zatim klikom na inspect koji označava pregled HTML teskat tog elementa. S gornje desne strane web stranice pojavit će se HTML tekst navedenog elementa, tj. grafičke kartice koju razmatram.



Slika 4 - Inspect element - HTML tekst s gornje desne strane

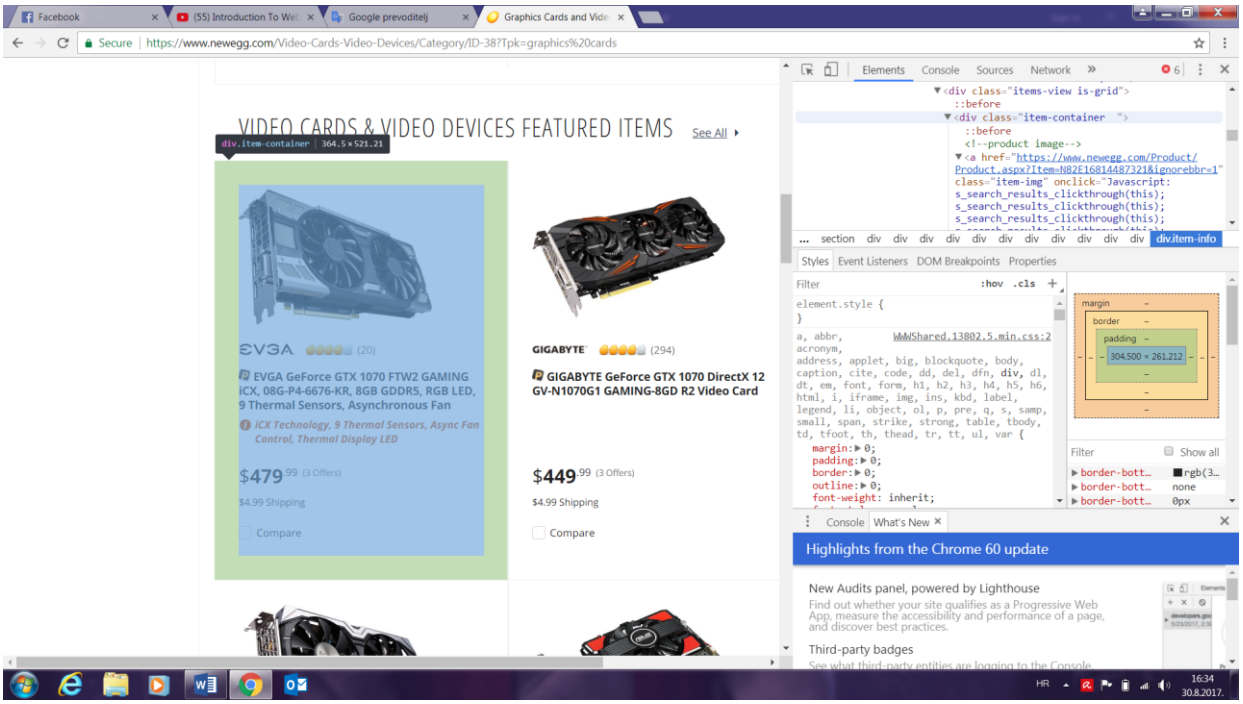
Ako pomičem strelicu miša unutar tog HTML-a prema dolje *(eng. scroll down)*, tada će se s lijeve strane označavati djelovi razmatranog elementa, npr. naziv grafičke kartice, njena slika, vrijednost i sl.



Slika 5 - Scroll po HTML-u - oznaka slike (img class)

Na isti način možemo pristupiti raznim stavkama određenog elementa, npr. ime, vrijednost, procjenu, itd. Ono što trebam učiniti je zahvatiti kontejner *(eng. container)* unutar kojega se nalaze sve stavke određene grafičke kartice. To se čini postavljanjem petlje *(eng. loop)* koja pomaže da dohvatimo stavke jedne grafičke kartice te će se na isti način moći doći do stavki drugih grafičkih kartica. Dakle, prvo će se parsirati stavke jedne grafičke kartice, a kada to parsiranje završi, dolazi do prelaska parsiranja stavki druge grafičke kartice, itd. Ta petlja će prolaziti kroz sve kontejnere grafičkih kartica po njihovim HTML tekstovima za svaki pojedini dio svake pojedine grafičke kartice.

Funkcija koja se koristi za pronalazak onoga što trebam, u ovom slučaju HTML teksta unutar kontejnera, tj. podataka grafičkih kartica, naziva se `findAll()`. Dakle, ono što pokuašavam ovdje učiniti jest dohvatiti sve `<div>` oznakes u HTML tekstu koji u sebi sadržava `class:item-container`.



Slika 6 - div tag s klasom kontejnera koji označava sve stavke

To činim na sljedeći način:

```
>>> containers = page_soup.findAll (“div“, {“class“: “item-container“})
```

Ovu liniju kôda sam testirao da provjerim koliko objekata, tj. koliko grafičkih kartica se nalazi na web stranici NewEgg.com, odnosno iz koliko ću ukupno kontejnera dohvatiti podatke. Trenutno se na NewEgg-u nalazi 12 grafičkih kartica u ponudi i stoga je rezultat donje linije kôda jednaka 12.

```
>>> len(containers)

'12'
```

Postoji jedan mali problem kod grafičkih kartica koje su trenutno navedene na NewEgg.com, a to je da nemaju sve jednake stavke. Npr. neke grafičke kartice nemaju navedenu cijenu, neke pak nemaju navedenu procjenu, itd. U svrhu objašnjavanja rada Beautiful Soupa, u ovom slučaju ću se baviti samo stvarima koje su im zajedničke, dok se za stvari koje im nisu zajedničke zapravo radi ista stvar, samo se kôd malo modificira pomoću uvjetnih naredbi if i else.

Ako želim dohvatiti kontejner 1. grafičke kartice, to ću učiniti na sljedeći način:

```
>>> containers[0]
```

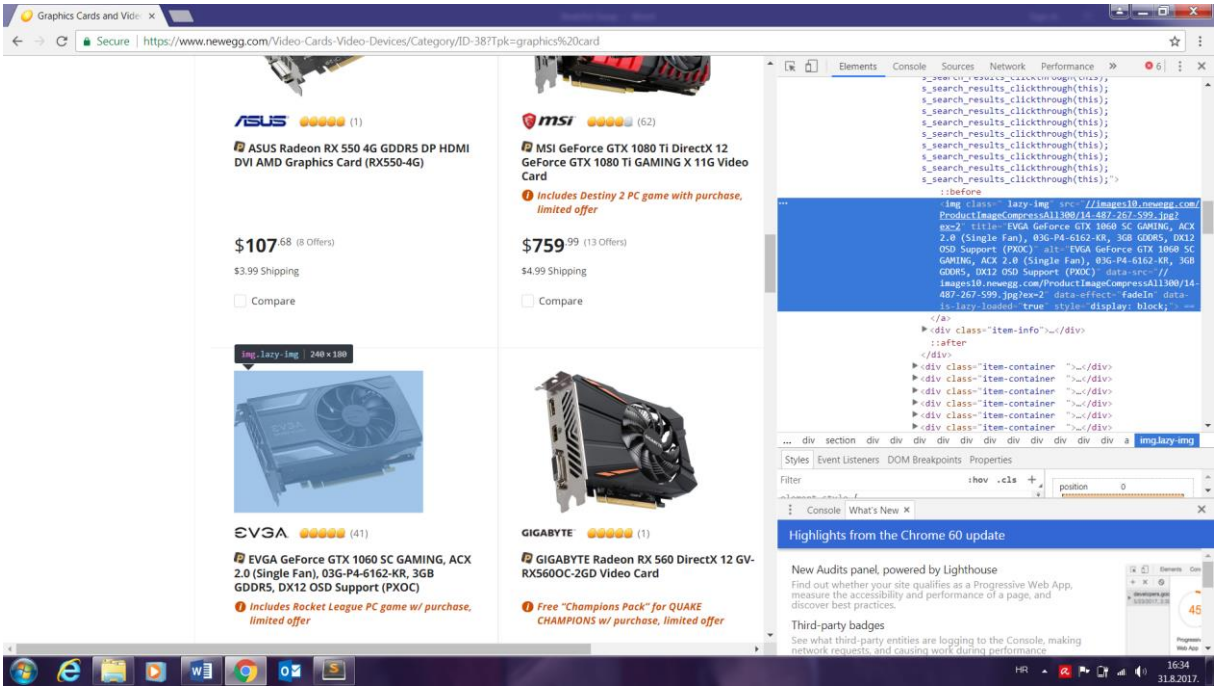
Pomoću toga, dohvatiti ću čitavi kontejner prve grafičke kartice i sav HTML tekst koji se u njemu nalazi. Ako gornju liniju kôda spremim u varijablu `container`,

```
>>> container = containers[0]
```

Na taj način spremam sav HTML tekst koji predstavlja prvu grafičku karticu u varijablu container. Npr. ako želim dohvatiti `<a>` HTML oznaku, to ću činiti ovako:

```
>>> container.a
```

Na taj način dohvaćam sve što se nalazi u HTML tekstu unutar oznake `<a>`, a to je u ovom slučaju slika grafičke kartice.



Slika 7 - <a> oznaka unutar koje se nalazi HTML tekst za sliku grafičke kartice

Međutim, slika nije ono što mi treba kako bi dohvatio podatke o samoj grafičkoj kartici. Za početak, treba mi ime proizvođača grafičke kartice. To ime dohvaćam na isti način na koji sam gore dohvatio sliku. Koristim HTML oznake kako bih "putovao" po samom tekstu, odnosno ručno parsiram po DOM stablu (za sada, kasnije će se to činiti automatski).

Ime grafičke kartice dohvaćam na sljedeći način:

```
>>> container.div.div.a
```

Nakon pokretanja ove linije kôda, dobije se:

```
<a href="https://www.newegg.com/EVGA/BrandStore/ID-1402"
class="item-brand">

<img class=" lazy-img"
src="//images10.newegg.com/Brandimage_70x28//Brand1402.gif"
title="EVGA" alt="EVGA" data-
src="//images10.newegg.com/Brandimage_70x28//Brand1402.gif" data-
effect="fadeIn" data-is-lazy-loaded="true" style="display: block;">
</a>
```

Slika 8 - HTML tekst kontejnera prve grafičke kartice unutar kojeg se nalazi atribut "title" na dnu samog HTML teksta

Dakle, sada se nalazim u `<a>` oznaci koja u sebi sadržava `<img>` oznaku unutar koje se nalazi atribut `title` koji označava naslov, odnosno ime proizvođača grafičke kartice, u ovom slučaju je to EVGA. Ono što je prikazano na samoj web stranici je zapravo link [https://www.newegg.com/EVGA/BrandStore/ID-1402] koji vodi na tu stranicu, ali naslov tog linka je upravo EVGA, a to je ono što mi treba. Konačno, da bi dohvatio ime proizvođača grafičke kartice, treba mi

```
>>> container.div.div.a.img
```

kako bih se pozicionirao unutar <img> oznake unutar koje se pak nalazi ono što mi treba, a to je naslov, tj. ime proizvođača grafičke kartice. Ime dohvaćam na način da koristim uglate zagrade, kao u slučaju kada koristim npr. indeks kontejnera kojeg želim dohvatiti.

```
>>> container.div.div.a.img[“title“]
'EVGA'
```

Sada kada sam dohvatio ime proizvođača grafičke kartice, treba napraviti pelju koja će dohvatiti imena proizvođača svih ostalih grafičkih kartica koje se nalaze na web stranici NewEgg.com. Naravno, ovo se može upotrijebiti za bilo koju web stranicu, ali u ovom slučaju, mi se bavimo gore navedenom.

Petlja:

```
For container in containers
        brand = container.div.div.img[“title“]
```

Linija kôda `brand = container.div.div.img[“title“]` predstavlja spremanje imena proizvođača grafičkih kartica u varijablu koju sam nazvao `brand`, što na hrvatskom znači marka, ali pošto cijelo vrijeme koristim englesku notaciju unutar kôda, tako ću nastaviti i dalje.

Sada mi treba ime same grafičke kartice, kako bi se mogla razlikovati od ostalih grafičkih kartica koje recimo pripadaju istom proizvođaču. U ovom slučaju dolazim do malog problema, a to je da HTML oznaka koja mi treba, a to je `<a class=“item-title“>`, ne može biti dohvaćena s `>>> container.div.div.a`. U teoriji i na prvi pogled, izgleda da bih to i mogao napraviti pošto je `<a class=“item-title“>` ugrađena *(eng. embedded)* u HTML tekstu upravo na takav način, no ono što bi linija kôda `>>> container.div.div.a` napravila je sljedeće:

```
<a href="https://www.newegg.com/EVGA/BrandStore/ID-1402"
class="item-brand">

<img class=" lazy-img"
src="//images10.newegg.com/Brandimage_70x28//Brand1402.gif"
title="EVGA" alt="EVGA" data-
src="//images10.newegg.com/Brandimage_70x28//Brand1402.gif" data-
effect="fadeIn" data-is-lazy-loaded="true" style="display: block;">

</a>
```

Dakle, vratilo bi se natrag na `brand`, odnosno proizvođača grafičke kartice, a to je već riješeno. Može se reći da je ova naredba vratila krivu `<a>` HTML oznaku. Ono što mi treba je ponoviti postupak koji sam gore već naveo, a to je ponovno iskoristiti funkciju `findAll ()`.

```
>>> title_container = container.findAll (“a“, {“class“:“item-
title“})

>>> title_container

<a
href="https://www.newegg.com/Product/Product.aspx?Item=N82E168144873
21&amp;ignorebbr=1" class="item-title" title="View Details"><i
class="icon-premier icon-premier-xsm"></i>EVGA GeForce GTX 1060 SC
GAMING, ACX 2.0 (Single Fan), 03G-P4-6162-KR, 3GB GDDRS, DX12 OSD
Support (PXOC)

</a>
```

Iz ovoga se ponovno vidi koliko je Beautiful Soup moćan alat za dohvaćanje željenih podataka. Sve što je bilo potrebno je jedna linija kôda da se dohvati željeni dio traženog elementa.

Treba primijetiti kako se samo ime grafičke kartice ne u `<i>` HTML oznaci, kako se na prvi pogled čini, već je ona tekst koji se nalazi u `<a>` oznaci. Prema tome, njezinom imenu pristupamo na sljedeći način te podatke koje dobijem pomoću ove linije kôda spremam u varijablu koju ću nazvati `product_name`:

```
>>> product_name = title_container.text

'EVGA GeForce GTX 1060 SC GAMING, ACX 2.0 (Single Fan), 03G-P4-6162-
KR, 3GB GDDRS, DX12 OSD Support (PXOC)'
```

Sljedeći podatak koju imaju sve grafičke kartice na samoj web stranici je dostava *(eng. shipping).* Ponovno koristim funkciju `findAll ()`, samo što se dostava unutar HTML teksta elementa, tj. grafičke kartice, sada nalazi u `<li>` (najlakše se pronađe ako se klikne desnim klikom na dostavu ili bilo koju drugu stavku traženog elementa te zatim klikom na pregled *(eng. inspect)* koji automatski odvede na mjesto gdje se to nalazi u HTML tekstu) oznaci pa stoga koristim:

```
>>> container.findAll (“li“, {“class“: “price-ship“})
```

Kako ne bih izgubio dobivene podatke, rezultate spremam u varijablu:

```
>>> shipping_container = container.findAll (“li“, {“class“: “price-ship“})
```

Nakon toga, testiram ono što sam napravio gore:

```
>>> shipping_container
<li class="price-ship">
        Free Shipping
    </li>
```

Za moje željene podatke, a to je u ovom slučaju dostava, ne trebaju mi `<li>` HTML oznake, nego mi samo treba tekst, odnosno “Free Shipping“. Kako bi dohvatio samo tekst, koristim istu stvar koju sam koristio kada sam želio dohvatiti ime grafičke kartice, a to je:

```
>>> shipping.container.text
'\r\n        Free Shipping\r\n'
```

Ovdje se primijeti kako kod dobivenog rezultata dobijem \r i \n oznake. \r je oznaka koja u HTML-u označava povratnu vrijednost nečega *(eng. return),* a \n označava novi redak *(eng. new line).*

Kako bih se riješio tih oznaka i praznog mjesta koje mi ne treba, a treba mi samo tekst, malo ću očistiti dobiveni rezultat sljedećom linijom kôda:

```
>>> shipping = shipping.container.text.strip()
'Free Shipping'
```

`Strip ()` funkcija vraća kopiju stringa, u ovom slučaju vrijednost dostave grafičke kartice, bez ikakvih dodatnih znakova i praznih mjesta prije ili poslije stringa (Stack Overflow, 2017). Također, rezultate gornje linije kôda spremio sam u varijablu `shipping`, koji predstavlja dostavu.

Ono što sada želim je vidjeti stvarne rezultate dosadašnjeg dijela kôda te ih želim ispisati. U dosadašnjem dijelu, ono što sam radio u Anacondi, to sam kopirao u SublimeText 3. Sada u prijeći samo u SublimeText 3 jer mi on treba kako bi na što efektivniji način dobio tražene i ispisane rezultate te ću na kraju krajeva i pomoću njega moći sve ispisati u .csv datoteci, tj- u Excelu. To činim na sljedeći način:

```
print (“brand:  “ + brand)
print (“product_name:  “ + product_name)
print (“shipping:  “ + shipping)
```

Prije nego što provjerim da li će petlja raditi, tj. da li će raditi sve što sam do sada napravio, prvo ću spremiti *(eng. save)* kôd iz SublimeTexta 3. To spremam pod Python datoteku koju ću nazvati Moj_Scraper.py.

Sljedeće što ću učiniti jest otvoriti novi naredbeni prozor. Taj novi naredbeni prozor sadržava istu putanju kao kod onog naredbenog prozora u kojoj sam do sada radio Python, samo što je u njemu bila Anaconda. Pošto sam tek otvorio novi naredbeni prozor, on ne zna da želim raditi u Pythonu te mu zato moram reći da “otvori“ Python, te odmah do njega upišem ime svog web strugača, tj. skripta iz uređivača teksta SublimeText 3.

```
C: \Users\Bubara\Desktop\webscrape> python Moj_Scraper.py
```

Nakon pokretanja te linije kôda, u naredbenom prozoru izbacile su se sve preostale grafičke kartice i svi podaci o njima koje sam želio dohvatiti i kojima sam se u dosadašnjem dijelu rada bavio. Ono što sada trebam napraviti jest prebaciti dobivene rezultate u .csv datoteku koju nakon toga mogu otvoriti u Excelu.

Prijelaz dosadašnjeg kôda u .csv datoteku čini se na sljedeći način (unutar SublimeTexta 3):

```
filename = “graficke_kartice.csv“
f = open(filename, “w“)
headers = “brand, product_name, shipping\n“
f.write(headers)
```

Ovaj dio kôda nalazi se iznad for petlje koju sam koristio i koju i dalje koristim kako bih dohvatio podatke o svakoj grafičkoj kartici koja se nalazi na NewEgg.com.

```
16
17 filename = "products.csv"
18 f = open(filename, "w")
19
20 headers = "brand, product_name, shipping\n"
21
22 f.write(headers)
23
24 for container in containers:
25     brand = container.div.div.a.img["title"]
26
27     title_container = container.findAll("a", {"class":"item-title"})
28     product_name = title_container[0].text
29
30     shipping_container = container.findAll("li", {"class":"price-ship"})
31     shipping = shipping_container[0].text.strip()
32
33     print("brand: " + brand)
```

Slika 9 - prikaz dijelova kôda

Ono što sada želim jest “reći“ kôdu da svaki put kada dođe do kraja petlje, da ispiše podatke druge grafičke kartice. Ono što dio kôda koji se odnosi na ispis *(eng. print)* imena proizvođača, imena same grafičke kartice i dostave zapravo radi jest da on jednostavo ispiše te tri stvari.

Međutim, ono što treba napraviti jest te tri stvari spojiti u jedno tako da te tri stvari zajedno čine jednu stavku unutar Excel tablice.

```
f.write (brand + product_name + shipping + “\n“)
f.close ()
```

“\n“ se koristi kao razdvojnik *(eng. deliminator)*, kako bi se svaka druga grafička kartica ispisala u novu stavku unutar Excela.

`f.close ()` se koristi kako bi se datoteka zatvorila, jer kada bi je ponovno pokušali otvoriti, to ne bi bilo moguće upravo zato što je već neka datoteka, nebitno je li ona ista ta datoteka ili ne, otvorena.

Nakon toga, spremam kôd, pokrećem skript u SublimeText-u 3 i automatski se stvara .csv datoteka u Excelu koju mogu otvoriti i u koju su upisani svi dohvaćeni podaci o svim grafičkim karticama koje se trenutno nalaze na web stranici NewEgg.com.

	A	B	C	D
1	brand	product_name	shipping	
2	EVGA	EVGA GeForce GTX 1060 SC GAMING| ACX 2.0 (Single Fan)| 03G-P4-6162-KR| 3GB GDDR5| DX12 OSD Support (PXOC)	Free Shipping	
3	XFX	XFX Radeon RS RX 480 DirectX 12 RX-480P836BM Video Card	Free Shipping	
4	GIGABYTE	GIGABYTE GeForce GTX 1070 DirectX 12 GV-N1070WF2OC-8GD Video Cards	Free Shipping	
5	Sapphire	SAPPHIRE NITRO+ Radeon RX 470 100407NT+4GOCL Video Card	$4.99 Shipping	
6	MSI	MSI Radeon RX 480 DirectX 12 RX 480 ARMOR 4G OC Video Card	Free Shipping	
7	ASUS	ASUS GeForce GTX 1070 DUAL-GTX1070-O8G Video Card	Free Shipping	
8	Sapphire	SAPPHIRE NITRO Radeon RX 460 100409NT-4GOCL Video Card	$4.99 Shipping	
9	XFX	XFX R9-FURY-4QFA RADEON R9 FURY X 4GB HBM Liquid Cooled 4096-Bit PCI Express 3.0 CrossFireX Support Video Card	Free Shipping	
10	EVGA	EVGA GeForce GTX 1070 SC GAMING ACX 3.0 Black Edition| 08G-P4-5173-KR| 8GB GDDR5| LED| DX12 OSD Support (PXOC)	Free Shipping	
11	ASUS	ASUS ROG Radeon RX 480 STRIX-RX480-O8G-GAMING Video Card	Free Shipping	
12	GIGABYTE	GIGABYTE GeForce GTX 1080 DirectX 12 GV-N1080D5X-8GD Video Cards	Free Shipping	
13	ZOTAC	ZOTAC GeForce GTX 1060 Mini| ZT-P10600A-10L| 6GB GDDR5 Super Compact	$4.99 Shipping	
14				

Slika 10 - konačni ispis podataka o grafičkim karticama unutar Excela

5. Zaključak

U ovom radu opisan je detaljan rad Pythonove biblioteke Beautiful Soup koja služi za dohvaćanje podataka iz HTML i XML datoteka. Način rada koji opisuje rad same biblioteke Beautiful Soup naziva se web struganje, čime čitav internet postaje baza podataka iz koje se mogu dohvatiti željeni podaci. Ova Pythonova biblioteka omogućuje nekoliko jednostavnih metoda za navigaciju, pretraživanje i modificiranje stabla parsiranja koje se odnosi na raščlambu određenog dokumenta ili web stranice te dohvaćanja onih informacija koje tražimo.

Kako bi uopće mogli raditi unutar Pythonove bibloteke Beautiful Soup, potrebno ju je prvo instalirati unutar sučelja *command prompt* ili neke druge platforme koja podržava Python i njegove dodatke. Također, bitno je poznavati HTML i znati se kretati po hijerarhiji njegovih oznaka s ciljem dohvaćanja željenog dijela samog HTML teksta te na kraju i podataka koje se traže unutar neke određene web stranice. Osim poznavanja Pythona i HTML-a, ono što je najbitnije kod web struganja je znati točno koje podatke želimo dohvatiti jer ako to ne znamo, nećemo znati što trebamo tražiti unutar HTML-a i koje oznake trebamo provjeravati kako bi došli do onoga što se traži. Izbor kretanja po DOM stablu svodi se na dva dijela: krenuti od općeg prema pojedinačnom ili obrnuto. Uobičajena praksa je krenuti od općeg i kretati se prema pojedinačnom, a razlog je brzina, učinkovitost te automatizacija koja bi se mogla prenijeti na druge parsere ili web strugače.

Iz predstavljenog se može vidjeti koliko je Beautiful Soup moćan alat i moje mišljenje je da će doći još neke biblioteke, ne samo u Pythonu, koje će web struganje činiti puno lakšim i jednostavnijim nego što trenutno jest. Trenutno, jedini nedostatak Beautiful Soupa je brzina. Aplikacijsko programsko sučelje *(eng. application programming interface)* lxml je, prema recenzijama navedenima na Stack Overflowu još uvijek jedan od najboljih parsera. Što se tiče samog parsiranja, lxml i html5lib prednjače pred Beautiful Soupom što se tiče brzine, međutim Beautiful Soup je pouzdaniji što se tiče kretanja i traženja traženih elemenata po DOM stablu. Samo je pitanje vremena kada će, i to ne samo Beautiful Soup, već i novi parseri i web strugači, biti sve više prilagođeniji korisnicima *(eng. user friendly)* koji nisu upućeni u informatiku.

Može se doći do svakog djelića podataka, elemenata, stavki, bilo čega što je nekome potrebno. Ono što je najbolje od svega, svatko ga može napraviti kod kuće sa malo znanja iz HTML-a i Pythona te malo logike programiranja. Tehnologija napreduje, napredovat će i ljudi te će sve veći broj ljudi koristiti ovu stranu računalne tehnologije i to u potpunosti podržavam

6. Literatura i izvori:

1. Sweigart, A. (2005) *Automate the Boring Stuff with Python.* No Starch Press, [Internet], <raspoloživo na: http://www.onlineprogrammingbooks.com/automate-boring-stuff-python/>, [pristupljeno: 25. kolovoza].

2. Romano, F. (2015) *Learning Python.* Packt Publishing, [Internet] <raspoloživo na: https://www.packtpub.com/packt/free-ebook/learning-python>, [pristupljeno: 26. kolovoza, 2017.].

3. Richardson, L. (2017.), *Beautiful Soup Documentation*,[Internet], posjećeno: kolovoz, 2017. <raspoloživo na: https://www.crummy.com/software/BeautifulSoup/>, [pristupljeno: 24. kolovoza].

4. Erikkson, B., 13.1.2012., [Fotografija], *The Document Object Model used to access objects in web pages with eg. Javascript*, [Internet], <raspoloživo na: https://commons.wikimedia.org/wiki/File:DOM-model.svg>, [pristupljeno: 26. kolovoza, 2017.].

5. *Web Scraping Service*, n.d., [Fotografija], [Internet], <raspoloživo na: http://webdata-scraping.com/>, [pristupljeno: kolovoz, 2017].

6. NewEgg.com, [Internet], <raspoloživo na: https://www.newegg.com>, [pristupljeno: 26. kolovoza, 2017.].

7. Anaconda.com, [Internet], <https://www.anaconda.com/distribution/>, [pristupljeno: 24.kolovoza, 2017.].

8. SublimeText 3, <https://www.sublimetext.com/3>, [pristupljeno: 24. kolovoza, 2017.].

9. Wikipedia, [Internet], <https://en.wikipedia.org/wiki/Python_(programming_language)>, [pristupljeno: 25. kolovoza, 2017.].

10. MDN Web Docs, [Internet], <raspoloživo na: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model>, [pristupljeno: 26. kolovoza, 2017.].

11. Webhose, [Internet], <raspoloživo na: http://blog.webhose.io/2014/11/27/vertical-aggregation-pattern-matching-crawlers/>, [pristupljeno: 26. kolovoza, 2017.].

12. Stack Overflow, [Internet], <raspoloživo na: https://stackoverflow.com/questions/13783934/what-does-s-strip-do-exactly>, [pristupljeno: 26. kolovoza, 2017.].

13. Duong, P., [Internet], <raspoloživo na: https://www.linkedin.com/in/phuchduong>, [pristupljeno: 24. kolovoza, 2017.]

14. Data Science Dojo, [Internet], <raspoloživo na: https://datasciencedojo.com/>, [pristupljeno: 24. kolovoza, 2017.]