Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

Seminar 2 - Ekstrakcija podatkov

Anja Hrovatič, Jan Fekonja in Nina Vehovec

5. Maj, 2019

1 Uvod

Cilj seminarske naloge [1] je bilo implementirati tri različne metode za ekstrakcijo podatkov iz spleta. Metode so bile: ekstrakcija podatkov s pomočjo regularnih izrazov, potem s pomočjo *XPath* in nenazadnje še z algoritmom *RoadRunner*. Zadnje metode nismo implementirali. Pravilnost metod smo preverili na treh različnih tipih strani, kjer smo imeli za vsak tip strani dve vzorčni strani, ki sta imeli podobne strukturo in atribute. Za implementacijo smo uporabili programski jezik Python. Izhod metod smo zapisali na standardni izhod in sicer v obliki formata JSON. Zaradi lažjega pregleda izhoda metod smo rezultate tudi shranili v svojo datoteko *output*.

2 Izbrani spletni strani

Za dodatni dve spletni strani smo izbrali dve strani iz domene *ebay.com*, ker vsebujeta seznam elementov(izdelkov) in na strani lahko nastavimo število elementov, ki jih želimo prikazati na strani. Prav tako imata strani skoraj enako strukturo. Tako zadostimo pogojem, ki so bili podani v seminarski nalogi. Prva stran, ki je prikazana na sliki 1 vsebuje seznam slušalk in njihove podrobnosti, druga spletna stran pa je prikazana na sliki 2 in vsebuje seznam prenosnih računalnikov. Prva stran vsebuje 25 elementov, računalnikov na drugi strani pa je 50.

Na sliki 1 so tudi označeni atributi oz. podatki, ki smo se jih odločili ekstrahirati iz podanih strani. Za vsak izdelek smo poiskali naslov, atribut *Title*, stanje izdelka, atribut *ItemCondition* torej ali je nek izdelek nov, že uporabljen, prenovljen, ..., nato smo poiskali ceno izdelka, atribut *Price*, ki je bila lahko podana številka ali pa je bila podana kot interval med dvema cenama. Pri kupvanju izdelkov je pomembno tudi kje se izdelek trenutno nahaja, atribut

ItemLocation in cena dostave izdelka do nas, atribut *ShippingCost*. Poiskali smo tudi način kupovanja, atribut *BuyingMethod*, torej nekatere izdelke lahko kupimo takoj, nekateri so trenutno za nas najboljša izbira, nekateri so na dražbi in zanje lahko samo damo ponudbo. Poleg vseh teh atributov pa smo se odločili poiskati še privlačnost izdelka, atribut *ItemHotness*, ki nam pove eno pozitivno lastnost o izdelku, ki si ga ogledujemo. Privlačnost je lahko število prodanih izdelkov, število ljudi, ki si ogleduje ta izdelek, potem dejstvo da ima izdelek popust, ... Lahko se zgodi tudi, da izdelek ne vsebuje tega atributa.

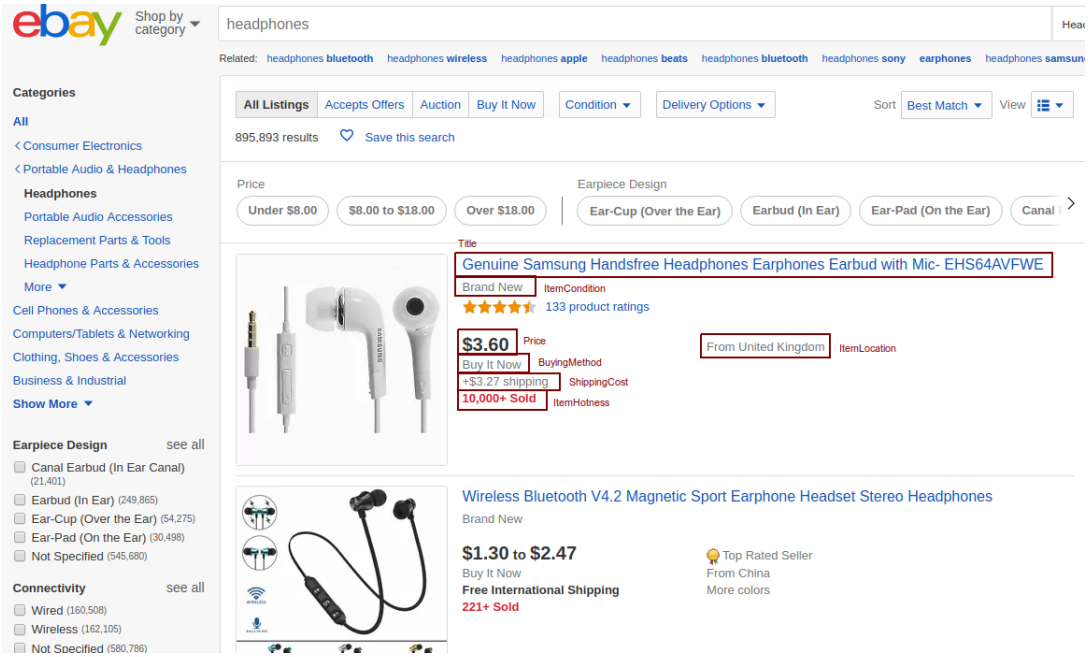

Figure 1: Izbrani atributi elementov, ki smo jih ekstrahirali, so označeni na sliki

3 Implementirane metode

Implemantirali smo metode za ekstrakcijo podatkov s pomočjo regularnih izrazov in *XPath* izrazov. Za vsak tip strani smo napisali eno metodo, ki je uporabljala regularne izraze in eno, ki je uporabljala *XPath* izraze. Obe sta na vhodu prejeli html dokument in vrnili podatke v formatu *JSON*.

3.1 Spletna domena Rtvslo.si

Iz spletne domene Rtvslo.si smo morali pridobiti avtorja, datum objave, naslov, podnaslov, najpomembnejše besedilo članka(*angl. lead*) in vsebino članka. Posamezni podatkovni objekt smo pridobili tako, da smo v HTML dokumentu poiskali

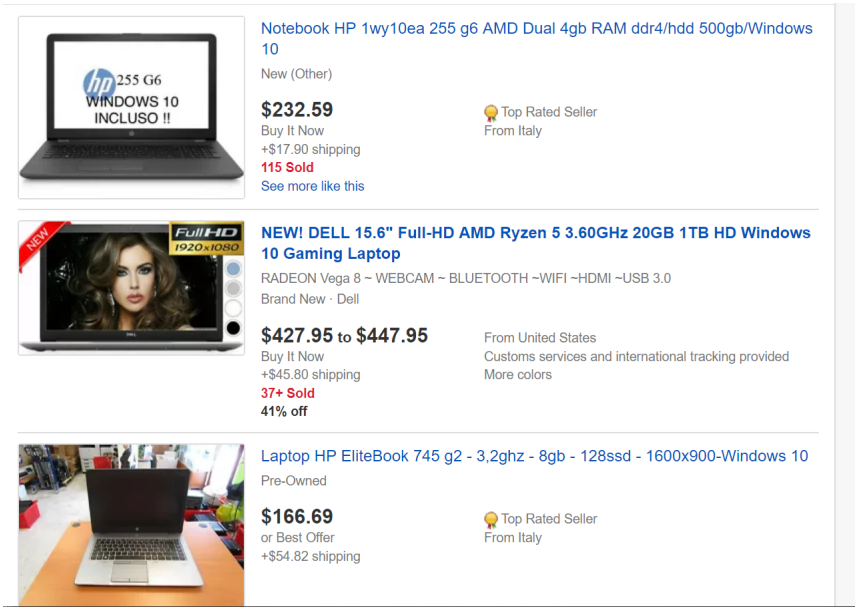

Figure 2: Posnetek druge spletne strani, kjer so elementi prenosni računalniki

HTML značko, ki je vsebovala željeno vsebino ter jo referencirali z imenom razreda. Pri pridobivanju naslova smo morali vzeti poleg značke <h1>, še značko <header>, saj je ta vsebovala ime razreda, tako je bil naslov bolj enolično določen. Paziti smo morali, da značka ni označevala tudi neželjene vsebine, zato smo včasih vzeli več značk.

3.1.1 Metoda z regularni izrazi

Ekstrakcijo podatkov smo izvedli s pomočjo regularnih izrazov [2] kot lahko vidimo v prikazu kode 1 Zopet smo poiskali ustrezne HTML značke ali več značk ter njihove razrede. Da smo iz značk pridobili besedilo smo večinoma uporabljali *(.*)*, torej operatorja ' in *, torej včasih še ?. Želeli smo, da se poljuben znak, ki ni končna vrtica, tabulator... znotraj značk ponovi poljubnokrat, dokler ne najdemo zadnje ponovitve končne značke. ? pa smo uporabljali, če smo želeli, da je ujemanje, kar se da majhno, torej da zajamemo vsebino do prve ponovitve končne značke. Vsebino članka smo pridobili tako da smo poiskali vse <p> značke, ki predtem niso vsebovale [3] znaka za presledek *angl. whitespace character*. Znotraj <p> značk tudi nismo dovolili značk <iframe>, dovolili pa smo nove vrstice. Pri vsaki <p> znački, smo želeli, da je ujemanje kar se da majhno, totrej smo uporabili ?, da smo našli prvo končno značko. Značke
, in <sub> smo naknadno odstranili iz besedila.

```
1       #Title
```

```
    regex_title = r"<header class=\"article-header\">(.|\n)*?<h1>(.*?)<\/h1>"
    match = re.compile(regex_title).search(pageContent)
    title = match.group(2)

    #Author and timestamp
    regex_author_timestamp = r'<div class="author-timestamp">\s+<strong>(.*?)<\/strong>\|\s+([0-9]{2}\.\s+[a-z]*\s+[0-9]{4}\s+ob\s+[0-9]{2}:[0-9]{2})'
    match = re.compile(regex_author_timestamp).search(pageContent)
    author = match.group(1)
    timestamp = match.group(2)

    #Subtitle
    regex_subtitle = r'<div class="subtitle">(.*)</div>'
    match = re.compile(regex_subtitle).search(pageContent)
    subtitle = match.group(1)

    #Lead
    regex_lead = r'<p class="lead">(.*)</p>'
    match = re.compile(regex_lead).search(pageContent)
    lead = match.group(1)

    #Content
    content = " "
    regex_content_p = r'(?<!\s)<p( class="Body")?>(?!<iframe.*?</iframe>)((.|\n)*?)</p>'
    matches = re.finditer(regex_content_p, pageContent)
    for match in matches:
        content = content + match.group(2).replace("<br>", " ").replace("<strong>", " ").replace("</strong>"," ").replace("<sub>"," ").replace("</sub>", " ")
    # removing the first blank space that I generated
    content = content[1:]
```

Listing 1: Regex za Rtvslo.si

3.1.2 Metoda z izrazi XPath

Ekstrakcijo podatkov smo izvedli z *XPath* kot lahko vidimo v prikazu kode 2. Poiskali smo ustrezne HTML značke in njihove razrede. Da smo iz HTML značk pridobili besedilo, smo uporabili še funkcijo *text()*. Vsebino članka smo pridobili tako, da smo izvedli operacijo unija (|) med večimi *XPath* izrazi. Vsebina je bila večinoma znotraj značk <p>, vendar je bila v nekaterih primerih poleg značke <p> vgnezdena še v značkah ali <sub>. Tako smo izvedli ekstrakcijo celotne vsebine.

```
    # author
```

```
31      author = str(tree.xpath('//div[@class="author-name"]/text()
            ')[0])
32
33      # published time
34      publishedTime = str(tree.xpath('//div[@class="publish-meta
            "]/text()')[0])
35      publishedTime = publishedTime.replace('\n\t\t', ' ')
36      # title
37      title = str(tree.xpath('//header[@class="article-header"]//
            h1/text()')[0])
38
39      # subtitle
40      subTitle = str(tree.xpath('//div[@class="subtitle"]/text()'
            )[0])
41
42      # lead
43      lead = str(tree.xpath('//p[@class="lead"]/text()')[0])
44
45      content = tree.xpath('//article[@class="article"]/p/text()
            | //p/strong/text() | //p/sub/text()')
46      content = ' '.join(content)
```

Listing 2: XPath za Rtvslo.si

3.2 Spletna domena Overstock.com

Za posamezen objekt smo pridobili podatke: naslov, vsebina, začetna cena, cena, prihranek in prihranek v procentih. Posamezni podatkovni objekt smo pridobili tako, da smo v HTML dokumentu poiskali HTML značko ali značke, ki so definirale željeno vsebino ter jo referencirali z imenom razreda, če je bilo to možno. Zopet smo morali paziti, da značka ni označevala tudi neželjene vsebine, zato smo včasih vzeli več značk. Primer te uporabe lahko vidimo pri ekstrakciji naslova objekta, kjer uporabimo značke <td> z atributom "valign", <a> in <b> za določitev naslova.

3.2.1 Metoda z Regularnimi izrazi

V prikazu kode 3 je prikazana implementacija ekstrakcije podatkov z regularnimi izrazi za spletno stran Overstock.com. Vsebino znotraj značk smo zopet večinoma pridobili s pomočjo *(.*)*. Prihranek in procent prihranka smo pridobili z enim regularnim izrazom, vendar smo vsak podatek grupirali posebej. Podatka smo nato ločili s pomočjo funkcije *group()*, ki omogoča da izberemo različne elemente, če smo jih grupirali v regularnem izrazu. Pridobljenim podatkom smo odstranili morebitne prehode v novo vrstico in podvojene presledke.

```
47      List price
48      listprice = []
49      regex_listprice = r'List Price:.*<s>(.*)</s>'
50
```

```
    matches = re.finditer(regex_listprice, pageContent)
    for match in matches:
        listprice.append(match.group(1))

    #Price
    price = []
    regex_price = r'Price:.*<span class="bigred"><b>(.*)</b></
        span>'

    matches = re.finditer(regex_price, pageContent)
    for match in matches:
        price.append(match.group(1))

    #Saving and saving percent
    saving = []
    savingPercent = []
    regex_save = r'You Save:.*<span class="littleorange">(\$
        [0-9\.,]*) \(([0-9]{1,3}%)\)</span>'

    matches = re.finditer(regex_save, pageContent)
    for match in matches:
        saving.append(match.group(1))
        savingPercent.append(match.group(2))


    #Title
    title = []
    regex_title = r'<td valign="top">\s*<a href=".*?"><b>(.*)</
        b></a>'

    matches = re.finditer(regex_title, pageContent)
    for match in matches:
        title.append(match.group(1))

    #Content
    content = []
    regex_content = r'<span class="normal">((.|\n)*?)<br>'

    matches = re.finditer(regex_content, pageContent)
    for match in matches:
        content_pretified = match.group(1).replace("\n", " ").
            replace("  ", " ")
        content.append(content_pretified)
```

Listing 3: Regex za Overstock.com

3.2.2 Metoda z izrazi XPath

V prikazu kode 4 je prikazana implementacija ekstrakcije podatkov z *XPath* za spletno stran Overstock.com. Vsebino znotraj značk smo zopet pridobili s

pomočjo metode *text()*. Prihranek in procent prihranka smo pridobili z enim XPath izrazom. Podatka smo nato ločili s pomočjo funkcije split(), s katero smo pridobljen tekst ločili na dva dela - prihranek in procent prihranka. Pridobljenim podatkom smo odstranili morebitne prehode v novo vrstico.

```
90      #Title
91      title = tree.xpath('//td[@valign="top"]/a/b/text()')
92
93      # ListPrice
94      listprice = tree.xpath('//s/text()')
95
96      # Price
97      price = tree.xpath('//span[@class="bigred"]/b/text()')
98
99      # Saving and SavingPercent
100     save = tree.xpath('//span[@class="littleorange"]/text()')
101
102     saving  = []
103     savingpercent = []
104     for i in range(len(save)):
105         saving.append(save[i].split('(')[0].strip(" "))
106         savingpercent.append(save[i].split('(')[1].strip(" )"))
107
108
109     #Content
110     content = tree.xpath('//span[@class="normal"]/text()')
111
112     for i in range(len(content)):
113         content[i] = content[i].replace('\n', ' ')
```

Listing 4: XPath za Overstock.si

3.3 Spletna domena Ebay.com

Iz spletne domene Ebay.si smo pridobili željene podatke, ki smo jih opisali v 2. poglavju. Posamezni podatkovni objekt smo pridobili tako, da smo v HTML dokumentu poiskali HTML značko ali značke, ki so vnas pripeljale do željene vsebino ter jih referencirali z imenom razreda. Pri atributu *"BuyingMethod"* smo morali poiskati značko, z razredom, ki je označeval da izdelek lahko kupimo, ali pa smo morali poiskati značko z razredom, ki je označeval da je element na dražbi. Noben element ni imel obeh razredov hkrati.

3.3.1 Metoda z regularnimi izrazi

V prikazu kode 5 je prikazana implementacija ekstrakcije podatkov z regularnimi izrazi za spletno stran Ebay.com. Vsebino znotraj značk smo večinoma pridobili s pomočjo *(.*?)*, ker smo želeli pridobiti čim manjše ujemanje, torej ujemanje s prvo končno značko, da smo ločili posamezna elemente. Pri pridobivanju cene izdelka, je cena lahko v obliki od-do. To pomeni, da značka

vsebuje obe ceni nenkrat ali pa samo eno. To smo rešili z uporabo vprašajev in grupiranja. Za pridobitev različnih vrednosti iz poizvede smo uporabili metodo *group()*. Pri poizvedbi za atribut *"BuyingMethod"*, smo uporabili operator *"|"*, da smo v regularni izraz lahko vstavili znački z različnima razredoma. Oba razreda se nista smela pojaviti naenkrat. Pri iskanju vrednosti za privlačnost izdelka *ItemHotness*, pa nismo vedeli kako uporabiti samo en regularni izraz, ker enosatvno, če izdelek privlačnosti nima v HTML dokumentu tudi ni nobenih značk in razredov za to lastnost. Zato smo napisali en regularni izraz zato, da smo poiskali značke s podrobnostimi izdelka, ki lahko tudi vsebuje značko z razredom, ki označuje privlačnost. Nato smo iterirali po pridobljenih elementih in za vsak element preverjali če vsebuje drugi regularni izraz, ki je iskal značko z razredom, ki označuje privlačnost. Če smo našli ujemanje smo element shranili v seznam, drugače smo v seznam dodali prazen string.

```
114 # Titles
115
116     titles = []
117     regex_title = r'<h3 class="s-item__title">(<span class="
            LIGHT_HIGHLIGHT">.*?</span>)?(<span class="BOLD">)?(.*?)
            (</span>)?</h3>'
118
119     matches = re.finditer(regex_title, pageContent)
120     for match in matches:
121         titles.append(match.group(3))
122
123
124     # Item Conditionss
125
126     item_conditions = []
127     regex_itemCondition = r'<span class="SECONDARY_INFO">(.*?)
            </span>'
128
129     matches = re.finditer(regex_itemCondition, pageContent)
130     for match in matches:
131         item_conditions.append(match.group(1))
132
133     # Item Locations
134
135     item_locations = []
136     regex_location = r'<span class="s-item__location s-
            item__itemLocation">(.*?)</span>'
137
138     matches = re.finditer(regex_location, pageContent)
139     for match in matches:
140         item_locations.append(match.group(1))
141
142     # Item prices
143
144     item_prices = []
```

```
145         regex_price = r'<span class="s-item__price">(\$
                [0-9]*\.[0-9]*)(<span class="DEFAULT">)?(.*?)(</span>)
                ?(\$[0-9]*\.[0-9]*)?</span>'
146
147         matches = re.finditer(regex_price, pageContent)
148         for match in matches:
149             # if the price looks like : x$ to y$
150             if (match.group(3) and match.group(5)):
151                 price = match.group(1) + match.group(3) + match.
                        group(5)
152             else:
153                 price = match.group(1)
154             item_prices.append(price)
155
156         # Shippings
157
158         item_shippings = []
159         regex_shipping = r'<span class="s-item__shipping s-
                item__logisticsCost">(<span class="BOLD">)?(.*?)(</span
                >)?</span>'
160
161         matches = re.finditer(regex_shipping, pageContent)
162         for match in matches:
163             item_shippings.append(match.group(2))
164
165         # BuyMethods
166
167         item_buymethods = []
168         regex_buyMethod = r'<div class="s-item__detail s-
                item__detail--primary">((<span class="s-item__bids s-
                item__bidCount">)|(<span class="s-item__purchase-options
                 s-item__purchaseOptions">))(.*?)</span>'
169
170         matches = re.finditer(regex_buyMethod, pageContent)
171         for match in matches:
172             item_buymethods.append(match.group(4))
173
174         # Item Hotness
175
176         items_hotness = []
177         regex_itemHotness = r'<span class="s-item__hotness s-
                item__itemHotness" aria-label="">(<span class="clipped
                ">.*?</span>)?<span class="BOLD NEGATIVE">(.*?)</span></
                span>'
178         regex_all_item_details = r'<div class="s-item__details
                clearfix">.*?</li>'
179
180         items = re.finditer(regex_all_item_details, pageContent)
181         for el in items:
```

```
182         match = re.compile(regex_itemHotness).search(el.group
                (0))
183         if (match):
184             item_hotness = match.group(2)
185         else:
186             item_hotness = ""
187         items_hotness.append(item_hotness)
```

Listing 5: Regex za Ebay.com

3.3.2 Metoda z izrazi XPath

V prikazu kode 6 je prikazana implementacija ekstrakcije podatkov z *XPath* za spletno stran Ebay.com. Pri pridobivanju naslova smo morali paziti, da nismo zajeli besedila, ki se je nahajal znotraj značke z razredom "LIGHT_HIGHLIGHT". Pri pridobivanju cene izdelka, je cena lahko v obliki od-do. To pomeni, da značka vsebuje obe ceni nenkrat, ločuje pa ju še ena značka , ki vsebuje samo besedilo *"to"*, zato smo pridobili besedilo iz , torej obe ceni nenkrat in tudi vmesno besedilo. Tako smo v seznamu dobili tri različne elemente za en podatek. Te tri elemente smo nato s pomočjo programiranja združili v en element. Pri atributu *ItemHotness*, pa se lahko zgodi, da ta podatek ne obstaja in preprosta poizvedba nam vrne manjše število vrednosti kot pa je elementov na strani. To smo rešili tako, da smo iz strani ekstrahirali objekte vseh elementov na strani, tako da smo iskali objekte z razredom *"s-item_ details"*. Poleg tega pa smo tudi ekstrahirali besedilo iz vseh elementov, ki so vsebovali razred, ki je opisoval privlačnost izdelka. Nato smo iz poizvedbe (seznama) odstranili tiste objekte, ki so bili podvojeni, torej pustili smo tiste objekte, za katere nismo našli privlačnosti (besedila). Nato smo iterirali čez seznam, ki ni vseboval podvojenih elementov in kjer objekt ni bil tipa string, smo ga odstranili in namesto njega vstavili prazen string.

```
188 # Titles
189     titles = tree.xpath('//h3[@class="s-item__title"]/text()'
190                         '| //h3[@class="s-item__title"]/span[
                                not(contains(@class, "
                                LIGHT_HIGHLIGHT"))]/text()')
191
192     # Conditions
193     itemsConds = tree.xpath('//span[@class="SECONDARY_INFO"]/
            text()')
194
195     # Locations
196     itemsLocs = tree.xpath('//span[@class="s-item__location s-
            item__itemLocation"]/text()')
197
198     # Shippings
199     itemsShippings = tree.xpath('//span[@class="s-
            item__shipping s-item__logisticsCost"]/text()'
```

```
                                '| //span[@class="s-
                                    item__shipping s-
                                    item__logisticsCost"]/span[
                                    @class="BOLD"]/text()')

    # Buying methods
    # Buying method can be a bit or ability to purchase, that
        is why we extract two diferent elements
    itemsBuyMethods = tree.xpath('//span[@class="s-
        item__purchase-options s-item__purchaseOptions"]/text()'
                                 '| //span[@class="s-item__bids s
                                     -item__bidCount"]/text()')

    # Prices
    itemsPrices = tree.xpath('//span[@class="s-item__price"]/
        span[contains(text(),"to")]/text()'
                             '| //span[@class="s-item__price"]/
                                 text()')

        # span can contain price in a form x$ to y$, both
            prices are in the same span, therefore
        # the following code merges two prices for the same
            item in one element in the list
    length = len(itemsPrices)
    i = 0
    while (i < length):
        if (itemsPrices[i] == " to "):
            element = itemsPrices[i - 1] + itemsPrices[i] +
                itemsPrices[i + 1]
            itemsPrices.pop(i + 1)
            itemsPrices.pop(i)
            itemsPrices.pop(i - 1)
            length -= 2
            itemsPrices.insert((i - 1), element)
            i -= 1
        i += 1

    # Desirabilities or hotness of items

    itemsHotness = tree.xpath('//div[@class="s-item__details
        clearfix"]'
                              '| //span[@class="s-item__hotness
                                  s-item__itemHotness"]/span[
                                  @class="BOLD NEGATIVE"]/text()'
                                  )

    length = len(itemsHotness)
    i = 0
    str_bolean = isinstance(itemsHotness[i], str)
    while (i < length):
```

```
235             if (isinstance(itemsHotness[i], str)):
236                 itemsHotness.pop(i - 1)
237                 length -= 1
238                 i -= 1
239             i += 1
240 
241             # Now we only have one istance of each item, but in
                    difrent formats
242             # Taking the items that are not strings from the list
                    and inserting empty strings instead,
243         i = 0
244         while (i < len(itemsHotness)):
245             if (not isinstance(itemsHotness[i], str)):
246                 itemsHotness.pop(i)
247                 itemsHotness.insert(i, "")
248             i += 1
```

Listing 6: XPath za Ebay.com

4 Izhodi implementiranih metod

Vsi izhodi method se nahajajo v mapi *outputs*. Vsak izhod je v svoji *.txt* datoteki, ki je poimenovana glede na metodo in tip strani, za katero so bili ekstrahirani podatki. Izhodi implementiranih metod se tudi izpišejo na standardni izhod ob zagonu programa.

5 Zaključek

Impelementirali smo dve metodi za ekstrakcijo podatkov, atributov. Pri prvi metodi smo podatke pridobili s pomočjo regularnih izrazov, pri drugi metodi pa smo uporabili *XPath*. Z obema metodama smo uspešno pridobili želene podatke za vse tri tipe strani, torej za vseh šest primerov strani. Bolj so bile strani strukturirane, lažje je bilo pridobiti elemente, še posebno pomaga če imajo značke razrede ali identifikatorje. Vendar je bilo potrebno precej dobro poznati strukturo strani, da smo lahko napisali ustrezne regularne izraze in *XPath* izraze. Pri metodi, kjer smo uporabili *XPath* je bilo izraze napisati lažje, ker ni bilo potrebno poznati direktnih sosedov značk ali paziti na kakršnekoli presledke, nove vrstice ali tabulatorje, kar je bilo pomemeno pri regularnih izrazih. Pri *XPath* je bilo pomembno tudi določiti samo začetne značke za pridobitev vsebine, ki smo jih po določitvi ključnih značk, pridobili kar z metodo *text()*. Pri regularnih izrazi pa je bilo potrebno dololčiti končne značke, kar pa je lahko kar zoprno, če se veliko značk konča z *div*, da smo lahko pridobili vmesno vsebino. Torej uporaba *Xpath* nam omogoča več fleksibilnosti, vendar je še vedno zelo pomembna struktura,kar pa je lahko problem, če ljudje stran dosti spreminjajo. Boljše bi bilo uporabiti kakšno metodo, ki je bolj generična. Dobro bi

bilo tudi implementirati algoritem *RoadRunner*. Lahko bi ga primerjali tudi z implementacijo, ki je objavljena na spletu.

Literatura

[1] Slavko Žitnik, "Programming assignment 2," [Dostopano: 7. 5. 2019]. [Online]. Available: http://zitnik.si/teaching/wier/PA2.html

[2] "Regular expression operations," [Dostopano: 7. 5. 2019]. [Online]. Available: https://docs.python.org/3/library/re.html

[3] "Lookaround," [Dostopano: 7. 5. 2019]. [Online]. Available: https://www.regular-expressions.info/lookaround.html