

# Poročilo projekta 2

XPath, regular expressions, Automatic Web extraction

Aljaž Rupar, Nal Lukšič, Gregor Ažbe

Opis spletne strani:

The screenshot shows a web page for car listings. The main heading is 'Zadnjih 100 oglasov: Osebna vozila'. Below this, there's a section 'Zadnjih 100:' with a description of the listings. To the right, there's a list of cars. The first car is a Peugeot 207 Trendy 1.4. Red boxes and arrows highlight specific fields for extraction: 'Name' (Peugeot 207 Trendy 1.4), 'Distance' (124 km), 'Year' (2008), 'Price' (2.500 €), 'Engine' (1.4), 'Power' (64 kW / 86 HP), and 'Gear' (ročni menjalnik (6 pr.)). The second car is an Audi A6 1.9 TDI, and the third is a Ford Galaxy 2.0 TDCI. The page also includes a sidebar with 'Zadnjih 100 po rubrikah:' and a list of categories like 'Rubrika AVTO:', 'Rubrika MOTO:', and 'ATV'.

Automatic Web extraction

Psevdo koda:

Read both pages with BeautifulSoup using lxml

Delete tags like script, style, comments, iframe, link

Delete attributes of all tags

procedure generate\_wrapper(html1, html2, wrapper):

while(iterate trough both html):

If(both Strings):

If(same String):

add string to wrapper

If(different string):

mark string miss match

If(both same element of type Tag):

add element to wrapper

if(both have children):

go recursive for all children

else:

if any has children, then add them to wrapper

# Here is a miss match

else:

find next match in the other

If(not found): # for both

```

        add element as optional
    else: # both found element
        add all skipped elements as optional
        continue on next match that matches sooner

    # one ends before other
    iterator -> check if iterator(repeating in same tag)
    If(iterator):
        add and mark as iterator
    else: # for both
        add as optional element
end procedure

print wrapper

```

Opis kode:

Pri implementaciji smo uporabili knjižnico BeautifulSoup. Potrebno se je bilo sprehoditi čez oba HTML dokumenta in izpisati ovojnico. Generiranje ovojnice je potekalo na način opisan v psevdo kodi. Pri implementaciji smo si pomagali s člankom, ki opisuje algoritem Road Runner.

Pravila, ki smo jih uporabili so sledeča:

Preverjali smo podobnost obeh HTML dokumentov s sprehajanjem po dokumentu v globino. Potrebno je bilo preverjati ali gre za html oznako ali za besedilo (NavigatableString ali Tag)

Vključili smo sledeče situacije:

- a) Iste oznake in besedila smo dodali v ovojnico.
- b) V primeru različnega besedila smo v ovojnici označili, da gre za besedilo na istem nivoju ter različnim besedilom. Oznaka za to je bila #Text.
- c) V primeru različne oznake smo najprej preverjali če pride do ujemanja v nadaljevanju drugega dokumenta. V primeru neujemanja smo dodali element kot opsijski in nadaljevali na naslednjem elementu. V primeru ujemanja smo nadaljevali na enem, ki se konča prej kot drugi ter dodali vse ostale elemente, ki smo jih izpustili kot opsijske.
- d) V primeru, ko se je eden izmed elementov končal prej kot drugi smo preverjali, če gre za iterator. To smo preverili z ujemanjem elementa v prejšnjih elementov v istem staršu. V primeru ujemanj smo elemente označili kot iteratorje in v nasprotnem primeru označili kot opsijske ter jih dodali v ovojnico.