resitev

January 28, 2024

1 Miklavževa pisma

Prišel je čas, da pomagamo Miklavžu.

Nalogi so priloženi podatki. Programiraj tako, da se bodo nahajali v podimeniku "podatki" imenika, v katerem je tvoj program. Če je tvoj program v imeniku /Users/benjamin/programiranje/naloga, morajo biti podatki v /Users/benjamin/programiranje/naloga/podatki. (Pisma, recimo, bodo v /Users/benjamin/programiranje/naloga/podatki/pisma).

Pri reševanju ne smeš predpostaviti, kako je ime otrokom, ponudnikom, darilom. Z drugimi besedami, v tvojem programu se ne sme pojaviti nobeno ime otroka, nobeno ime ponudnika, nobeno konkretno darilo. Program mora biti uporaben tudi prihodnje leto, ko bo Miklavž obdaroval druge otroke z drugimi darili drugih dobaviteljev.

1.1 Za oceno 6

V delu za oceno 6 bomo pripravili funkcije, s katerimi bomo kasneje lahko razbirali sopomenke (oziroma, predvsem različne oblike besed), ki pomenijo isto darilo.

1.1.1 `sopomenke_besede`

- `sopomenke_besede(s)` prejme niz s z različnimi oblikami neke besede. Oblike so ločene s presledki, "osnovna" beseda je označena z zvezdico. Funkcija mora vrniti par (terko), ki vsebuje "osnovno besedo" in množico vseh besed (vključno z osnovno). Klic `sopomenke_besede("sladkarije bombončki *bomboni liziko")` vrne `("bomboni", {"sladkarije", "bombončki", "bomboni", "liziko"})`.

Rešitev Gremo čez niz, ki ga razdelimo na besede. V množico shranjujemo besede, od katerih od-`strip`-amo morebitno začetno zvezdico. Ko naletimo na besedo z zvezdico, si jo zapomnimo še kot gl vno besedo. Na koncu vrnemo glavno besedo in množico.

[1]:
```
def sopomenke_besede(s):
    besede = set()
    for beseda in s.split():
        besede.add(beseda.strip("*"))
        if beseda[0] == "*":
            glavna = beseda[1:]
    return glavna, besede
```

[2]:
```
sopomenke_besede("sladkarije bombončki *bomboni liziko")
```

[2]:
```
('bomboni', {'bomboni', 'bombončki', 'liziko', 'sladkarije'})
```

1.1.2 `preberi_sopomenke`

- `preberi_sopomenke()` prebere "sopomenke" iz datoteke podatki/sopomenke.txt. Vrniti mora slovar, katerega ključi so vse besede, ki se pojavijo v datoteki, pripadajoče vrednosti pa "osnovne oblike" te besede. Če bi bila datoteka videti tako

  ```
  sladkarije bombončki *bomboni
  *zemljevid
  *vlak *vlakec
  ```

  bi funkcija vrnila

  ```
  {"sladkarije": "bomboni", "bombončki": "bomboni", "bomboni": "bomboni",
   "zemljevid": "zemljevid", "vlak": "vlak", "vlakec": "vlak"}
  ```

Rešitev Beremo datoteko, za vsako vrstico pokličemo prejšnjo funkcijo `sopomenke_besede` in v slovar kot ključe dodajamo besede iz množice, kot pripadajočo vrednost pa glavno besedo.

[3]:
```
def preberi_sopomenke():
    slovar = {}
    for vrstica in open("podatki/sopomenke.txt", encoding="utf-8"):
        glavna, druge = sopomenke_besede(vrstica.strip())
        for beseda in druge:
            slovar[beseda] = glavna
    return slovar
```

[4]:
```
sopomenke = preberi_sopomenke()

sopomenke["vlakec"]
```

[4]:
```
'vlak'
```

Notranji zanki se lahko izognemo z dvema metodama slovarjev, ki smo ju na predavanjih komajda omenili, če sploh.

Prva je `fromkeys`, ki sestavi slovar s podanimi ključi in določeno vrednostjo (če je ne podamo, pa bodo vrednosti `None`). `fromkeys` ni običajna metoda, saj ne pripada posamičnemu slovarju, kot recimo `get` ali `setdefault`, temveč razredu. (Uradno: `fromkeys` je `classmethod`.)

[5]:
```
imena = ["Ana", "Berta", "Cilka"]

dict.fromkeys(imena, 42)
```

[5]:
```
{'Ana': 42, 'Berta': 42, 'Cilka': 42}
```

Druga malo omenjena metoda je `update`. Če sta `d` in `e` neka slovarja, bo `d.update(e)` v slovar `d` prepisal vse, kar je v slovarju `e`.

S tema metodama lahko skrajšamo funkcijo v

[6]:
```
def preberi_sopomenke():
    slovar = {}
    for vrstica in open("podatki/sopomenke.txt", encoding="utf-8"):
        glavna, druge = sopomenke_besede(vrstica.strip())
        slovar.update(dict.fromkeys(druge, glavna))
    return slovar
```

- Ko napišeš drugo funkcijo, dodaj v program vrstico `sopomenke = preberi_sopomenke()` (postavi jo pod to funkcijo in na začetek vrstice). Slovar sopomenk se nikoli ne spreminja, torej ni nič narobe, če si poenostavimo delo tako, da ga uporabljamo kar kot *globalno spremenljivko*.
- Napiši funkcijo `prevod_besede(beseda)`, ki prejme besedo in vrne njeno "osnovno obliko". Če besede ni v slovarju sopomenk, pa naj funkcija vrne `None`. Klic `prevod_besede("vlakec")` vrne `"vlak"`. Klic `prevod_besede("vlak")` prav tako vrne `"vlak"`. Klic `prevod_besede("avtobus")` vrne `None`, ker te besede ni v slovarju.

  V funkciji lahko uporabiš spremenljivko `sopomenke` (da ne boš stalno klical branja sopomenk).

Rešitev Lahko bi napisali

[7]:
```
def prevod_besede(beseda):
    beseda = beseda.lower()
    if beseda in sopomenke:
        return sopomenke[beseda]
    else:
        return None
```

Lahko pa bi se spomnili, da natančno to - če ključ obstaja, vrne pripadajočo vrednost, sicer pa `None` - naredi metoda 'get.

[8]:
```
def prevod_besede(beseda):
    return sopomenke.get(beseda.lower())
```

1.2 Za oceno 7

V nalogah za oceno 7 bomo brali pisma Miklavžu. Nahajajo se v imeniku podatki/pisma in so videti, na primer, tako

```
Dragi Miklavž,

Pozdravljen! Kako kaj? Jaz sem v redu. Letos sem bila pridna (vsaj večino časa).
Prosim, prosim, prinesi mi kakšno knjigo, ki jo lahko prebiram pred spanjem. In
če lahko, dodaj še mehkega plišastega medvedka, da bova lahko skupaj brala.
Obljubljam, da bom vedno pospravljala svojo sobo.
```

```
Lepo se imej,
Ema
```

Opomba: večino pisem mi je prijazno sestavil chatGPT. :))))

1.2.1 `poenostavi_besedilo`

- `poenostavi_besedilo(s)` prejme niz. Vrniti mora niz, ki je enak podanemu, vendar vsebuje le črke, presledke in znake za novo vrstico. Poleg tega spremeni vse črke v male črke. Klic `poenostavi_besedilo("Dragi Miklavž,\n\npišem ti (kot vsako leto): lep božič!")` vrne `"dragi miklavž\n\npišem ti kot vsako leto lep božič"`.

Rešitev Nekateri so to reševali z neskončno verigo `replace`, s katerimi so zamenjevali vse, na kar so naleteli v kakem pismu. `s.replace(".", "").replace(",", "").replace("-", "").replace("(", "")`... in tako naprej. To ni preveč varna ideja. Bogve, kakšne znake bodo otroci še dopisali. Boljše je obdržati, kar želimo obdržati - pa čeprav bo takšna rešitev malo daljša.

[9]:
```
def poenostavi_besedilo(s):
    t = ""
    for c in s.lower():
        if c.isalpha() or c in " \n\t":
            t += c
    return t
```

[10]:
```
poenostavi_besedilo(
    "Dragi Miklavž,\n\npišem ti (kot vsako leto): lep božič!")
```

[10]:
```
'dragi miklavž\n\npišem ti kot vsako leto lep božič'
```

V resničnem svetu se to rešuje z regularnimi izrazi. Če boste pridni, vam jih bom nekoč razložil.

[11]:
```
import re

def poenostavi_besedilo(s):
    return re.sub("[^\w\s]", "", s.lower())
```

1.2.2 `izlusci_avtorja`

- `izlusci_avtorja(ime_dat)` prejme ime datoteke s pismom in vrne ime njenega avtorja. Ime je tisto, kar je zapisano v zadnji neprazni vrstici. Avtor gornjega pisma je očitno Ema.

Rešitev Beremo vrstice in si jih zapomnimo - če niso prazne. Na koncu vrnemo zadnjo zapomnjeno vrstico.

[12]:
```
def izlusci_avtorja(ime_dat):
    for vrstica in open(ime_dat):
        if vrstica.strip():
            avtor = vrstica.strip()
    return avtor
```

Edini trik tule je, da moramo pisati `if vrstica.strip()`. Nobena vrstica datoteke ni prazna, vsaka vsebuje vsaj `\n`. Zato je potreben `strip()`; vrstica nas zanima, če je neprazna tudi potem, ko odstranimo beli prostor (`\n` in morebitne presledke v sicer praznih vrsticah).

1.2.3 `izlusci_darila`

- `izlusci_darila(ime_dat)` vrne množico vseh daril, omenjenih v pismu v datoteki s podanim imenom. Beseda predstavlja darilo, če je enaka kateri od besed, ki se pojavijo med sopomenkami; v vrnjeni množici mora biti osnovna oblika besede. Za gornje pismo vrne `{"knjiga", "medvedek"}`

  Pazi: če se v besedilu pojavi beseda "navlaka" ali pa otrok piše, kako je videti "gozdna vlaka", to še ne pomeni, da bi rad vlak! Ne išči delov besed!

  Program bo seveda nenatančen: če otrok napiše "lani si mi prinesel piškote", jih bo pač dobil tudi letos; če napiše "na kolesu vedno nosim čelado", utegne dobiti kolo in čelado.

Rešitev Skoraj vse, kar smo programirali doslej, služi poenostavljanju te funkcije. Konkretno, vsako besedilo je potrebno poenostaviti, potem pa za vsako besedo preveriti, ali se (prek sopomenk) preslika v kako darilo. Kdor ni uporabljal doslej napisanih funkcij, je imel problem, ki si ga je nakopal čisto sam. Kdor jih je uporabljal, pa je tule napisal čisto preprosto funkcijo.

Odpremo datoteko, jo z `read` preberemo v celoti, nato pokličemo `poenostavi_besedilo`, rezultat razbijemo na besede in jih zlagamo v množico.

[13]:
```
def izlusci_darila(ime_dat):
    darila = set()
    besedilo = open(ime_dat, encoding="utf-8").read()
    besedilo = poenostavi_besedilo(besedilo)
    for beseda in besedilo.split():
        darilo = prevod_besede(beseda)
        if darilo:
            darila.add(darilo)
    return darila
```

Bolj neučakani pa sestavijo množico prevodov vseh besed, ki jih najdejo v poenostavljenem besedilu iz datoteke prebranem (besedni red tega stavka sledi kodi :), iz katere odstranijo `None`, ki ga v to množico nvlačejo besede, ki ne predstavljajo daril.

[14]:
```
def izlusci_darila(ime_dat):
    return {
        prevod_besede(beseda)
        for beseda in poenostavi_besedilo(
            open(ime_dat, encoding="utf-8").read()).split()
    } - {None}
```

1.2.4 `darila_po_otrocih`

- `darila_po_otrocih()` ne prejme nobenih argumentov. Vrniti mora slovar, katerega ključi so avtorji pisem, pripadajoče vrednosti pa darila, ki so v pismu omenjena. Vrnjeni slovar se

začne tako:

```
{'Albert': {'žoga'},
 'Ana': {'knjiga', 'pisalo', 'zvezek'},
 'Benjamin': {'čelada', 'kolo'},
 'Berta': {'čokolada', 'piškoti'},
 'Cilka': {'barvice', 'bomboni'},
 'Dani': {'kocke', 'bomboni'},
 'Daniela': {'knjiga', 'blazina', 'medvedek'},
 'Dudley': {'vlak', 'zvezek', 'pero', 'čokolada', 'bomboni', 'žoga', 'knjiga', 'kolo'},
 ...
```

Rešitev Gremo po datotekah v poddirektoriju "podatki/pisma". Iz vsake datoteke izluščimo avtorja in darila; prvo bo ključ, drugo pripadajoča vrednost.

[15]:
```
import os

def darila_po_otrocih():
    darila = {}
    for ime_dat in os.listdir("podatki/pisma"):
        ime_dat = "podatki/pisma/" + ime_dat
        darila[izlusci_avtorja(ime_dat)] = izlusci_darila(ime_dat)
    return darila
```

Glavna zafrkancija te funkcije je, da `os.listdir` vrača le imena datotek, ne pa tudi poti, ki smo jo podali `listdir`-u. Zato je potrebno to dodati. Ker bomo ime datoteke, skupaj s potjo, potrebovali na dveh mestih, kar spremenimo vrednost spremenljivke.

Veliko preprosteje je, če v okviru funkcije začasno zamenjamo trenutni direktorij.

[16]:
```
def darila_po_otrocih():
    darila = {}
    os.chdir("podatki/pisma")
    for ime_dat in os.listdir():
        darila[izlusci_avtorja(ime_dat)] = izlusci_darila(ime_dat)
    os.chdir("../..")
    return darila
```

Pripravljaje to nalogo pa sem odkril, da ima Pythonov modul `contextlib` funkcijo, ki začasno zamenja direktorij. (Kako sem to odkril? Kako človek kar tako, slučajno odkrije takšne stvari? Preprosto, zazdelo se mi je, da se to velikokrat potrebuje in da za to morda obstaja taka in taka bližnjica, vedel pa sem tudi, kje jo iskati.)

Funkcija `contextlib.chdir` vrne kontekst in uporabiti ga moramo z `with`. Kaj je to, vam najrž ne bom razlagal niti, če boste pridni. Če želite, uporabljajte kot magični urok za začasno menjavo direktorija.

[17]:
```
import contextlib
```

```
def darila_po_otrocih():
    darila = {}
    with contextlib.chdir("podatki/pisma"):
        for ime_dat in os.listdir():
            darila[izlusci_avtorja(ime_dat)] = izlusci_darila(ime_dat)
    return darila
```

To potem vodi v briljantno kratko rešitev.

[18]:
```
def darila_po_otrocih():
    with contextlib.chdir("podatki/pisma"):
        return {izlusci_avtorja(ime_dat): izlusci_darila(ime_dat)
                for ime_dat in os.listdir()}
```

Podobno jedrnati bi lahko bili tudi brez `contextlib.chdir`, le s prištevanjem poti bi se morali zafrkavati.

1.2.5 `zbirnik_daril`

- `zbirnik_daril()` ne prejme nobenih argumentov. Vrniti mora slovar, katerega ključi so darila, vrednosti pa število "naročenih" kosov tega darila. Tako ključu `"barvice"` pripada vrednost 7, saj si barvice želi 7 otrok.

Rešitev Pokličemo `darila_po_otrocih`. Ker nas imena otrok ne zanimajo, gremo le čez vrednosti in seštevamo.

Tu se nam splača uporabiti `collections.defaultdict(int)`. Na ta način nam ne bo treba preverjati, ali smo na določen predmet že naleteli in je ključ že v slovarju ter ga v nasprotnem primeru dodajati.

[20]:
```
from collections import defaultdict

def zbirnik_daril():
    zbirnik = defaultdict(int)
    for darila in darila_po_otrocih().values():
        for darilo in darila:
            zbirnik[darilo] += 1
    return zbirnik
```

Da se naučimo še nečesa novega iz zakladnice funkcij, ki se valjajo po Pythonovi standardni knjižnici: `Counter` iz modula `collections`. `Counter` je posebna vrsta slovarja. Ko ga konstruiramo, mu lahko podamo nek seznam ali kaj takega, pa bo preštel njegove elemente: ključi bodo elementi, vrednosti pa število pojavitev.

[1]:
```
from collections import Counter

s = ["Cilka", "Ana", "Berta", "Ana", "Ana", "Cilka"]
t = ["Ana", "Dani"]
```

```
c = Counter(s)
c
```

[1]:
```
Counter({'Ana': 3, 'Cilka': 2, 'Berta': 1})
```

K že preštetim lahko dodajamo nove.

[2]:
```
c.update(t)
c
```

[2]:
```
Counter({'Ana': 4, 'Cilka': 2, 'Berta': 1, 'Dani': 1})
```

Če ga pokličemo brez argumentov, seveda dobimo prazen slovar. In mu potem kasneje dodajamo elemente.

S `Counter` lahko rešimo nalogo tako:

[3]:
```
def zbirnik_daril():
    vsa_darila = Counter()
    for darila in darila_po_otrocih().values():
        vsa_darila.update(darila)
    return vsa_darila
```

Naloga pravi, da mora funkcija vrniti slovar. Ta vrne `Counter`. Vendar je `Counter` vrsta slovarja, torej je vse v redu. (Kaj pomeni biti "vrsta slovarja", pri tem predmetu ne bomo jasno definirali; za nas bodi dovolj, da se da z njim delati vse, kar se da delati s slovarjem - pa še par stvari zraven.)

1.3 Ocena 8

Miklavž nabavlja darila pri dolgoletnih dobaviteljih: Godler s.p., Klemenčič s.p., Lampič s.p., Pavlič s.p. in Dežman s.p. Njihovi ceniki so v podatki/dobavitelji. V kakšni obliki so, si oglej sam.

1.3.1 `preberi_cenik`

- `preberi_cenik(ime_ponudnika)` prejme ime ponudnika, na primer "Lampic" (brez šumnikov, da ni komedij na MS Windows). Vrniti mora slovar, katerega ključi so vse, kar ponuja podani s.p., pripadajoče vrednosti pa cene. Klic `preberi_cenik("Lampic")` vrne

  ```
  {'avtomobilček': 15.0,
   'barvice': 9.0,
   'bomboni': 3.0,
   'knjiga': 12.0,
   'medvedek': 15.0,
   ... in tako naprej
  ```

Rešitev Tule preprosto uporabimo `cvs.reader`. Vse, kar vrača, zlagamo v slovar, ki ga na koncu vrnemo.

[21]:
```
import csv
```

```
def preberi_cenik(ime_ponudnika):
    cenik = {}
    for vrstica in csv.reader(
            open(f"podatki/dobavitelji/{ime_ponudnika}.txt",
                 encoding="utf-8"),
            delimiter=":"):
        cenik[vrstica[0]] = float(vrstica[1])
    return cenik
```

Pri vseh funkciji je najbolj zanimivo, kako smo sestavili ime datoteke: ime ponudnika smo vstavili kar v f-niz.

Kdor počasneje tipka, pa raje napiše samo

[22]:
```
def preberi_cenik(ime_ponudnika):
    return {vrstica[0]: float(vrstica[1])
            for vrstica in csv.reader(
                open(f"podatki/dobavitelji/{ime_ponudnika}.txt",
                     encoding="utf-8"),
                delimiter=":")}
```

`csv.reader` je sicer praktičen vobče, pri tej nalogi pa od njega pravzaprav ni posebne koristi, saj rešitev brez njega ni prav nič bolj zapletena. Skoraj nasprotno.

[ ]:
```
def preberi_cenik(ime_ponudnika):
    cenik = {}
    for vrstica in open(f"podatki/dobavitelji/{ime_ponudnika}.txt"):
        predmet, cena = vrstica.split(": ")
        cenik[predmet] = float(cena)
    return cenik
```

1.3.2 `ceniki`

- `ceniki()` vrne slovar, katerega ključi so imena ponudnikov, vrednosti pa slovarji z njihovimi ceniki (torej tem, kar vrača prejšnja funkcija.) Za primer poglej kar v teste, funkcija `test_ceniki`.

Rešitev To je podobno kot `darila_po_otrocih`: gremo čez datoteke direktorija in zlagamo prebrano v slovar. Ime ponudnika pa dobimo tako, da imenu datoteke odbijemo zadnje štiri znake (.txt).

[23]:
```
def ceniki():
    vsi_ceniki = {}
    for ime_ponudnika in os.listdir("podatki/dobavitelji"):
        ime_ponudnika = ime_ponudnika[:-4]
        vsi_ceniki[ime_ponudnika] = preberi_cenik(ime_ponudnika)
    return vsi_ceniki
```

1.3.3 `najcenejsi_ponudnik`

- `najcenejsi_ponudnik(ceniki, darilo)` prejme cenike v enaki obliki, kot jih vrača prejšnja funkcija in ime darila. Vrniti mora par (terko) z imenom najcenejšega podudnika in ceno. (Pravni oddelek inštitucije svetega Miklavža ima še bolj preplašene pravnike kot UL in fakultete; ker je sv. Miklavž javna oseba, jih skrbi, da morda sodi pod javno upravo in mora vedno nabavljati pri najcenejšem ponudniku, ne glede na (ne)kvaliteto.)

  V primeru, da ima najcenejšo ponudbo več ponudnikov, mora vrniti tistega, ki je *kasneje* po abecedi.

  Klic `najcenejsi_ponudnik(ponudbe, "žoga")` vrne `("Dezman", 20)`: žoge so najcenejše pri Dežmanu in sicer stanejo 20.

  Klic `najcenejsi_ponudnik(ponudbe, "medvedek")` vrne `("Pavlic", 15)`. Medvedki po enaki ceni, 15, se dobijo tudi drugod, vendar je Pavlič zadnji po abecedi.

Rešitev Prav neumno je, da ima tako preprosta funkcija tako dolg opis. Gre za običajno funkcijo, ki išče največjo vrednost oziroma nekaj povezanega z njo (najnižjo ceno, oziroma ponudnika, ki jo ponuja).

[24]:
```
import math

def najcenejsi_ponudnik(ceniki, darilo):
    najcenejsi = None
    najcena = math.inf
    for ponudnik, cenik in ceniki.items():
        if cenik.get(darilo, math.inf) < najcena \
                or (cenik.get(darilo) == najcena
                    and ponudnik > najcenejsi):
            najcena = cenik[darilo]
            najcenejsi = ponudnik
    return najcenejsi, najcena
```

Glavni trik tu je pogoj. Kdaj je ponudnik boljši od vseh doslej? V osnovni, če je `cenik[darilo] < najcena`. - Vendar dotični ponudnik morda te reči sploh ne prodaja. Zato `cenik[darilo]` zamenjamo s `cenik.get(darilo, math.inf)`: če nečesa ni, se delamo, da je cena te reči neskončna. - Obenem poskrbimo tudi za začetno vrednost: `naj_cena` je v začetku `math.inf`, tako da bo že prva cena (pri nekom, ki reč dejansko prodaja) manjša. - Zadnji del pa razrešuje primere, ko novi ponudnik ponuja enako ceno kot najcenejši doslej, `cenik.get(darilo) == najcena`. Spet uporabimo `get`, saj ponudnik te reči morda nima, `math.inf` pa niti ni več potreben. Če bo `get` vrnil `None`, to pač ne bo enako najboljši ceni doslej. Skratka, če je cena enaka najnižji, se bomo za tega ponudnika odločili, če bo pa abecedi za najugodnjšim doslej.

Oklepaj v pogoju niti ni potreben, saj `and` veže močneje kot `or`. Napišemo ga zaradi preglednosti, pa tudi zato, da Python ve, da se pogoj nadaljuje v naslednji vrstici, ne da bi morali na konec vrstice pisati `\`, tako kot smo morali narediti vrstico višje.

1.3.4 `ponudniki_in_cene`

- `ponudniki_in_cene(ceniki, darila)` je podobna prejšnji funkciji, le da prejme množico daril. Vrniti mora slovar, katerega ključi so podana darila, vrednosti pa pari, kakršne vrača prejšnja funkcija.

  Klic `ponudniki_in_cene(ponudbe, {"medvedek", "kolo", "glina", "žoga"})` vrne

  ```
  {'glina': ('Godler', 3.0),
   'kolo': ('Pavlic', 210.0),
   'medvedek': ('Pavlic', 15.0),
   'žoga': ('Dezman', 20.0)}
  ```

  Ta funkcija nam torej pove, kje bo Miklavž dobil našteta darila in po koliko.

Rešitev Ta funkcija le pokliče prejšnjo za vsako darilo.

Bodisi tako

[25]:
```
def ponudniki_in_cene(ceniki, darila):
    pic = {}
    for darilo in darila:
        pic[darilo] = najcenejsi_ponudnik(ceniki, darilo)
    return pic
```

bodisi kar tako

[26]:
```
def ponudniki_in_cene(ceniki, darila):
    return {darilo: najcenejsi_ponudnik(ceniki, darilo)
            for darilo in darila}
```

1.4 Ocena 9

1.4.1 `vrednost_daril`

- `vrednost_daril(otrok)` vrne skupno vrednost daril, ki si jih želi otrok ob predpostavki, da bomo vsako od njih kupili pri najcenejšem ponudniku. Predpostaviti smeš, da je otrokovo pismo v datoteki podatki/pisma/{otrok}.txt.

Rešitev Iz pisma z `izlusci_darila` preberemo darila, ki jih želi otrok. Nato pokličemo `ponudniki_in_cene`, da izvemo ponudnike in cene teh daril. V dobljenem slovarju nas ne zanimajo ključi (darila) temveč le vrednosti, `.values()`. Vrednosti so pari `(ponudnik, cena)`; zanimajo nas le cene.

[27]:
```
def vrednost_daril(otrok):
    vrednost = 0
    darila = izlusci_darila(f"podatki/pisma/{otrok}.txt")
    for _, cena in ponudniki_in_cene(ceniki(), darila).values():
        vrednost += cena
    return vrednost
```

Spet je zabavno, kako smo sestavili ime datoteke.

Jedrnateje (a ne prav zares) je

[28]:
```
def vrednost_daril(otrok):
    return sum(
        cena for _, cena in ponudniki_in_cene(
            ceniki(),
            izlusci_darila(f"podatki/pisma/{otrok}.txt")
        ).values())
```

1.4.2 `preberi_tockovalnik`

- `preberi_tockovalnik()` prebere datoteko, v kateri Miklavž vodi evidenco o tem, kateri otrok je bil priden in poreden. Vsaka vrstica vsebuje ime otroka, ki mu sledi dvopičje. Sledijo pozitivne točke za dobra in negativna za slaba dela. Funkcija mora vrniti slovar, katerega ključi so imena otrok, ki se pojavijo v točkovalniku, pripadajoče vrednosti pa vsota njihovih točk.

Rešitev Ta je dokaj rutinska.

[29]:
```
def preberi_tockovalnik():
    tockovalnik = {}
    for vrstica in open("podatki/tockovalnik.txt", encoding="utf-8"):
        otrok, dela = vrstica.split(":")
        pridnost = 0
        for delo in dela.split():
            pridnost += int(delo)
        tockovalnik[otrok] = pridnost
    return tockovalnik
```

1.4.3 `dolocitev_daril`

- `dolocitev_daril(proracun, darila, ponudniki_cene)` prejme skupno ceno daril, ki jih lahko dobi nek otrok, množico daril, ki si jih želi, in slovar, kakršnega vrne funkcija `ponudniki_in_cene`. Funkcija vrne množico daril, ki jih bo ta otrok dejansko dobil. Določi jih na naslednji način.
  - Uredi darila od dražjih proti cenejšim, enako draga darila uredi padajoče po abecedi.
  - Za vsako darilo po vrsti pogleda, ali je njegova cena manjša od preostalega zneska, ki ga je na voljo. Če je, doda darilo med darila, ki jih bo prejel in ustrezno zmanjša znesek. Če je darilo dražje od preostale količina denarja, pa ga preskoči.
  - Nato nadaljuje z naslednjim darilom…

  Pomoč: če Pythonu damo urediti seznam terk, recimo parov, jih bo uredil po prvem elementu, tiste, pri katerih je prvi element enak, pa po drugem. Če hočemo nek seznam urediti padajoče, podamo funkciji za urejanje dodatni argument `reverse=True`

Rešitev To je pa ta zabavna naloga. Edina, v kateri ne gre samo za neko relativno mehansko prekladanj vrednosti po slovarjih in množicah.

[30]:
```
def dolocitev_daril(proracun, darila, ponudniki_cene):
    cene_in_darila = []
    for darilo in darila:
        cene_in_darila.append((ponudniki_cene[darilo][1], darilo))
    cene_in_darila.sort(reverse=True)

    darila = set()
    for cena, darilo in cene_in_darila:
        if cena <= proracun:
            darila.add(darilo)
            proracun -= cena
    return darila
```

V prvem delu sestavimo seznam parov cen in daril. V tem vrstnem redu. Nato ta seznam uredimo. Urejanje najprej primerja prve elemente (cene) in če imata dve stvari enak prvi element, ju primerja po drugem (ime darila). Če dodamo `reverse=True`, bo uredil po padajočih cenah in znotraj tega po padajočih imenih daril. (Naloga je napisana prijazno. Če bi zahtevala, da otrok dobi darilo, ki je prej po abecedi, bi bilo to bolj zopno.

V drugem delu gremo čez ta seznam. Vsako darilo, ki je v okviru proračuna, dodamo v košarico in ustrezno zmanjšamo proračun.

Lahko pa gremo tudi v drugo smer: darila uredimo naraščajoče in jih jemljemo s konca. Če za to uporabimo `pop`, ki vrne in *pobriše* zadnji element seznama, lahko zanko `for` zamenjamo z `while`, ki jo pustimo teči, dokler je seznam `cene_in_darila` neprazen - `while cene_in_darila`.

[4]:
```
def dolocitev_daril(proracun, darila, ponudniki_cene):
    cene_in_darila = []
    for darilo in darila:
        cene_in_darila.append((ponudniki_cene[darilo][1], darilo))
    cene_in_darila.sort()

    dobi = set()
    while cene_in_darila:
        cena, darilo = cene_in_darila.pop()
        if cena <= proracun:
            dobi.add(darilo)
            proracun -= cena
    return dobi
```

1.4.4 `darila_za_otroka`

- `darila_za_otroka(otrok, tockovalnik, otroci_darila, ponudniki_cene)` prejme ime otroka, točkovalnik (kot ga vrača funkcija `preberi_tockovalnik`), slovar, kot ga vrača `darila_po_otrociih` in slovar, kot ga vrača `ponudniki_in_cene`. Vrniti mora množico daril, ki jih bo dobil otrok glede na to, katera darila si želi in koliko denarja mu je Miklavž pripravljen nameniti.

  Skupna vrednost daril, ki jih bo dobil otrok, ne more preseči števila točk, deljenega z 10.

Katera darila bo dobil, je določeno v opisu prejšnje funkcije.

Rešitev Pri tej nalogi sem videl neke dolge izdelke, v resnici pa moramo le poklicati prejšnjo funkcijo z ustreznimi argumenti. Vse podatke, ki jih potrebujemo, dobimo že v argumentih.

[31]:
```
def darila_za_otroka(otrok, tockovalnik, otroci_darila, ponudniki_cene):
    return dolocitev_daril(
        tockovalnik[otrok] / 10,
        otroci_darila[otrok],
        ponudniki_cene)
```

1.5 Ocena 10

Napiši funkcijo `narocila()`, ki sestavi naročila za dobavitelje: funkcija mora sestaviti datoteke z imeni Dezman.txt, Godler.txt in tako naprej z vsebino v naslednji obliki:

```
Spoštovani dobavitelj,

pri vas bi rad naročil naslednja darila:

                     kosov
barvice.................6
blazina.................2
kocke...................2

Lep pozdrav,
Sveti Miklavž
```

Konkretna vsebina je za vsakega dobavitelja seveda drugačna. Datoteka mora biti oblikovana do pike in presledka tako, kot kaže vzorec. Naročeni predmeti naj bodo urejeni po abecedi.

Funkcija mora prej seveda ugotoviti, katera darila dobi kateri otrok in pri kom bodo nabavljena.

Rešitev Tole pa zdaj zahteva, da združimo vse, kar imamo: preberemo vsa pisma, ugotovimo cene želenih daril, preberemo točkovalnike, določimo, kaj bo dobil kateri otrok… To se zgodi v prvem bloku spodnje funkcije.

V drugem sestavimo slovar `narocila`: njegovi ključi bodo imena ponudnikov, vrednosti pa slovarji, katerega ključi bodo darila, pripadajoče vrednosti pa število kosov tega darila.

V tretjem gremo čez ta slovar. Vsak element pripada ponudniku, torej za vsak element odpremo datoteko in vanjo zapišemo seznam daril, ki jih naročamo pri tem ponudniku.

[5]:
```
def narocila():
    otroci_darila = darila_po_otrocih()
    vsa_darila = set(zbirnik_daril())
    ponudbe = ceniki()
    ponudniki_cene = ponudniki_in_cene(ponudbe, vsa_darila)
    tockovalnik = preberi_tockovalnik()
```

```
    narocila = {}
    for ponudnik, _ in ponudniki_cene.values():
        narocila[ponudnik] = defaultdict(int)
    for otrok in otroci_darila:
        for darilo in darila_za_otroka(otrok,
                                       tockovalnik,
                                       otroci_darila,
                                       ponudniki_cene):
            narocila[ponudniki_cene[darilo][0]][darilo] += 1

    for ponudnik, darila in narocila.items():
        f = open(ponudnik + ".txt", "wt")
        f.write("Spoštovani dobavitelj,\n\n"
                "pri vas bi rad naročil naslednja darila:\n\n")
        f.write(f"{'kosov':>26}\n")
        for darilo, kosov in sorted(darila.items()):
            f.write(f"{darilo:.<20}{kosov:.>6}\n")
        f.write("\nLep pozdrav,\nSveti Miklavž")
```