

Nejc Povšič, Vid Ribič, Luka Bezovšek

2. seminar pri predmetu Iskanje in ekstrakcija podatkov s spleta

MENTORJA: prof. dr. Marko Bajec in doc. dr. Slavko Žitnik

1. Uvod

Za drugi seminar smo imeli nalogo izluščiti podatke iz dveh že določenih domen (Overstock in RTVSLO.si) ter iz ene poljubne domene (sami smo si izbrali Avto.net). Za vsako domeno smo imeli dve dokaj podobni strani, iz katerih smo nato pridobili podatke. Uporabili smo tri metode, ki smo jih spoznali tudi na predavanjih:

1. XPath
2. regularni izrazi (regex)
3. RoadRunner

V tem poročilu bomo predstavili strani, ki smo jih obdelali, opisali metode in funkcije, ki smo jih pri tem uporabili, ter se dotaknili problemov, s katerimi smo se spopadali.

2. Vsebina

2.1. Izbira spletnih strani

2.1.1. Overstock



Primer elementa v seznamu na strani Overstock in zahtevani atributi za ekstrakcijo

Pri spletni strani Overstock smo morali izluščiti naslednje podatke za vse produkte, ki so bili na strani: naslov (**title**), vsebino (**content**), prvotno ceno (**list price**), trenutno ceno (**price**), prihranek (**saving**) in odstotek prihranka (**saving percent**).

2.1.2. RTVSLO.si

The screenshot shows the RTVSLO.si website interface. At the top is a navigation bar with categories: SLOVENIJA, SVET, ŠPORT, KULTURA, ŽIVLJENJSKI SLOG, and SVET ZABAVE. Below this is a 'Testi' section. The main article is titled 'Audi A6 50 TDI quattro: nemir v prestižnem razredu' by Miha Merljak, published on 28. december 2018 at 08:51. The subtitle is 'Test nove generacije'. The lead text states: 'To je novi audi A6. V razred najdražjih in najbolj prestižnih žrebcev je vnesel nemir, še preden je sploh zapeljal na parkirni prostor, rezerviran za izvršnega direktorja.' The main content area features a large image of the Audi A6 50 TDI quattro. To the left of the main image is a 'Poudarki' (Highlights) section with three items. To the right is a 'Zadnje iz sekcije' (Latest from the section) section with four smaller article teasers. At the bottom of the main content area is an 'Oglas' (Advertisement) for a dog.

Primer strani RTVSLO.si in zahtevani atributi za ekstrakcijo

Pri RTVSLO.si smo imeli na voljo dva članka, iz katerih smo morali izluščiti naslednje podatke: avtorja (**author**), čas izdaje članka (**published time**), naslov (**title**), podnaslov (**subtitle**), povzetek (**lead**) in vsebino (**content**).

2.1.3. Avto.net

Mercedes-Benz E-Razred **E 220 d Avantgarde Avt.** **33.990 €** Price

Prodajalec: avtoqram
AVTOGRAM AVTO d.o.o.
Leskova cesta 11
Ljubljana

01 / 810 - 04 44
Avtoqram
040 / 332 - 000
Mila
040 / 678 - 000
Mila

Poslji sporočilo prodajalcu

Osnovni podatki:

1. registracija:	9 / 2016
Letnik:	2016 Year
Starost:	rabljeno / ima garancijo
Tehnični pregled:	9/2020
Prevoženi km:	85379 Mileage
Gorivo:	diesel motor Fuel type
Motor:	1950 ccm, 143 kW (195 KM) Engine
Menjalnik:	avtomatski menjalnik Transmission
Oblika:	limuzina, 4 vr.
Barva:	java metalik Color
Notranjost:	rvava / usnje
Interna številka:	19007
VIN / številka šasije:	WDD2130041A125026
Zgodovina vozila:	CARFAX kliknite za ogled Carfax poročila
Kraj ogleda:	Leskova cesta 11, Ljubljana

Poraba goriva in emisije:

barcelona weather - ikanya Co...	3,9 litrov / 100 km
...	3,6 litrov / 100 km
...	4,3 litrov / 100 km
...	Euro 6

Dodatne možnosti:

- vsi oglasi trgovca
- matljani ponudbo
- nazaj na prejšnjo stran
- o ponudbi obvesti prijatelja
- prijava spornega oglasa
- parkiraj v moj avto.net
- ocenite vrednost vozila

Oglejte si tudi ponudbo:

- iste znamke in modela
- podobne cene

Kupujte varno:

- seznam ukradenih vozil
- register zastavnih pravic
- vpogled v podatke o vozilu
- ogled Carfax poročila

Primer strani avto.net in zahtevani atributi

Poleg podanih dveh spletnih mest, smo si morali izbrati še tretje spletno mesto, pri čemer smo vzeli dve enaki strani, ki pripadata le-temu. V naši skupini smo se odločili, da bomo preiskovali spletne strani domene **avto.net**, ki prikazuje oglase novih in rabljenih vozil na slovenskem trgu. Odločili smo se, da bomo kot primer za preiskovanje vzeli detajlno stran, ki prikazuje oglas vozila. Na tej strani smo izbrali deset atributov, ki jih je potrebno prebrati: znamka in model vozila (**title**), opis (**description**), cena (**price**), podatki o prodajalcu (**seller**), letnik izdelave (**year**), število prevoženih kilometrov (**mileage**), tip goriva (**fuel type**), podatki o motorju (**engine**), vrsta menjalnika (**transmission**) in barva vozila (**color**). Primeri strani in razčlemba na attribute je prikazana na spodnjih slikah. Pri atributu cena na strani obstaja možnost, da prodajalec redno ceno zniža in ji doda akcijsko ceno. V tem primeru je atribut cena na strani prečrtan, pod njim pa je izpisana vrednost akcijske cene. Za tovrstne primere smo se dogovorili, da poleg atributa cena, prebrani vsebini pripišemo še atribut akcijska cena (**sale price**).

arska	Mehanizacija	Prosti ?as	Deli in oprema	moj.avto.net
car	Karambolirana / nebrezhibna vozila	Oldtimer-ji	Pregled po znamkah	

STNIK-OGROMNO


41.950 €

AKCIJSKA CENA :


29.990 €

price

sale_price



Prodajalec:



AVTOMARKET REBERNIK d.o.o.
Brezno 47
Podvelka

Posebnost na strani avto.net, akcijska cena

3. Implementirane metode

Pregledovanja strani smo se lotili sistematično. Ker smo imeli nalogo pregledati iste strani na več načinov, smo vse metode želeli implementirati na čim bolj generičen in modularen način. Vstopna točka naše naloge je pythonska skripta *start.py*, ki skrbi za uvoz in branje vsebine vseh strani, ter združuje vse načina pregledovanja strani - z uporabo pristopa Xpath in Regex ter s pomočjo metode RoadRunner. Postopke ekstrakcije podatkov smo nato implementirali s pomočjo objektnega programiranja. Za vsakega izmed pristopov smo definirali svoj razred s pripadajočimi metodami, preko instance razreda pa smo nato zagnali pregledovanje vseh prej odprtih strani.

```
pageContent1 = open('pages/overstock.com/jewelry01.html', 'r').read()
pageContent2 = open('pages/overstock.com/jewelry02.html', 'r').read()
pageContent3 = open('pages/rtvslo.si/Audi A6 50 TDI quattro nemir v premijskem razredu - RTVSLO.si.html', 'r').read()
pageContent4 = open('pages/rtvslo.si/Volvo XC 40 D4 AWD momentum suvereno med najboljši v razredu - RTVSLO.si.html', 'r').read()
pageContent5 = open('pages/avto.net/mercedes-benz-e/mercedes-benz-e.html', 'r').read()
pageContent6 = open('pages/avto.net/ford-galaxy/ford-galaxy.html', 'r').read()

test1 = open('./test1.html', 'r').read()
test2 = open('./test2.html', 'r').read()

xpath_parser = XPathParser([
    {
        'type': 'overstock',
        'pages': [pageContent1, pageContent2]
    },
    {
        'type': 'rtvslo',
        'pages': [pageContent3, pageContent4]
    },
    {
        'type': 'avtonet',
        'pages': [pageContent5, pageContent6]
    }
])

xpath_parser.parse_pages()

regex_parser = RegexParser(
    [
        {
            "type": "rtvslo",
            "pages": [pageContent3, pageContent4]
        },
        {
            "type": "overstock",
            "pages": [pageContent1, pageContent2]
        },
        {
            "type": "avto.net",
            "pages": [pageContent5, pageContent6]
        }
    ]
)

regex_parser.parse_pages()

roadrunner = RoadRunner([test1, test2])
roadrunner.start()
```

Vsebina skripte start.py

3.1. Xpath

Za pregledovanje strani z uporabo pristopa Xpath, smo definirali razred *XPathParser*. Ta kot vhodni atribut sprejme seznam vseh strani, ki jih želimo pregledati. Vsak element seznama je zapisan v obliki objekta, ki nosi informacije o straneh istega tipa in vsebuje dva atributa. Prvi označuje tip strani, ki jo moramo pregledati (ločimo 'overstock', 'rtvslo' in 'avtonet'), drugi pa vsebuje seznam vseh strani, ki pripadajo označenemu tipu. Primer inicializacije razreda je prikazan na spodnji sliki.

```

xpath_parser = XPathParser([
    {
        'type': 'overstock',
        'pages': [pageContent1, pageContent2]
    },
    {
        'type': 'rtvslo',
        'pages': [pageContent3, pageContent4]
    },
    {
        'type': 'avtonet',
        'pages': [pageContent5, pageContent6]
    }
])

```

Inicializacija razreda XPathParser

Ko smo ustvarili nov objekt, ki pripada razredu *XPathParser*, moramo s klicem ustrezne metode zagnati ekstrakcijo podatkov iz vseh strani, ki smo jih podali objektu pri inicializaciji. Za ta namen smo v razredu definirali metodo *parse_pages*, ki se sprehodi po celotnem seznamu vseh podanih strani in glede na tip strani kliče ustrezno metodo, ki poskrbi za ekstrakcijo in izpis ustreznih podatkov. Več o ekstrakciji podatkov iz posamezne strani, si bomo pogledali v naslednjih poglavjih.

Preden smo se sploh lotili ekstrakcije podatkov iz posamezne strani, smo vsebino strani ustrezno obdelali, z namenom poenostavitve samega procesa ekstrakcije. Zato smo v sklopu razreda definirali metodo *prettify_content*, ki iz vsebine odstrani vse posebne znake (nova vrstica, tabulator) in poskrbi, da vsi presledki vsebujejo zgolj en znak. Opcijsko smo v funkciji omogočili tudi zamenjavo '
' značk s presledki, saj nam te v nekaterih primerih tekst neustrezno razdelijo na več delov, ki jih Xpath poizvedbe nato tretirajo kot ločene elemente, čeprav se nahajajo znotraj istega elementa. Tako smo v odvisnosti od primera, pri klicu funkcije podali logični parameter *remove_br*, s katerim določamo ali želimo tovrstne znake obdržati ali jih zamenjati.

```

def prettify_content(self, page_content, remove_br):
    content = page_content.replace('\t', ' ').replace('\n', ' ').replace('\r', ' ').replace('&nbsp;', ' ')

    if (remove_br):
        content = content.replace('<br>', ' ')

    return ' '.join(content.split())

```

Metoda, ki poskrbi za olepšanje vsebine pred ekstrakcijo podatkov

Za potrebe pravilnega izpisa strani smo v razredu implementirali metodo *format_text*, ki iz besedila odstrani vse presledke na koncu in na začetku.


```
def format_text(self, text):  
    return text.lstrip().rstrip()
```

Metoda, ki poskrbi za olepšanje besedila pred zapisom v datoteko

3.1.1. Pregledovanje strani Overstock

Primeri strani Overstock so nekoliko starejši od sodobnih spletnih strani. Celotna stran je zgrajena s pomočjo tabel, zelo malo elementov pa vsebuje dodatne attribute, po katerih bi lahko iskali oz. unikatno določili posamezen element. Nekateri elementi sicer vsebujejo atribut class, vendar ta ne določa elementa unikatno (služi le za dodeljevanje generičnih stilov). Iz tega razloga so poizvedbe za elemente tovrstnih strani nekoliko daljše. Ker gre za stran, ki prikazuje seznam zadetkov, smo z eno poizvedbo za specifičen atribut lahko pridobili vrednosti za vse elemente v seznamu. Xpath poizvedba nam namreč vrne seznam vseh zadetkov, ki ustrezajo predpisu. Kot smo že omenili, smo razred xpathparser implementirali tako, da lahko naenkrat sprejme seznam večih strani, na koncu pa vrne vsebino vseh strani v json formatu zapisa. Metoda se tako na začetku sprehodi preko vseh podanih strani in za vsako stran generira XML drevo s klicem metode *html.fromstring*, ki pripada zunanji knjižnici lxml. Tej metodi podamo že olepšano vsebino strani, za kar poskrbi metoda *prettyfy_content*, ki smo jo opisali zgoraj. Nato vse potrebne attribute izluščimo s pomočjo xpath poizvedb. Ker sklepamo, da vsak zadek v seznamu vsebuje vse zahtevane attribute, se z enostavno for zanko lahko sprehodimo po seznamih zadetkov vseh atributov in za vsak izdelek zgradimo objekt, ki je opisan z vsemi zahtevanimi atributi. Posamezen objekt nato dodamo v seznam vseh izdelkov.

```
def parse_overstock(self, pages):  
  
    pages_output = []  
  
    for page in pages:  
  
        page_content = {  
            'type': 'overstock',  
            'items': []  
        }  
  
        # First form an XML tree  
        tree = html.fromstring(self.prettyfy_content(page, False))  
  
        # title  
        title_items = tree.xpath(  
            '/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/a/b/text(.)')  
        # description  
        description_items = tree.xpath(  
            '/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/table/tbody/tr/td[2]/span/text(.)')  
        # listPrice  
        list_price_items = tree.xpath(  
            '/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/table/tbody/tr/td[1]/table/tbody/tr[1]/td[2]/a/text(.)')  
        # price  
        price_items = tree.xpath(  
            '/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/table/tbody/tr/td[1]/table/tbody/tr[2]/td[2]/span/b/text(.)')  
        # saving  
        saving_items = tree.xpath(  
            '/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/table/tbody/tr/td[1]/table/tbody/tr[3]/td[2]/span/text(.)')  
  
        for index, item in enumerate(title_items):  
            item_data = {  
                'title': self.format_text(item),  
                'content': self.format_text(description_items[index]),  
                'listPrice': self.format_text(list_price_items[index]),  
                'price': self.format_text(price_items[index]),  
                'saving': self.format_text(self.split_saving(saving_items[index])[0]),  
                'savingPercentage': self.format_text(self.split_saving(saving_items[index])[1]),  
            }  
  
            page_content['items'].append(item_data)  
  
        self.parsed_output['pages'].append(page_content)  
  
        pages_output.append({'items': page_content['items']})  
  
    self.write_to_file('xpath_overstock', pages_output)
```

Celotna implementacija metode za ekstrakcijo podatkov iz strani Overstock

Ekstrakcija vseh atributov je v tem primeru zelo enostavna, nekoliko bolj kompleksna je ekstrakcija atributov *'saving'* in *'savingPercentage'*, saj sta oba atributa zapisana znotraj enega samega DOM elementa. Iz tega razloga smo implementirali metodo *split_saving*, ki iz teksta posebej izlušči prihranek v denarni enoti in v odstotkih. Prihranek v odstotkih je namreč zapisan v oklepaju, zato moramo tekst razdeliti na besedilo izven in znotraj oklepaja, ter nato ekstrahirati vsak podatek posebej.

```
def split_saving(self, saving):  
    return saving.replace('(', '').replace(')', '').split()
```

Ekstrakcija obeh podatkov o prihranku iz enega tekstovnega elementa

Na koncu pregledovanja vseh strani tipa Overstock, nam metoda zapiše rezultat v json datoteko. Na vrhovnem nivoju se nahaja seznam, ki vsebuje objekte, ki predstavljajo posamezno instanco strani, ki smo jo prebrali. Vsaka stran je predstavljena z objektom, ki vsebuje seznam vseh izdelkov na strani. Vsak izdelek je predstavljen v obliki objekta s pripadajočimi atributi. Primer izpisa podatkov dveh strani tipa Overstock, se nahaja v datoteki *'output/xpath_overstock.json'*.

3.1.2. Pregledovanje strani RTVSLO.si

Pregledovanje strani Rtv slo je implementirano zelo podobno kot pregledovanje strani Overstock. Na enak način se vsebina najprej ustrezno pripravi, nato pa se iz nje izluščijo ustrezni atributi. Ker v tem primeru ne gre za stran s seznamom izdelkov, se na izpisu posamezne instance strani ne generira seznam zadetkov. Prav tako gre za novejšo stran, pri kateri elementi vsebujejo več atributov, zato jih lahko najdemo s krajšimi Xpath poizvedbami, saj nekateri atributi class unikatno označujejo posamezne dele strani, preko katerih nat lahko enostavneje dostopamo do iskanega elementa oz. njegove vrednosti. Primer izpisa podatkov dveh strani tipa RTVSLO.si, se nahaja v datoteki *'output/xpath_rtv slo.json'*.


```

def parse_rtvslo(self, pages):
    pages_output = []

    for page in pages:
        page_content = {
            'type': 'rtvslo',
            'content': {}
        }

        # First form an XML tree
        tree = html.fromstring(self.prettify_content(page, False))

        # author
        author = tree.xpath('//div[@class="author-name"]/text()')

        # publishedTime
        publishedTime = tree.xpath('//*[@id="main-container"]/div[3]/div/div[1]/div[2]/text()')

        # title
        title = tree.xpath('//header[@class="article-header"]/h1/text()')

        # subtitle
        subtitle = tree.xpath('//header[@class="article-header"]/div[@class="subtitle"]/text()')

        # lead
        lead = tree.xpath('//header[@class="article-header"]/p[@class="lead"]/text()')

        # content
        content = tree.xpath('//article[@class="article"]/p/text()')

        item_data = {
            'author': self.format_text(author[0]),
            'publishedTime': self.format_text(publishedTime[0]),
            'title': self.format_text(title[0]),
            'subtitle': self.format_text(subtitle[0]),
            'lead': self.format_text(lead[0]),
            'content': self.format_text(' '.join(content))
        }

        page_content['content'] = item_data
        self.parsed_output['pages'].append(page_content)

    pages_output.append(item_data)

self.write_to_file('xpath_rtvslo', pages_output)

```

Celotna implementacija metode za ekstrakcijo podatkov iz strani RTVSLO.si

3.1.3. Pregledovanje strani Avto.net

Tudi pregledovanje strani Avto.net je implementirano na podoben način kot zgoraj opisani metodi. Ta metoda je načeloma bolj podobna metodi za ekstrakcijo podatkov iz strani RTVSLO.si, saj gre tudi v tem primeru za stran brez seznama zadetkov. Enako kot v prejšnjem primeru, tudi tukaj nekateri atributi unikatno označujejo posamezne dele strani, zato smo XPath poizvedbe lahko ustrezno optimizirali in skrajšali. Na tem mestu smo ob klicu metode *'prettify_content'* omogočili odstranjevanje značk `
` iz vsebine strani, saj so nam te ločevale zapis o prodajalcu po vrsticah, s tem pa nismo mogli pridobiti celotnega zahtevanega besedila. Primer izpisa podatkov dveh strani tipa Avto.net, se nahaja v datoteki *'output/xpath_avto_net.json'*.

```
def parse_avtonet(self, pages):

    pages_output = []

    for page in pages:

        page_content = {
            'type': 'avtonet',
            'content': {}
        }

        tree = html.fromstring(self.prettify_content(page, True))

        # title
        title = tree.xpath('//div[@class="OglasDataTitle"]/h1/text()')

        # description
        description = tree.xpath('//div[@class="OglasDataTitle"]/h1/small/text()')

        # price
        has_action_price = tree.xpath('//p[@class="OglasDataAkcijaCena"]')

        if not has_action_price:
            price = tree.xpath('//p[@class="OglasDataCenaTOP"]/text()')
        else:
            ## action price
            price = tree.xpath('//p[@class="OglasDataStaraCena"]/span/text()')
            sale_price = tree.xpath('//p[@class="OglasDataAkcijaCena"]/span/text()')

        # seller
        seller = tree.xpath('//div[@class="Padding14 Bold"]/text()')

        # year
        year = tree.xpath(
            '//div[@class="OglasWrapper RoundedBottom MarginBTM"]/div[3]/div[@class="OglasDataRight"]/text()')

        # mileage
        mileage = tree.xpath(
            '//div[@class="OglasWrapper RoundedBottom MarginBTM"]/div[6]/div[@class="OglasDataRight"]/text()')

        # fuel type
        fuel_type = tree.xpath(
            '//div[@class="OglasWrapper RoundedBottom MarginBTM"]/div[7]/div[@class="OglasDataRight"]/text()')

        # engine
        engine = tree.xpath(
            '//div[@class="OglasWrapper RoundedBottom MarginBTM"]/div[8]/div[@class="OglasDataRight"]/text()')
```

```

# transmission
transmission = tree.xpath(
    '//div[@class="OglasWrapper RoundedBottom MarginBTM"]/div[9]/div[@class="OglasDataRight"]/text()'

# color
color = tree.xpath(
    '//div[@class="OglasWrapper RoundedBottom MarginBTM"]/div[11]/div[@class="OglasDataRight"]/text()'

item_data = {
    'title': self.format_text(title[0]),
    'description': self.format_text(description[0]),
    'price': self.format_text(price[0]),
    'seller': self.format_text(seller[0]),
    'year': self.format_text(year[0]),
    'mileage': self.format_text(mileage[0]),
    'fuel_type': self.format_text(fuel_type[0]),
    'engine': self.format_text(engine[0]),
    'transmission': self.format_text(transmission[0]),
    'color': self.format_text(color[0])
}

if has_action_price:
    item_data['sale_price'] = sale_price[0]

page_content['content'] = item_data
self.parsed_output['pages'].append(page_content)

pages_output.append(item_data)

self.write_to_file('xpath_avto_net', pages_output)

```

Celotna implementacija metode za ekstrakcijo podatkov iz strani Avto.net

3.2. Regex

Za pregledovanje strani z uporabo regularnih izrazov, smo definirali razred *RegexParser*. Ta za vhodni atribut sprejme seznam vseh strani, ki jih želimo pregledati. Oblika seznama je enaka kot pri *XPathParser*-ju – Vsak element seznama je objekt, ki vsebuje tip strani, ki jo moramo pregledati ('overstock', 'rtvslo' ali 'avtonet'), in seznam strani, ki pripadajo temu tipu. Primer inicializacije razreda je prikazan na spodnji sliki.

```

regex_parser = RegexParser(
[
    {
        "type": "rtvslo",
        "pages": [pageContent3, pageContent4]
    }, {
        "type": "overstock",
        "pages": [pageContent1, pageContent2],
    }, {
        "type": "avto.net",
        "pages": [pageContent5, pageContent6],
    },
]
)

```

Inicializacija razreda RegexParser

Ko smo ustvarili *RegexParser*, izvedemo metodo *parse_pages*. Ta se sprehodi po celotnem seznamu podanih strani in glede na tip strani kliče ustrezno metodo, ki poskrbi za ekstrakcijo in zapis ustreznih podatkov v datoteko formata *.json*.

```
def parse_pages(self):
    for site in self.sites:
        if site["type"] == "rtvslo":
            self.parse_rtvsl0(site["pages"])

        elif site["type"] == "overstock":
            self.parse_overstock(site["pages"])

        elif site["type"] == "avto.net":
            self.parse_avto_net(site["pages"])
```

Klic ustrezne metode za ekstrakcijo podatkov glede na tip strani

3.2.1. Pregledovanje strani Overstock

Vsako stran najprej očistimo. Odstranimo presledke in nove vrstice med začetkom in koncem značk (<, >) ter s pomočjo zunanje knjižnice BeautifulSoup odstranimo vsebino znotraj značk <script>. To storimo zato, ker te vsebujejo tabele, ki so bile prevelik zalogaj za regularni izraz. Po odstranitvi <script> značk z regularnim izrazom izluščimo tabelo posameznih stvari, ki se prodajajo na Overstocku in se čez njih sprehodimo z ukazom *finditer*. S pomočjo BeautifulSoupa izluščimo vsebino in naslov, z dodatnima regularnima izrazoma pa nato še podatke o ceni, znižanju in prihranku. Končni rezultat, ki ga zapišemo v datoteko .json, je seznam stvari. Vsaka stvar je predstavljena s slovarjem, ki podaja informacijo o naslovu, prodajni ceni, ceni, prihranku, prihranku v odstotkih in vsebini.

```
def parse_overstock(self, pages):
    data_to_save = []

    for page in pages:
        page_content = self.remove_script_tags(" ".join(page.split()).replace('> <', '><'))

        regex = r"<td valign='top'><a href='(.*)'>(.*)</a><br/><table><tbody><tr><td valign='top'><table><tbody><tr><td align='left' (.*)><s>(.*)</s></td></tr>"

        items = []

        for item in re.finditer(regex, page_content):
            title = self.get_text(item.group(2))
            prices_info = item.group(3)
            content = self.get_text(item.group(5))

            list_price_regex = r"<tr>(.*)<td align='left' (.*)><s>(.*)</s></td></tr>"
            list_price_match = re.compile(list_price_regex).search(prices_info)
            list_price = self.get_text(list_price_match.group(3))

            price_regex = r"<tr>(.*)<td align='left' (.*)><span class='bigred'>(.*)</span></td></tr>"
            price_match = re.compile(price_regex).search(prices_info)
            price = self.get_text(price_match.group(3))

            saving_regex = r"<tr>(.*)<td align='left' (.*)><span class='littleorange'>(\$\\d\\.\\d)+\\((\\d*)\\)</span></td></tr>"
            saving_match = re.compile(saving_regex).search(prices_info)
            saving = self.get_text(saving_match.group(3))
            saving_percent = self.get_text(saving_match.group(5))

            items.append({
                "title": title,
                "list_price": list_price,
                "price": price,
                "saving": saving,
                "saving_percent": saving_percent,
                "content": content,
            })

        data_to_save.append({
            "items": items
        })

    self.write_to_file("regex_overstock", data_to_save)
```

Celotna implementacija metode za ekstrakcijo podatkov iz strani Overstock

3.2.2. Pregledovanje strani RTVSLO.si

Iz RTV-jevih strani smo najprej odstranili vse `<script>` in `<figure>` značke. Slednje zato, ker so v njih oglasi, ki nas med razčlenjevanjem strani niso zanimali. Pri odstranjevanju značk smo si pomagali z BeautifulSoup-om.

Prvo smo razčlenili header in iz njega izluščili podatke o naslovu, podnaslovu in uvodno besedilo (lead). Nato smo pridobili div z razredom `article-meta`, v katerem so informacije o avtorju in času objave. Nazadnje smo razčlenili še vsebino. Tu ni bilo veliko dela, saj so z odstranitvijo oglasov ostale samo fotografije in besedilo. S pomočjo metode `get_text`, smo izluščili zgolj tekstovni del članka.

Tako pripravljen slovar s podatki o članku smo shranili v .json datoteko.

```
def parse_rtvsl0(self, pages):
    data_to_save = []

    for page in pages:
        page_content = self.remove_tags(" ".join(page.split()).replace('> <', '><'), ["script", "figure"])

        header_regex = r"<header class='article-header'>(.*?)<h1>(.*?)</h1><div class='subtitle'>(.*?)</div>(.*?)"
        header_match = re.compile(header_regex).search(page_content)

        title = header_match.group(2)
        subtitle = header_match.group(3)
        lead = header_match.group(5)

        meta_regex = r"<div class='article-meta'>(.*?)<div class='author-name'>(.*?)</div>(.*?)<div class='published_time'>(.*?)</div>(.*?)"
        meta_match = re.compile(meta_regex).search(page_content)

        author = meta_match.group(2)
        published_time = meta_match.group(4)

        article_regex = r"<article class='article'>(.*?)</article>"
        article = re.compile(article_regex).search(page_content)
        content = self.get_text(article.group(1))

        data_to_save.append({
            "author": author.lstrip().rstrip(),
            "published_time": published_time.lstrip().rstrip(),
            "title": title.lstrip().rstrip(),
            "subtitle": subtitle.lstrip().rstrip(),
            "lead": lead.lstrip().rstrip(),
            "content": content.lstrip().rstrip(),
        })
```

Celotna implementacija metode za ekstrakcijo podatkov iz strani RTVSLO

3.2.3. Pregledovanje strani Avto.net

Tudi pri straneh iz avto.net smo sprva odstranili prazne vrstice in presledke. Prvi div, ki smo ga razčlenili z regularnim izrazom je imel razred `OglasDataTitle`. V njem sta se nahajala podatka o naslovu in podnaslovu oglasa. Nato smo pridobili redno in akcijsko ceno. Slednjo le v primeru, da jo je oglas imel (v našem primeru jo je imel Ford Galaxy, Mercedes Benz pa ne). V divu z razredom `Padding14 Bold` smo našli podatek o prodajalcu. Nato pa je sledilo razčlenjevanje posameznih informacij o prodajanem avtomobilu. Lastnosti avtomobila so bile podane v dveh divih: v prvem je bil zapisan atribut (npr. motor, gorivo, prevoženi kilometri itd.), v drugem pa vrednost za ta atribut. Za lažje pridobivanje želenih atributov, smo jih predhodno obdelali: pretvorili smo jih v male tiskane črke, šumnike smo zamenjali s črkami c, s in z, presledke pa s podčrtaji. Tako smo lahko s preprostim if stavkom pridobili iskane lastnosti: letnik, prevožene kilometre, gorivo, motor, menjalnik in barvo. Kot pri vseh ostalih primerih smo tudi tu končni rezultat zapisali v datoteko .json.


```
def parse_avtonet(self, pages):
    data_to_save = []

    for page in pages:
        page_content = " ".join(page.split()).replace('> <', '><')

        title_regex = r"<div class='OglasDataTitle'><h1>(.*?)<small>(.*?)</small></h1></div>"
        title_match = re.compile(title_regex).search(page_content)
        title = self.get_text(title_match.group(1))
        subtitle = self.get_text(title_match.group(2))

        price_regex = r"<div class='OglasDataCena (.*?)'>(.*?)<p class='(OglasDataStaraCena|OglasDataCenaTOP)'>(.*?)</p></div>"
        price_match = re.compile(price_regex).search(page_content)
        price = self.get_text(price_match.group(4)).lstrip().rstrip()

        seller_regex = r"<div class='Padding14 Bold'>(.*?)</div>"
        seller_match = re.compile(seller_regex).search(page_content)
        seller = self.get_text(seller_match.group(1))

        car_data = {
            "title": title,
            "subtitle": subtitle,
            "price": price,
            "seller": seller,
        }

        sale_price = price_match.group(7)
        if sale_price is not None:
            sale_price = self.get_text(price_match.group(7)).lstrip().rstrip()
            car_data["sale_price"] = sale_price

        data_regex = r"<div class='OglasData'><div class='OglasDataLeft'>(.*?)</div><div class='OglasDataRight'>"

        for item in re.finditer(data_regex, page_content):
            attr = item.group(1).lower().replace(":", "").replace(" ", "_").replace("ž", "z").replace("č", "c").replace("š", "s")
            value = item.group(2)

            if attr == "letnik":
                car_data["year"] = value

            elif attr == "prevozeni_km":
                car_data["mileage"] = value

            elif attr == "gorivo":
                fuel_type = self.get_text(value)
                car_data["fuel_type"] = fuel_type

            elif attr == "motor":
                engine = self.get_text(value)
                car_data["engine"] = engine

            elif attr == "menjalnik":
                car_data["transmission"] = value

            elif attr == "barva":
                color = self.get_text(value)
                car_data["color"] = color

        data_to_save.append(car_data)

    self.write_to_file("regex_avto_net", data_to_save)
```

Celotna implementacija metode za ekstrakcijo podatkov iz strani Avto.net

3.3. RoadRunner

Tudi implementacije postopka RoadRunner, smo se lotili s pomočjo objektnega programiranja. Definirali smo razred RoadRunner, ki, enako kot razreda za Xpath in regularni izraz, sprejme seznam objektov, v katerih so zapisani tip domene in vsebini obeh strani. Na spodnji sliki je prikazana inicializacija objekta, ki pripada razredu RoadRunner.

```
roadrunner = RoadRunner(sites_to_parse)

roadrunner.start()
```

Inicializacija objekta RoadRunner

Po inicializaciji objekta, smo postopek izgradnje ovojnice zagnali s klicem metode *start*. Preden smo se lotili grajenja ovojnice s pomočjo primerjave obeh strani, smo vsebino posamezne strani s pomočjo zunanje knjižnice BeautifulSoup pretvorili v gnezdeno podatkovno strukturo. Implementacija je prikazana na spodnji sliki.

```
page1 = site["pages"][0]
soup1 = BeautifulSoup(page1, 'lxml')
```

Pretvorba vsebine v gnezdeno strukturo z uporabo knjižnice BeautifulSoup

Implementacijo algoritma smo razdelili na tri poglavitne korake. V naslednjih poglavjih si bomo pogledali delovanje in namen posameznega koraka.

3.3.1. Gradnja drevesa

V prvem koraku moramo za vsako stran zgraditi drevo, ki ustreza standardni obliki, ki smo jo določili glede na podatke, ki jih bomo potrebovali pri implementaciji nadaljnjih korakov, hkrati pa smo želeli vsebino zapisati v čim bolj enostavni obliki, tako, da se jo bo dalo čim bolj enostavno brati z rekurzivnimi metodami. Ker pri html vsebini govorimo o gnezdeni obliki podatkov, smo tudi naše drevo načrtovali v tej smeri. Struktura podatkov, ki predstavljajo drevo, se gradi rekurzivno glede na samo strukturo strani. Posamezen element drevesa, je predstavljen z naslednjimi atributi:

```
{
    "tag": <predstavlja dejanski tip DOM elementa>
    "has_closing_tag": <ali je element zaključen s končno značko?>
    "has_text": <ali element vsebuje/obkroža besedilo?>
    "children": [ <seznam podrejenih elementov - z enako strukturo> ]
}
```

Vsakemu elementu drevesa dodali še atributa `id` in `class`, vendar jih pri naši implementaciji nismo potrebovali, zato smo jih predhodno odstranili iz začetne strukture strani.

Za vsako stran smo pričeli z gradnjo drevesa na zgornjem elementu (`<html>`) in končali takrat, ko so se vse rekurzivne zanke zaključile (so bile pregledane vse podrejene veje v DOM strukturi). Primer drevesa enostavne strani je prikazan na spodnji sliki.

```
{
  "tag": "html",
  "id": "",
  "class": [],
  "children": [
    {
      "tag": "body",
      "id": "",
      "class": [],
      "children": [
        { "id": "" },
        { "id": "" },
        {
          "tag": "ul",
          "id": "",
          "class": [],
          "children": [
            {
              "tag": "li",
              "id": "",
              "class": [],
              "children": [
                {
                  "tag": "i",
                  "id": "",
                  "class": [],
                  "children": [],
                  "has_closing_tag": True,
                  "has_text": True
                }
              ]
            },
            {
              "tag": "i",
              "id": "",
              "class": [],
                  "has_closing_tag": True,
                  "has_text": False
            },
            { "id": "" }
          ]
        },
        {
          "tag": "i",
          "id": "",
          "class": [],
          "children": [],
          "has_closing_tag": True,
          "has_text": False
        }
      ]
    },
    {
      "tag": "i",
      "id": "",
      "class": [],
      "children": [],
      "has_closing_tag": True,
      "has_text": False
    }
  ]
}
```

Primer enostavnega drevesa ene strani po prvem koraku

Da smo lahko zgradili tovrstno drevo, smo si pomagali z zunanjo knjižnico BeautifulSoup. Kot smo že omenili, smo na začetku vsebino strani s pomočjo te knjižnice pretvorili v gnezdeno podatkovno strukturo, nato pa smo si z metodami uporabljene knjižnice pomagali pri gradnji našega drevesa po opisani strukturi. S klicem določenih metod knjižnice BeautifulSoup, smo tako za vsak DOM element na strani lahko pridobili tekstovne vrednosti posameznih atributov za naše drevo (npr. *tag*, *id*, *class*), glede na obliko elementa nastavili vrednosti logičnim atributom (npr. *has_closing_tag* in *has_text*), ter ob prisotnosti morebitnih podrejenih elementov rekurzivno dopolnili seznam vseh podrejenih elementov ("children"). Implementacija funkcije, ki zgradi drevo po opisani strukturi s pomočjo metod knjižnice BeautifulSoup, je prikazana na spodnji sliki.

```
def build_dom_tree(self, soup):
    return None if soup.name is None else {"tag": soup.name,
                                           "id": soup.id if soup.id else "",
                                           "class": soup.attrs['class'] if "class" in soup.attrs else [],
                                           "children": list(filter(None, [self.build_dom_tree(child) for child in
                                                                           soup.children])),
                                           "has_closing_tag": not soup.is_empty_element,
                                           "has_text": soup.string is not None}
```

Implementacija metode za izgradnjo drevesa

3.3.2. Primerjava in združevanje dreves

Ko smo imeli zgrajeni drevesi po zgoraj opisanem postopku, smo morali ti dve drevesi medsebojno primerjati, ter poiskati morebitne razlike med obema. Zopet smo implementirali rekurzivno metodo, ki obe drevesi združi, ter elemente, ki se pojavijo zgolj v enem drevesu, označi kot opcijske (atributu "is_optional" priredi vrednost True). Metoda za osnovo vzame prvo drevo in nato tiste elemente, ki niso prisotni v drugem drevesu, označi kot opcijske. Hkrati se prepišejo tudi elementi iz drugega drevesa, ki ne obstajajo v prvem drevesu, tudi ti pa se označijo kot opcijski. Poleg označevanja elementov na trenutnem nivoju primerjave, se označijo tudi vsi podrejeni elementi. Spodnji dve sliki prikazujeta primer gradnje združenega drevesa z označevanjem opcijskih elementov.

```

def build_generalized_tree(self, tree1, tree2):
    if self.is_optional(tree1, tree2):
        tree1 = self.set_as_optional(tree1)
    else:
        if tree1["children"]:
            for index in range(min(len(tree1["children"]), len(tree2["children"]))):
                tree1["children"][index] = self.build_generalized_tree(tree1["children"][index],
                                                                        tree2["children"][index])

            if len(tree1["children"]) > len(tree2["children"]):
                for index in range(len(tree2["children"]), len(tree1["children"])):
                    tree1["children"][index] = self.set_as_optional(tree1["children"][index])
            else:
                for index in range(len(tree1["children"]), len(tree2["children"])):
                    tree1["children"].append(self.set_as_optional(tree2["children"][index]))

        elif tree2["children"]:
            tree1["children"] = [self.set_as_optional(child) for child in tree2["children"]]

        else:
            return tree1

    return tree1

```

Implementacija metode za združevanje in primerjavo dreves

```

def set_as_optional(self, element):
    element["is_optional"] = True
    element["children"] = [self.set_as_optional(child) for child in element["children"]]
    return element

def is_optional(self, element1, element2):
    return element1["tag"] != element2["tag"]

```

Struktura, ki jo vrne opisana metoda, je popolnoma enaka strukturi iz prvega koraka, le da se v tem koraku drevo poveča, elementom pa se doda še atribut, ki označuje element kot opcijski.

3.3.3. Generiranje ovojnice

Ko smo dobili združeno drevo v prej opisani obliki, je čas, da iz te oblike zgradimo ovojnico, ki na berljiv način opisuje splošno strukturo spletne strani in lahko služi kot pomoč pri implementaciji metod ekstrakcije podatkov iz katere koli instance dotične spletne strani, ki je zgrajena po opisani strukturi.

Metoda je implementirana tako, da se rekurzivno sprehodi po vseh elementih združenega drevesa. Elementi na trenutnem nivoju pregledovanja se zapišejo v obliki html značk, elementom, ki vsebujejo besedilo, pa se pripne še tekstovna oznaka (“#text”). Za vse elemente, ki vsebujejo podrejene elemente se nato preveri ali ti morebiti pripadajo seznamu. Preverjanje je realizirano tako, da se medsebojno primerjajo pari sosednjih elementov v seznamu - z uporabo rekurzije se primerjajo atributi, ki označujejo tip elementa (tag) na vseh nivojih trenutne veje drevesa. Če sta istoležna elementa enaka, hkrati pa so enaki tudi njima vsi podrejeni elementi, se element na trenutnem (najvišjem) nivoju označi kot seznam. Na koncu se za vsak element preveri še vrednost atributa *has_closing_tag* in glede na le-to, se

na konec elementa pogojno pripne še značka, ki zaključuje element. V primeru, da gre za opcijski element (*is_optional* = *True*), se element ovije z oklepaji, na konec pa se mu pripne vprašaj.

Spodnji odsek prikazuje primer ovojnice, ki jo vrne metoda ob pregledovanju dveh primerov enostavne strani. Za bolj jasn pregled smo rezultat formatirali po vrsticah, kot bi bilo to zapisano na pravi strani.

```
<html>
  <body>
    <b>#text</b>
    <img>
    <ul>(<li><i>#text</i></li>)+</ul>
    (<div>#text</div>)*
  </body>
</html>
```

Vsebina obravnavanih strani v nalogi, ki smo jo generirali z implementirano metodo *RoadRunner*, se nahaja v mapi *output*, vse ustrezne datoteke, pa imajo v imenu predpono *roadrunner*. Primer implementacije opisanega postopka prikazujejo spodnje tri slike.

```
def create_regex(self, generalized_tree):
    regex = ""

    is_optional = "is_optional" in generalized_tree and generalized_tree["is_optional"]

    if is_optional:
        regex += "("

    regex += "<" + generalized_tree["tag"] + ">"

    if generalized_tree["has_text"]:
        regex += "#text"

    prev_element = None

    for child in generalized_tree["children"]:
        is_list = self.check_if_list(prev_element, child)

        if not is_list:
            regex += self.create_regex(child)
        else:
            regex = self.mark_list_in_regex(regex, child["tag"])

        prev_element = child

    if generalized_tree["has_closing_tag"]:
        regex += "</" + generalized_tree["tag"] + ">"

    if is_optional:
        regex += ")?"
```

```
return regex
```

Implementacija metode, ki generira ovojnico

```
def check_if_list(self, prev_element, next_element):
    if prev_element is None or next_element is None:
        return False

    if prev_element["tag"] != next_element["tag"]:
        return False

    if not prev_element["children"] and not next_element["children"]:
        return True

    if prev_element["children"] and not next_element["children"] \
        or not prev_element["children"] and next_element["children"]:
        return False

    for index in range(len(prev_element["children"])):
        is_list = False if index >= len(next_element["children"]) else self.check_if_list(
            prev_element["children"][index], next_element["children"][index])
        if not is_list:
            return False

    return True
```

Implementacija metode, ki rekurzivno določi, ali je element seznam

```
def mark_list_in_regex(self, regex, tag):
    if regex[len(regex) - 1] == "+":
        return regex

    if regex[len(regex) - 1] == "?":
        return regex[:len(regex) - 1] + "*"

    i = regex.rfind("<" + tag + ">")

    return regex[:i] + "(" + regex[i:] + ")" + "
```

Implementacija metode, ki element označi kot seznam

Psevdokoda:

```
class RoadRunner:

    def start():
        tree1 = build_dom_tree(page1)
        tree2 = build_dom_tree(page2)

        generalized_tree = build_generalized_tree(tree1, tree2)

        regex = create_regex(generalized_tree)

        return regex

    def build_dom_tree(soup):
        return {
            tag: soup.name,
            has_closing_tag: !soup.is_empty_element,
            has_text: soup.string is not None,
            children: [build_dom_tree(child) for child in soup.children]
        }

    def build_generalized_tree(tree1, tree2):
```



```
    if tree1.tag != tree2.tag:
        set_as_optional(tree1)
    else:
        if tree1.children and tree2.children:
            tree1.children = build_generalized_tree(tree1.children,
tree2.children)

    return tree1

def set_as_optional(tree):
    tree.is_optional = True
    tree.children = [set_as_optional(child) for child in tree.children]

def create_regex(tree):
    regex = ""

    if tree.is_optional:
        regex += "("

    regex += "<" + tree.tag + ">"

    for child in tree.children:
        is_list = check_if_list(prev_element, child)

        if not is_list:
            regex += create_regex(child)
        else:
            add_list_tag_to_regex(regex, child.tag)

    if tree.has_closing_tag:
        regex += "</" + tree.tag + ">"

    if tree.is_optional:
        regex += ")??"

def check_if_list(prev_element, next_element):
    return prev_element.tag == next_element.tag and
check_if_list(prev_element.children, next_element.children)

def add_list_tag_to_regex(regex, tag)
    remove_optional_tag(regex)

    i = regex.rfind("<" + tag + ">")

    return regex[:i] + "(" + regex[i:] + ")+"
```

4. Zaključek

Sam seminar nam je bil zelo zanimiv, ker smo spoznali nove načine pridobivanja podatkov iz spletnih strani. Do sedaj smo poznali samo sprehajanje po DOM drevesu z uporabo npr. knjižnice Cheerio. Implementacija metod nam je vzela kar nekaj časa, a smo na koncu uspeli ustvariti ustrezne rešitve za vse tri.

5. Viri

1. W3Schools Xpath tutorial : https://www.w3schools.com/xml/xpath_intro.asp
2. Xpath hints: <https://devhints.io/xpath>
3. <https://regex101.com/>
4. 4. predavanja, WIER: Structured data extraction:
https://ucilnica.fri.uni-lj.si/pluginfile.php/98677/mod_label/intro/Structured%20data%20extraction_2.pdf
5. Parsing and Traversing DOM tree with BeautifulSoup:
<http://makble.com/parsing-and-traversing-dom-tree-with-beautifulsoup>