resitev

January 28, 2024

1 Gutenberg

V domači nalogi delamo z enakimi podatki kot na predavanju in vajah, **vendar je projekt Gutenberg na dan ali dva po predavanjih spremenil obliko svojih strani**. To je odlično, saj boste morali svoje znanje pokazati na podobni, vendar malenkost drugačni nalogi.

Podatki projekta Gutenberg so dostopni tudi v lažje berljivi obliki. Tole delamo za vajo.

1.1 Ocena 6

Za oceno 6 bo potrebno napisati le dve funkciji, ki ne bi smeli imeti več kot 10 vrstic.

1.1.1 `prenesi_podatke()`

- Funkcija `prenesi_podatke()` naj stori naslednje.
  - Če v trenutnem direktoriju (v katerem teče program) ni poddirektorija `authors`, ga naredi (glej *koristne funkcije*, spodaj).
  - Vanj direktorij `authors` shrani datoteke s strani https://ucilnica.fri.uni-lj.si/mod/folder/view.php?id=54634. Rezultat prenašanja morajo torej biti datoteke authors/a.html, authors/b.html, authors/c.html … **Datotek, ki že obstajajo, ne prenašaj ponovno.**

  Predpostaviti smeš, da se datoteke nahajajo na naslovih `https://ucilnica.fri.uni-lj.si/pluginfile.php/217381/mod_folder/content/0/{crka}.html`, pri čemer so `crka` male črke angleške abecede.

  Koristne funkcije:

  - `os.path.exists`
  - `os.mkdir` ali `os.mkdirs`
  - ‘string.ascii_lowercase“
  - `urllib.request.urlopen`, ki, pazi, ne vrne niza temveč bajte. Kdor je pozabil, kako to spremeniti v niz, naj bere zapiske, saj smo morali to narediti tudi na predavanju.

  V vsem ostanku naloge uporabljaj te datoteke, da ne nadlegueš Učilnice brez potrebe.

Rešitev Komentirati ni kaj dosti, saj vse piše že v nasvetih o koristnih funkcijah: preverimo, ali direktorij obstaja in ga naredimo, če ga ni, nato pa gremo čez vse začetnice in poberemo, česar ni. Z `urlopen` “odpremo” stran, z `read()` preberemo vsebino in jo z `decode("utf-8")` pretvorimo iz bajtov v niz.

[1]:
```
import os
import string
from urllib.request import urlopen

def prenesi_podatke():
    if not os.path.exists("authors"):
        os.mkdir("authors")
    for crka in string.ascii_lowercase:
        if not os.path.exists(f"authors/{crka}.html"):
            open(f"authors/{crka}.html", "w").write(
                urlopen(f'https://ucilnica.fri.uni-lj.si/pluginfile.php/217381/
↪mod_folder/content/0/{crka}.html'
                        ).read().decode("utf-8"))
```

1.1.2 `avtorji(priimek)`

- Funkcija `avtorji(priimek)` vrne seznam vseh avtorjev s podanim priimkom.
  - Kaj so "avtorji", je očitno s spletne strani: poglej strukturo strani, da uvidiš, za kateri element gre.
  - Priimek je vse, kar se nahaja pred prvo vejico v vrstici z imenom avtorja.
  - V seznamu naj bodo celotne vrstice.

  Klic `avtorji("Fairfax")` vrne seznam `['Fairfax, Cecil', 'Fairfax, Edward, -1635']`. Seznam ne vsebuje "Fairfax Muckley, A. (Angelo)", saj je priimek tega avtorja Fairfax Muckley in ne samo Fairfax.

  Koristne funkcije:

  - Elementi, ki jih vrača BeautifulSoup imajo atribut `strings` z vsemi nizi znotraj elementa. Če imamo element `el`, ki vsebuje

  ```
  <div><a href="peter.html">Peter</<a> je pojedel veliko <img src="melona.png"> in šel v <im
  ```

  bo `el.strings` vseboval `"Peter"`, `" je pojedel veliko "`, `" in šel v "`, in `"."`. `strings` ni seznam, temveč generator. Če želimo seznam, pokličemo `list(el.strings)`, lahko pa ga podamo tudi kaki drugi funkciji ali metodi, na primer (namig!) metodi `join`, ki jo imajo nizi. (Nasvet: ko združuješ, ne vrivaj presledkov, razen v nalogi za oceno 10.)

1.1.3 Rešitev

Najprej preberemo html, v katerem bomo našli iskanega avtorja. Nekateri študenti so delali zanko prek vseh črk; to nima smisla, saj vemo, da je iskani avtor v `f"authors/{priimek[0]}.html"`.

Preberemo torej to datoteko. Z lepo župo gremo čez vse `h2` in `strings` poberemo vse, kar je napisano znotraj. Razdelimo glede na vejice in če je prvi element enak iskanemu priimku, dodamo celotno ime avtorja v seznam, ki ga na koncu vrnemo.

[2]:
```
def avtorji(priimek):
    html = open(f"authors/{priimek[0]}.html").read()
    soup = BeautifulSoup(html, "html.parser")
```

```
    avtorji = []
    for h2 in soup.find_all("h2"):
        avtor = "".join(h2.strings)
        if avtor and avtor.split(", ")[0] == priimek:
            avtorji.append(avtor)
    return avtorji
```

1.2 Ocena 7

1.2.1 `razberi_avtorja`

- Funkcija `razberi_avtorja(s)`, prejme vrstico z imenom avtorja ter letnicami rojstva in smrti.

  Če vrstica ne vsebuje ne letnice rojstva ne letnice smrti, funkcija vrne `None`.

  Sicer pa vrne četverko z naslednjimi elementi:

  - niz s priimkom avtorja (vse, kar je pred prvo vejico)
  - seznam z ostalimi elementi imena; dobimo ga tako, da vse, kar ni priimek ali letnice, razdelimo po vejicah
  - številka z letnico rojstva, če je podana; če ni naj bo ta element enak `None`.
  - številka z letnico smrti, če je podana; če ni naj bo ta element enak `None`.

  Letnici rojstva in smrti sta tisto, kar sledi zadnji vejici, pod pogojem, da imata *primerno obliko* in sta ločeni z `-` (brez presledkov).

  Letnica ima *primerno obliko*, če je

  - sestavljena iz števk,
  - ki jim morda sledi vprašaj (ki ga ignoriraj) ter,
  - morda, presledek, ki mu sledi BC.

  Dodatnih presledkov ali drugih znakov ni. Če se v datotekah pojavi kaj, kar je podobno letnici, vendar ne ustreza temu opisu, potem predpostavimo, da to ni letnica in jo ignoriramo. (**Pazi:** zadnji vejici mora slediti presledek in tako nato letnici. Če je vmes še kaj, na primer "*active*", potem tega avtorja ignoriraj.)

  Letnico pretvori v številko; če ji sledi BC, mora biti negativna.

  Če vrstica vsebuje ime brez vejic in nato letnico (na primer *Guérin de Bouscal, 1613?-1657*), se vedemo, kot da je celotno ime avtorja priimek, imena pa nima.

  - Klic `razberi_avtorja('Macaulay, James, Peter, Joe-Ann, 1817?-1902')` vrne `("Macaulay", ["James", "Peter", "Joe-Ann"], 1817, 1902)`.
  - Klic `razberi_avtorja("Cicero, Marcus Tullius, 106 BC-43 BC")` vrne `("Cicero", ["Marcus Tullius"], -106, -43)`
  - Klic `razberi_avtorja("Bach, P. D. Q., 33-12? BC")` vrne `("Bach", ["P. D. Q."], 33, -12)`. 33 nima BC, torej je pozitivna, 12 pa, torej je negativna. Ker P. D. Q. Bach.
  - Klic `razberi_avtorja("Štefan, Jožef")` vrne `None`, ker manjkata letnici.
  - Klic `razberi_avtorja("Kralj, Ludvik, 1914")` vrne `None`, ker manjkata letnici (1914 je videti kot leto, vendar ni znaka `-`)

– Klic `razberi_avtorja("Kralj, Ludvik, 1914-ič")` vrne `None`, ker manjkata letnici (`ič` niso števke)

Koristne funkcije:

– `re.match`; glej zapiske predavanja o regularnih izrazih. Tam je skoraj vse, kar potrebujemo, le za BC bo potrebno poskrbeti. Brez tega bo najbrž malo naporneje. Učenje regularnih izrazov je koristna naložba.

Rešitev Tule se začne najbolj zoprni del naloge - ne toliko zaradi zahtevnega dela temveč zato, ker je bilo potrebno (ne povsem predvideno) upoštevati toliko posebnosti.

[3]:
```
def razberi_avtorja(s):
    deli = s.split(", ")
    mo = re.match(r"(\d*)\??( BC)?-(\d*)\??( BC)?", deli[-1])
    if not mo:
        return None
    rojen, rbc, umrl, ubc = mo.groups()
    rojen = int(rojen) if rojen else None
    umrl = int(umrl) if umrl else None
    if rbc:
        rojen = -rojen
    if ubc:
        umrl = -umrl
    return deli[0], deli[1:-1], rojen, umrl
```

Regularni izraz smo na srečo napisali že na predavanju, le še `( BC)?` (ki pomeni da mora pride presledek in BC) dodamo.

Ker naloga pravi, da mora letnica slediti zadnji vejici in presledku, niz razbijemo glede na to ločilo in potem gledamo zadnji kos (`deli[-1]`). Če ta ne ustreza regularnemu izrazu, vrnemo `None`. Sicer shranimo njegove dele v spremenljivke - spet tako kot na predavanjih, le da imamo še dva dodatna `rbc` in `ubc`, ki sta morda prazna, morda pa sta enaka `" BC"`. Letnici pretvorimo v števili in če sta BC, ju še negiramo. Nato vrnemo, kar zahteva naloga: priimek (prvi element, `deli[0]`), ostala imena (vsi elementi razen prvega in zadnjega, `deli[1:-1]`) in letnici rojstva in smrti.

1.2.2 `zberi_podatke(crke)`

- Funkcija `zberi_podatke(crke)` prejme niz črk in vrne slovar, katerega ključi so priimki avtorjev, pripadajoče vrednosti pa seznam podatkov za vse avtorje s tem priimkom, pri čemer so "podatki" natančno četverke, ki jih vrača gornja funkcija. Avtorji so urejeni po vrstnem redu njihovega pojavljanja na strani.

  Funkcija uvrsti v slovar le tiste avtorje, katerih priimek se začne z eno od črk v podanem nizu `crke`. Če je ta niz prazen, pa prebere vse avtorje. Tako `zberi_podatke("mte")` sestavi slovar le za avtorje, katerih priimki se začnejo z m, t, ali e; klic `zberi_podatke("m")` zbere le podatke z avtorji s priimki na črko m; klic `zberi_podatke("")` zbere podatke za vse avtorje.

  Slovar, ki ga vrne `zberi_podatke("g")` se začne tako:

  ```
  {'Gaal': [('Gaal', ['György'], 1783, 1855),
  ```

```
          ('Gaal', ['József'], 1811, 1866),
          ('Gaal', ['Mózes'], 1863, 1936)],
 'Gabel': [('Gabel', ['Norman E.'], 1906, 1961)],
 'Gaboriau': [('Gaboriau', ['Emile'], 1832, 1873)],
 'Gabrielean': [('Gabrielean', ['M. Smbat'], 1856, 1919)],
 'Gabriel': [('Gabriel', ['Gilbert W. (Gilbert Wolf)'], 1890, 1952)],
 'Gabrielian': [('Gabrielian', ['Mugurdich Chojhauji'], 1857, None)],
 'Gade': [('Gade', ['John A. (John Allyne)'], 1875, 1955)],
 'Gadow': [('Gadow', ['Hans'], 1855, 1928)],
 'Gaebelein': [('Gaebelein', ['Arno Clemens'], 1861, 1945)],
 'Gaffarel': [('Gaffarel', ['Paul'], 1843, 1920)],
 'Gage': [('Gage', ['George W.'], 1887, 1957),
          ('Gage', ['Harry Lawrence'], 1887, 1982),
          ('Gage', ['Matilda Joslyn'], 1826, 1898),
          ('Gage', ['Thomas'], 1603, 1656),
          ('Gage', ['Thomas'], 1721, 1787)],
 'Gagneur': [('Gagneur', ['M.-L. (Marie-Louise)'], 1832, 1902)],

 ...
```

Če so poleg `"g"` še druge črke, vsebuje še priimke, ki se začnejo s temi črkami; če ga pokličemo s praznim nizem, pa vsebuje vse avtorje.

Rešitev Tale je precej rutinska: gremo čez podane črke ali, če je seznam črk prazen, čez vse črke. Preberemo ustrezni html, z župo poiščemo elemente `h2` in če vsebujejo avtorje (torej, če `razberi_avtorja` ne vrne `None`, v slovar dodamo, kar je pripravila `razberi_avtorja`.

[4]:
```
def zberi_podatke(crke):
    avtorji = defaultdict(list)
    for crka in crke or string.ascii_lowercase:
        html = open(f"authors/{crka}.html").read()
        soup = BeautifulSoup(html, "html.parser")
        for h2 in soup.find_all("h2"):
            avtor = "".join(h2.strings)
            podatki = razberi_avtorja(avtor)
            if podatki is not None:
                avtorji[podatki[0]].append(podatki)
    return avtorji
```

Izraz `crke or string.ascii_lowercase` bo imel vrednost `crke`, če je le-ta resnična (torej: če gre za neprazen niz), sicer pa bo imel vrednost `ascii_lowercase`. Brez tega trika bi bila funkcija pač vrstico ali dve daljša.

Uporabili smo `defaultdict(list)`, da se nam ni potrebno ukvarjati s tem, ali določen ključ (priimek) že obstaja. Če ga še ni, se bo pojavil "sam od sebe", pripadajoča vrednost pa bo slovar.

1.2.3 (nadaljevanje)

- V program dodaj (izven funkcije!) par vrstic, ki naredijo tole: če trenutni direktorij ne vsebuje datoteke "authors.json", pokliče funkcijo `zberi_podatke("")` in slovar, ki ga funkcija vrne kot rezultat, shrani v datoteko authors.json. Kot je očitno iz končnice, mora biti datoteka zapisana v obliki json.

  Testi ne bodo preverjali, da ne pišeš datoteke, če že obstaja. Tvoj problem, če bo program brez potrebe počasen.

  Vse naslednje funkcije. ki potrebujejo podatke o avtorjih, naj nalagajo to datoteko, namesto da brskajo po HTML-jih. Na ta način se bodo izvajale precej hitreje.

  Koristne funkcije:

  - modul `json`; glej zapiske.

Rešitev Dodati moramo torej

```
if not os.path.exists("authors.json"):
    open("authors.json", "w").write(json.dumps(zberi_podatke("")))
```

1.3 Ocena 8

Pazi: Ta in nadaljnje funkcije predpostavljajo, da bereš podatke iz jsona. Pri tem se tere spremenijo v sezname, zato bodo testi pričakovali, četverko s podatki o avtorjih kot sezname in ne terke. Pretvarjanje bi bilo možno, vendar nima smisla, da si zapletamo življenje.

- Funkcija `v_obdobju(rojen, umrl, zacetek, konec)` vrne `True`, če je oseba, ki je živela med leti `rojen` in `umrl`, vsaj del življenja živela znotraj obdobja med `zacetek` in `konec`.

  Pesnik, rojen v 1800 in umrl v 1848, je živel, na primer v obdobjih 1800-1900, 1750-1810, 1848-1552, ne pa v obdobju 1850-1900 ali 1750-1799.

  Ena izmed letnic, `rojen` ali `umrl`, je lahko tudi `None`. V tem primeru preveri le, ali se letnica, ki je znana, nahaja znotraj odbobja. (Tudi za osebo, ki je umrla 1551, ne bomo trdili, da je živela v 1500-1550, čeprav najbrž je, sicer je ne bi bilo na seznamu.)

  Predpostaviti smeš, da ne gre za P. D. Q. Bacha in je letnica rojstva manjša ali enaka letnici smrti. Letnica začetka obdobja je manjša ali enaka letnici konca.

  Da te odvrnemo od zank in drugih nepotrebnih komplikacij, na primer `range`, pa so letnice lahko tudi izjemno velike, na primer 10^22 in niso nujno cela števila.

  Koristne funkcije

  - Operator `<=` (pa še kakšen `is`, `not`, `and` in `or`) bo pametnemu povsem zadoščal. :)

Rešitev Tule si je potrebno malo risati ali pa vsaj mahati po zraku. En razmislek je tak: intervala A in B se sekata, če je začetek A znotraj B ali pa začetek B znotraj A. Se pravi, pisatelj je živel v določenem obdobju, če je bil rojen znotraj njega ali pa se je obdobje začelo, ko je bil pisatelj živ. Razmislite, da to dejansko pokrije situacije, ko je pisatelj rojen prej in živel v obdobje, ali pa rojen med in živel po njem, ali pa je bilo znotraj obdobja celo njegovo življenje ali pa je bilo celo obdobje

znotraj njegovega življenja. Če bi sestavili pogoj tako, da bi moral biti pisatelj znotraj obdobja bodisi rojen bodisi umrjen, potem France Prešeren (1800-1849) ne bi živel v letih 1820-1830.

Poleg tega se moramo poigrati še s primeri, ko ne poznamo bodisi letnice rojstva bodisi letnice smrti (vsaj ena pa je vedno znana).

Ena od možnih rešitev je torej

[5]:
```
def v_obdobju(rojen, umrl, zacetek, konec):
    if rojen is None:
        return zacetek <= umrl <= konec
    if umrl is None:
        return zacetek <= rojen <= konec
    return zacetek <= rojen <= konec or rojen <= zacetek <= umrl
```

1.3.1 `avtorji_v_obdobju(zacetek, konec)`

- Funkcija `avtorji_v_obdobju(zacetek, konec)` vrne seznam vseh avtorjev, ki so živeli v podanem obdobju. Vsak avtor je opisan s četverko, kot so shranjene v authors.json. Za avtorja se šteje, da je živel v podanem obdobju, če tako pravi prejšnja funkcija.

  Seznam naj bo urejen po priimkih, med tistimi z enakim priimkom pa po ostalih elementih imena.

  Klic `avtorji_v_obdobju(1310, 1311)` vrne

```
[['Amir Khusraw Dihlavi', [], 1253, 1325],
 ['Bury', ['Richard de'], 1287, 1345],
 ['Dante Alighieri', [], 1265, 1321],
 ['Gao', ['Ming'], 1306, 1359],
 ['Joinville', ['Jean', 'sire de'], 1224, 1317],
 ['Juan Manuel', ['Infante of Castile'], 1282, 1347],
 ['Liu', ['Ji'], 1311, 1375],
 ['Llull', ['Ramon'], 1232, 1316],
 ['Ma', ['Zhiyuan'], 1250, 1324],
 ['Petrarca', ['Francesco'], 1304, 1374],
 ['Polo', ['Marco'], 1254, 1324],
 ['Rolle', ['Richard', 'of Hampole'], 1290, 1349],
 ['Ruiz', ['Juan'], 1283, 1350],
 ['Ruusbroec', ['Jan van'], 1293, 1381],
 ['Shi', ["Nai'an"], 1290, 1365],
 ['Wang', ['Mian'], 1287, 1359],
 ['Wang', ['Shifu'], 1260, 1316],
 ['Zhu', ['Mingshi'], 1260, 1340]]
```

  Koristne funkcije:

  – Za urejenja bo dovolj poklicati `sorted` na končnem seznamu, brez posebnih ceremonij. Ta bo urejal po prvem elementu in potem po ostalih.

Rešitev Tale je precej rutinska: zanka prek vseh vrednosti v slovarju, ki ga naložimo iz `authors.json`. Te vrednosti so seznamu, torej naredimo še zanko prek seznama. Tako dobimo četvorke (priimek, imena, rojstvo, smrt). Če je pisatelj živel v podanem odbobju, ga dodamo. Na koncu vrnemo urejen seznam.

[6]:
```
def avtorji_v_obdobju(zacetek, konec):
    avtorji = json.loads(open("authors.json").read())
    v_obd = []
    for avtorjii in avtorji.values():
        for avtor in avtorjii:
            rojen, umrl = avtor[2:]
            if v_obdobju(rojen, umrl, zacetek, konec):
                v_obd.append(avtor)
    return sorted(v_obd)
```

Takole pa gre krajše.

[7]:
```
def avtorji_v_obdobju(zacetek, konec):
    avtorji = json.loads(open("authors.json").read())
    return sorted(avtor
                  for avtorjii in avtorji.values()
                  for avtor in avtorjii
                  if v_obdobju(*avtor[2:], zacetek, konec))
```

1.3.2 `razpon()`

- Funkcija `razpon()` vrne par števil z najzgodnejšo letnico rojstva in najkasnejšo letnico smrti v podatkih.

  Funkcija mora vrniti `(-1810, 2023)`. Vendar ne goljufaj in to dejansko izračunaj.

  (Kateri pisatelj je bil rojen leta -1810? Le poteši si radovednost. Ni napaka. Je zanimivo.)

Rešitev Ta funkcija je bila malo za oddih.

[8]:
```
def razpon():
    avtorji = json.loads(open("authors.json").read())
    najm = najv = 0
    for avtorjii in avtorji.values():
        for _, _, rojen, umrl in avtorjii:
            if rojen is not None and rojen < najm:
                najm = rojen
            if umrl is not None and umrl > najv:
                najv = umrl
    return najm, najv
```

Ali, krajše:

[9]:
```
def razpon():
    avtorji = json.loads(open("authors.json").read())
    return (min(rojen for avt in avtorji.values() for *_, rojen, _ in avt if␣
 ↪rojen is not None),
            max(umrl for avt in avtorji.values() for *_, _, umrl in avt if umrl␣
 ↪is not None))
```

1.3.3 `pokritost(zacetek, konec)`

- Funkcija `pokritost(zacetek, konec)` vrne seznam, ki vsebuje toliko števil, kolikor je let med `zacetek` in (vključno) `konec`. Element z indeksom `i` pove, koliko avtorjev v bazi je bilo živih v letu `zacetek + i`. Upoštevaj le pisatelje z znanima letnicama rojstva in smrti.

Rešitev Tudi tole ni znanost.

[10]:
```
def pokritost(zacetek, konec):
    avtorjev = [0] * (konec - zacetek + 1)
    avtorji = json.loads(open("authors.json").read())
    for avt in avtorji.values():
        for _, _, rojen, umrl in avt:
            if rojen is not None and umrl is not None:
                for x in range(max(rojen, zacetek), min(umrl, konec) + 1):
                    avtorjev[x - zacetek] += 1
    return avtorjev
```

V začetku si pripravimo seznam ničel, ki je dolg toliko, kolikor je let med `zacetek` in `konec` (vključno s koncem, zato `+ 1`). Potem gremo spet lepo prek avtorjev in če je za nekoga znano kdaj je rojen in kdaj je umrl, povečamo ustrezne letnice za `1`. Da ne pademo ven iz intervala, ki nas zanima, začnemo pri `max(rojen, zacetek)`: tako bomo, če je bil rojen pred začetkom intervala, začeli šteti šele na začetku intervala. Podobno je s koncem.

Nikjer nismo preverili, ali je avtor v resnici živel v tem obdobju. Ni treba. Če je bil rojen, recimo, po njem, bo `max(rojen, zacetek)` enak `rojen` in to bo več kot `min(umrl, konec)`. Če je "spodnja" meja višja od "zgornje", `range` pač ne naredi ničesar in zanka se ne izvede.

Pokritost se da izračunati tudi hitreje: namesto, da bi prešteval, tako kot zgoraj, si v neko začasno tabelo beležimo le, koliko avtorjev je bilo v nekem letu rojenih (`+ 1`) in koliko jih je umrlo (`- 1`). Z drugimi besedami, izvemo, za koliko se je v določenem letu spremenilo število avtorjev. Potem je potrebno to le sešteti; k temu prištejemo število avtorjev, ki so bili živi v začetku.

[11]:
```
def pokritost(zacetek, konec):
    spremembe = defaultdict(int)
    avtorji = json.loads(open("authors.json").read())
    for avt in avtorji.values():
        for *_, rojen, umrl in avt:
            if rojen is not None and umrl is not None:
                spremembe[rojen] += 1
                spremembe[umrl + 1] -= 1
    avtorjev = [sum(spremembe[x] for x in range(min(spremembe), zacetek + 1))]
```

```
    for x in range(zacetek + 1, konec + 1):
        avtorjev.append(avtorjev[-1] + spremembe[x])
    return avtorjev
```

1.4 Ocena 9

V tej nalogi se bomo ukvarjali z deli avtorjev. Upoštevajte le pisne knjige, ne avdio zapisov - v spodnjem primeru torej le prvo delo, ne drugega.

```
<li class="pgdbetext"><a href="/ebooks/16527">1001                    </a> (Russian) (as Author)<
<li class="pgdbaudio"><a href="/ebooks/19681">    </a> (Russian) (as Author)</li>
```

1.4.1 `razberi_delo(s)`

- Funkcija `razberi_delo(s)` dobi niz, ki morda vsebuje opis dela. Če gre za opis dela, vrne njegov naslov in jezik; sicer vrne `None`.

  Opis dela je oblikovan tako, da

  - naslovu dela
  - sledi presledek,
  - nato je v oklepajih naveden jezik; lahko vsebuje več besed (“Old English”) ali vezaje (“Napoletano-Calabrese”), (**opomba:** ignorirajte jezike, kateri opis vsebuje kak drug znak kot črke, presledke in vezaje; opravičujem se v času sestavljanja naloge spregledanim govorcem jezika `"Gaellic, Scottish"` - ta jezik ignorirajte, ker njegov opis vsebuje vejico)
  - sledi presledek,
  - nato pa v oklepajih niz `as Author`.

  Če ni oklepajev z jezikom in oklepajev z `as Author`, ne gre za opis dela.

  - `razberi_delo("Lightships and Lighthouses (English) (as Author)")` vrne `("Lightships and Lighthouses", "English")`.
  - `razberi_delo("Histoire de la Litterature Anglaise (Volume 1 de 5) (French) (as Author)")` vrne `'Histoire de la Litterature Anglaise (Volume 1 de 5)', 'French')`.
  - `razberi_delo("My Reminiscences (English) (as Translator)")` vrne `None`.

    Opisi nekaterih del obsegajo več kot eno vrstico (torej: vsebujejo znake `\n` ali `\r`). Ta dela ignoriraj. (Lepše bi bilo, če jih ne bi. Vendar bi vam bilo upoštevanje teh tako ali tako del najbrž samo otežilo nalogo, predvsem pa smo to posebnost spregledali v času pisanja testov, zato jih tudi testi ignorirajo.)

  Uporabne funkcije

  - `re.match`. Da bo preprosteje, izdam del izraza, ki bo ulovil jezik: `\(([\w\- ]+)\)`:
    * `\w` pomeni poljubno črko ali števko.
    * `[\w\- ]` pomeni poljubno črko ali števko ali `-` ali presledek. Vzvratna poševnica pred `-` je potrebna, ker ima `-` znotraj `[]` sicer poseben pomen; `a-f` bi pomenil vse znake med a in f.

* `[\w\- ]+` pomeni zaporedje z vsaj enim takšnim znakom.
* `([\w\- ]+)`: z oklepaji označimo ta del vzorca kot skupino, tako da jo bomo lahko potem izločili z `groups`.
* `\(([\w\- ]+)\)`: `\(` in `\)` pomenita oklepaj in zaklepaj; vzvratna poševnica je potrebna, ker imata `(` in `)` v regularnih sicer poseben pomen (glej gornjo vrstico).

V podanem dokumentu bi lahko delo in jezik sicer dobili tudi na drugačen, nekoliko preprostejši način. Vendar povadimo regularne izraze. Kdor ima težave, naj prebere zapiske. Če ne bo dovolj, pa pošlje mail.

Rešitev Tale funkcija je bila resno zoprna zaradi kupa izjem - na koncu koncev pa je bila čisto kratka.

[12]:
```
def razberi_delo(s):
    mo = re.match(r"(.*) \(([\w\- ]+)\) \(as Author\)$", s)
    if not mo or "\n" in s:
        return None
    return mo.groups()
```

1.4.2 `dela()`

- Funkcija `dela()` vrne seznam z vsemi deli vseh avtorjev v vseh HTML-jih, ki ste jih pripravili v nalogi za oceno 6. Delo se šteje kot delo, če ga gornja funkcija prepozna kot delo.

  Elementi seznama naj bodo šesterke z naslednjimi elementi

  – ime dela,
  – jezik,
  – priimek avtorja,
  – ostali deli imena avtorja,
  – leto rojstva,
  – leto smrti.

  Prva dva elementa sta torej takšna, kot ju vrne gornja funkcija, ostali štirje so takšni, s kakršnimi smo avtorje opisovali v prejšnjih.

  Vrstni red elementov v seznamu je poljuben.

  Ker se bo ta funkcija najbrž izvajala več kot 10 sekund, bodo testi prihranili tvoj čas tako, **jo bodo nehali testirati,** ko se v direktoriju pojavi datoteka works.json.

  Koristne funkcije:

  – Elementi, ki jih vrača `BeautifulSoup` imajo atribut `parent`, ki vrne očeta vozlišča.
  – Elementi imajo tudi metodo `find_previous_sibling(name)`, ki pregleduje prejšnje brate vozlišča in vrne prvega s podanim imenom. Če je `el` nek element, bo `el.find_previous_sibling("img")` med brati elementa `el` našel najbližji prejšnji element `img`.

Rešitev Z juho iščemo vse `li`, ki predsatvljajo dela (`pgdbetext`). Z `li.string` združimo vse, znotraj tega elementa in pokličemo `razberi_delo`. Če ta ni mnenja, da gre za delo, s `continue` nadaljujemo z naslednjim korakom zanke.

Sicer poiščemo prejšnji `h2` in iz njega razberemo avtorja. Če smo pri tem uspešni, dodamo delo.

[13]:
```
def dela():
    vsa_dela = []
    for crka in string.ascii_lowercase:
        html = open(f"authors/{crka}.html").read()
        soup = BeautifulSoup(html, "html.parser")
        for li in soup.find_all("li", class_="pgdbetext"):
            delo = razberi_delo("".join(li.strings))
            if delo is None:
                continue

            avtor = li.parent.find_previous_sibling("h2")
            avtor = razberi_avtorja("".join(avtor.strings))
            if avtor is None:
                continue
            vsa_dela.append(delo + avtor)
    return vsa_dela
```

1.4.3 (nadaljevanje)

- Podobno kot smo shranili avtorje, shranimo tudi dela. V program dodaj (izven funkcije!) dve, tri vrstice, ki naredijo tole: če trenutni direktorij ne vsebuje datoteke "works.json", pokliče funkcijo `dela()` ter seznam, ki ga funkcija vrne kot rezultat, shrani v datoteko works.json - datoteka mora biti očitno v obliki json.

Rešitev

```
if not os.path.exists("works.json"):
    open("works.json", "w").write(json.dumps(dela()))
```

1.4.4 `dela_po_jezikih()`

- Funkcija `dela_po_jezikih()` vrne slovar, katerega ključi so vsi jeziki, v katerih so napisana dela avtorjev, pripadajoče vrednosti pa število del v tem jeziku.

Rešitev Tole je bilo malo za počitek. :)

[14]:
```
def dela_po_jezikih():
    dela = json.loads(open("works.json").read())
    jeziki = defaultdict(int)
    for _, jezik, *_ in dela:
        jeziki[jezik] += 1
    return jeziki
```

Zanimiva je zanka `for`: `_` pobere prvi element (ki na ne zanima), in `*_` pobere vse elemente od tretjega naprej (ki nas prav tako ne zanimajo).

1.5 Ocena 10

Čestitam, prišli ste do nalog z oceno 10. Za vzpodbudo jih ni veliko in niso težke. Z malo spretnosti bo vsaka vsebovala le eno vrstico in potem (mogoče malo daljši, a ne zapleten) `return`.

1.5.1 `preveri_delo(delo, avtor=None, naslov=None, jezik=None, leto=None)`

- Funkcijo `preveri_delo` definiraj s takimi argumenti: `preveri_delo(delo, avtor=None, naslov=None, jezik=None, leto=None)`. Poklicati jo bo možno z, recimo `preveri_delo(delo, jezik="English", leto=1950)`.

  Funkcija prejme opis posameznega dela v obliki šesterke, kakršne smo sestavljali v funkciji `dela()` ter ime (ali del imena) avtorja, ime (ali del imena) dela, jezik dela ali leto. Katerikoli argument (razen prvega) je lahko tudi `None`; če je tako, ga ne upoštevamo. Funkcija vrne `True`, če velja vse naslednje.

  - Ime avtorja vsebuje podani niz `avtor`. Ime avtorja predtem obdelamo tako, da postavimo imena na začetek in jih združimo s presledki, priimek pa damo na konec. Tako iz `"Dostojevski", ["Fjodor", "Mihajlovič"]` nastane `"Fjodor Mihajlovič Dostojevski"`; funkcija vrne `True`, če je `avtor`, na primer `"Dostojevski"`, `"Mihaj"` ali tudi `"ovič Dos"`.
  - Ime dela vsebuje podani podniz `naslov`. `naslov "čin in kaz"`, se ujema z `"Zločin in kazen"`.
  - Delo je napisano v zahtevanem jeziku.
  - Avtor je bil živ v podanem letu `leto`.

  Tiste dele poizvedbe, ki so `None`, ignoriramo. Če je, recimo, podan le avtor in jezik, funkcija preveri le, ali gre za delo podanega avtorja v podanem jeziku, naslov dela in leto pa ignorira.

  Kadar je podan argument `leto`, vendar za avtorja ni znano leto rojstva ali smrti, funkcija vrne `False`. (Da si ne zapletamo življenja, vrnemo `False` tudi, če je, recimo, leto rojstva neznano, leto smrti pa sicer morda ravno enako iskanemu letu.)

Rešitev Nalogi za oceno 10 sta bili res preprosti. Njun namen je bil predvsem, da si pokažemo, kako koristne stvari smo zmožni narediti.

[15]:
```
def preveri_delo(delo, avtor=None, naslov=None, jezik=None, leto=None):
    pnaslov, pjezik, ppriimek, pimena, projen, pumrl = delo
    return ((avtor is None or avtor in " ".join(pimena) + " " + ppriimek)
            and (naslov is None or naslov in pnaslov)
            and (jezik is None or jezik == pjezik)
            and (leto is None or projen is not None and pumrl is not None and␣
 ↪projen <= leto <= pumrl))
```

1.5.2 `poisci(avtor=None, naslov=None, jezik=None, leto=None)`

- Napiši funkcijo `poisci(avtor=None, naslov=None, jezik=None, leto=None)`, ki vrne seznam vseh del, ki ustrezajo podanim kriterijem.

  Koristne funkcije:

– `preveri_delo` :)

Rešitev Tole bomo naredili z izpeljanim seznamom. Karkoli drugega bi bil greh, posebej za oceno 10.

[16]:
```
def poisci(avtor=None, naslov=None, jezik=None, leto=None):
    dela = json.loads(open("works.json").read())
    return [opis for opis in dela if preveri_delo(opis, avtor, naslov, jezik,␣
 ↪leto)]
```