rešitev

January 28, 2024

0.1 Ponovitev: Preštevanje daril

Dudley je za rojstni dan dobil darila, katerih imena je zložil v seznam, recimo `["avto", "lok", "avto", "avto", "bomboni", "lok"]`.

- Napiši funkcijo `prestej(s)`, ki dobi takšen seznam in vrne slovar, katerega ključi so imena daril, pripadajoče vrednosti pa število kosov tega darila. Za gornji seznam bi funkcija vrnila `{"avto": 3, "lok": 2, "bomboni": 1}`.

0.1.1 Rešitev

Če mora funkcija vrniti slovar, mora najprej narediti slovar. Prazen slovar. Nato gremo čez darila. Če je že dobil kakšno takšno darilo, povečamo števec. Če ni, ga vstavimo v slovar in zabeležimo, da je dobil en izvod tega darila.

[1]:
```
def prestej(darila):
    stevila = {}
    for darilo in darila:
        if darilo in stevila:
            stevila[darilo] += 1
        else:
            stevila[darilo] = 1
    return stevila

prestej(["avto", "lok", "avto", "avto", "bomboni", "lok"])
```

[1]:
```
{'avto': 3, 'lok': 2, 'bomboni': 1}
```

Alternativa je, da darila, ki jih še ni dobil, dodamo v slovar z vrednostjo 0 in potem v vsakem primeru povečamo števec.

[2]:
```
def prestej(darila):
    stevila = {}
    for darilo in darila:
        if darilo not in stevila:
            stevila[darilo] = 0
        stevila[darilo] += 1
    return stevila

prestej(["avto", "lok", "avto", "avto", "bomboni", "lok"])
```

[2]:
```
{'avto': 3, 'lok': 2, 'bomboni': 1}
```

Na predavanjih sem najbrž povedal za `defaultdict` - ali pa morda tudi ne, saj ga skoraj noben študent ni uporabil. Torej bo zanj potrebno povedati ali pa ga vsaj ponoviti. Dobimo ga v modulu `collections`. `defaultdict` nas odreši nastavljanja manjkajočih elementov iz 0.

[3]:
```
from collections import defaultdict

def prestej(darila):
    stevila = defaultdict(int)
    for darilo in darila:
        stevila[darilo] += 1
    return stevila

prestej(["avto", "lok", "avto", "avto", "bomboni", "lok"])
```

[3]:
```
defaultdict(int, {'avto': 3, 'lok': 2, 'bomboni': 1})
```

Ta funkcija sicer ne vrne slovarja (`dict`) temveč slovar s privzetimi vrednostmi (`defaultdict`), kar ni čisto skladno z navodili naloge … Ali pač. `defaultdict` je v resnici samo posebna vrsta `dict`-a.

Pač pa je več študentov odkrilo `Counter`. To je lepo, res pa so se o tem naučili manj o programiranju in več o googlanju. Če uporabimo `Counter`, namreč ni česa programirati, saj je vse že narejeno.

[4]:
```
from collections import Counter

def prestej(darila):
    return Counter(darila)

prestej(["avto", "lok", "avto", "avto", "bomboni", "lok"])
```

[4]:
```
Counter({'avto': 3, 'lok': 2, 'bomboni': 1})
```

0.2 Obvezna naloga

V posameznem paketu je lahko več daril - a vedno iste vrste, da ne bi bilo zmede. Tako je dobil, recimo paket z dvema avtoma, paket z dvema lokoma, paket s štirimi avti, paket z enim avtom, paket z 42 bomboni in paket s tremi loki, ali, v Pythonu: `[("avto", 2), ("lok", 2), ("avto", 4), ("avto", 1), ("bomboni", 42), ("lok", 3)]`.

- Napiši funkcijo `sestej(darila)`, ki dobi seznam v tej obliki in vrne podoben slovar kot v ogrevalni nalogi, torej, `{"avto": 7, "lok": 5, "bomboni": 42}`.
- Poleg tega napiši funkcijo `stevilo_daril(darila)`, ki prejme slovar, kot ga vrača gornja funkcija (recimo `{"avto": 7, "lok": 5, "bomboni": 42}`) in vrne skupno število daril (v gornjem primeru 54, to je `7 + 5 + 42`).
- Napiši tudi funkcijo `razlicnih_daril(darila)`, ki dobi takšen slovar in vrne število različnih daril - v gornjem primeru 3.

- Napiši funkcijo `cena(darila, cene)`, ki prejme takšen slovar in slovar s cenami različnih daril, vrne pa skupno ceno prejetih daril. Če pokličemo, recimo,

```
cena({"avto": 7, "lok": 5, "bomboni": 42},
     {"avto": 3, "lok": 5, "bomboni": 1, "dron": 10})
```

(avto stane 3 evre, lok 5, bombon 1, dron 10), je skupna cena daril 88 (`7 * 3 + 5 * 5 + 42 * 1`). V slovarju s cenami so cene vseh daril, ki jih je prejel, lahko pa je v njem tudi cena česa, česar mu niso kupili.

0.2.1 Rešitev

Funkcija: `sestej` Podobna reč, le da ne gremo prek daril, temveč prek parov daril in količin. Namesto, da bi povečevali za `1` (oz. postavljali na `1`), povečujemo za (in postavljamo na) `kolicina`.

[5]:
```
def sestej(darila):
    stevila = {}
    for darilo, kolicina in darila:
        if darilo in stevila:
            stevila[darilo] += kolicina
        else:
            stevila[darilo] = kolicina
    return stevila

sestej([("avto", 2), ("lok", 2), ("avto", 4), ("avto", 1), ("bomboni", 42),␣
 ↪("lok", 3)])
```

[5]:
```
{'avto': 7, 'lok': 5, 'bomboni': 42}
```

Ali z, recimo, `defaultdict`-om:

[6]:
```
def sestej(darila):
    stevila = defaultdict(int)
    for darilo, kolicina in darila:
        stevila[darilo] += kolicina
    return stevila

sestej([("avto", 2), ("lok", 2), ("avto", 4), ("avto", 1), ("bomboni", 42),␣
 ↪("lok", 3)])
```

[6]:
```
defaultdict(int, {'avto': 7, 'lok': 5, 'bomboni': 42})
```

Kdor je v začetnem delu uporabil `Counter`, tokrat nima več sreče z bližnjicami. :)

Funkcija: `vseh_daril` Če vemo, da `stevila.values()` vrne vse vrednosti in da funkcija `sum` sešteje, kar ji damo, je preštevanje vseh daril trivialno.

[7]:
```
def vseh_daril(stevila):
    return sum(stevila.values())
```

```
vseh_daril({"avto": 7, "lok": 5, "bomboni": 42})
```

[7]: 54

Po pričakovanjih je precej študentov tu pozabilo na funkcijo `sum`, tako da so seštevali sami. Tudi prav. Vaja iz tipkanja ne bo škodila. Pa iz programiranja tudi ne.

Funkcija: `stevilo_daril` Število različnih daril je kar število ključev v slovarju. Vsak ključ je namreč unikaten.

[8]:
```
def stevilo_daril(darila):
    return len(darila)

stevilo_daril({"avto": 7, "lok": 5, "bomboni": 42})
```

[8]: 3

Zanimivo, koliko študentov je dolžino slovarja raje izračunalo z zanko.

[9]:
```
def stevilo_daril(darila):
    i = 0
    for darilo in darila:
        i += 1
    return i

stevilo_daril({"avto": 7, "lok": 5, "bomboni": 42})
```

[9]: 3

Vaja iz tipkanja je mogoče res koristna, v smislu programiranja pa tale vaja pravzaprav ni nek presežek.

Funkcija: `cena` Za računanje cene gremo prek parov daril in kosov. Za to uporabimo `stevila.items()`. Za vsako `darilo` pogledamo, koliko stane; to imamo, priročno, zapisano v `cene[darilo]`. Ceno pomnožimo s številom kosov in prištejemo k vsoti.

[10]:
```
cene = {"avto": 3, "lok": 5, "bomboni": 1, "dron": 10}

def cena(stevila, cene):
    vsota = 0
    for darilo, kosov in stevila.items():
        vsota += kosov * cene[darilo]
    return vsota

cena({"avto": 7, "lok": 5, "bomboni": 42}, cene)
```

[10]: 88

Po pričakovanjih vas je precej sprogramiralo funkcijo `cena` takole:

[11]:
```
def cena(darila, cene):
    cena=0
    for darilo, kosov in cene.items():
        for darilo1, cena in darila.items():
            if darilo == darilo1:
                cena += kosov * cena
    return cena

cena({"avto": 7, "lok": 5, "bomboni": 42}, cene)
```

[11]: 42

To je kar slabo. Ne samo, da je notranja zanka je počasna in nepotrebna – slovarje imamo namreč prav zato, da nam je ne bi bilo potrebno pisati. Slabo je predvsem to, da to pomeni, da še ne vemo čisto dobro, kaj slovarji so in kaj se da z njimi početi. Da jih pravzaprav še ne razumemo čisto. Vendar … bo že prišlo, sčasoma.

0.3 Dodatna naloga: Napredek

Dudley je, kot je znano, občutljiv na to, koliko daril prejme.

- Napiši funkcijo `napredek(lani, letos)`, ki prejme dva slovarja - prvi vsebuje lanska darila, drugi letošnja funkcija naj vrne slovar, katerega ključi so imena daril, katerih število je letos drugačno kot lani, pripadajoče vrednosti pa povedo, za koliko. Če pokličemo

```
napredek({"avto": 3, "lok": 5, "bomboni": 3, "dron": 1},
         {"avto": 8, "lok": 3, "bomboni": 3, "čokolada": 2})
```

Funkcija vrne `{"avto": 5, "lok": -2, "dron": -1, "čokolada": 2}`. Ne spreglej, da rezultat ne vsebuje bombonov, saj jih je prejel toliko kot lani.

0.3.1 Rešitev

Pregledati bo potrebno oba slovarja.

- Za vsako stvar, ki jo je dobil letos (`for darilo, kolicina in letos.items()`) preverimo, ali je lani ni (v tem primeru je napredek očiten: `razlika[darilo] = kolicia`; sicer pa preverimo, ali je letošnja količina tega darila različna od lanske in v tem primeru zabeležimo razliko.
- Za vsako stvar, ki jo je prejel lani, preverimo, ali je letos ni in v tem primeru zabeležimo negativen trend (`razlika[darilo] = -kolicina`).

[12]:
```
def napredek(lani, letos):
    razlika = {}
    for darilo, kolicina in letos.items():
        if darilo not in lani:
            razlika[darilo] = kolicina
        elif kolicina != lani[darilo]:
            razlika[darilo] = kolicina - lani[darilo]
```

```
    for darilo, kolicina in lani.items():
        if darilo not in letos:
            razlika[darilo] = -kolicina

    return razlika

napredek({"avto": 3, "lok": 5, "bomboni": 3, "dron": 1},
         {"avto": 8, "lok": 3, "bomboni": 3, "čokolada": 2})
```

[12]:
```
{'avto': 5, 'lok': -2, 'čokolada': 2, 'dron': -1}
```

Nekateri študenti radi pišejo `0 - kolicina`. Ni potrebe po odštevanju od 0. Pythonov `-` se vede tudi kot unarni operator. Predpostavljam, da bi si `x = -15` upali napisati? In ne bi pisali `x = 0 - 15`? :)

Program lahko malenkost skrajšamo, če uporabimo `get`: če darila ni v lanskem slovarju, to pomeni, da je lani dobil 0 kosov tega darila.

[13]:
```
def napredek(lani, letos):
    razlika = {}
    for darilo, kolicina in letos.items():
        delta = kolicina - lani.get(darilo, 0)
        if delta != 0:
            razlika[darilo] = delta

    for darilo, kolicina in lani.items():
        if darilo not in letos:
            razlika[darilo] = -kolicina

    return razlika

napredek({"avto": 3, "lok": 5, "bomboni": 3, "dron": 1},
         {"avto": 8, "lok": 3, "bomboni": 3, "čokolada": 2})
```

[13]:
```
{'avto': 5, 'lok': -2, 'čokolada': 2, 'dron': -1}
```

Prav dosti pa nismo profitirali.

Pač pa so nekateri razmišljali podobno, vendar so se `get`-u izognili tako, da so dopolnili slovar.

[14]:
```
def napredek(lani, letos):
    # Tole je zelo slaba ideja!!!
    for darilo in letos:
        if darilo not in lani:
            lani[darilo] = 0

    razlika = {}
    for darilo, kolicina in letos.items():
```

```
        delta = kolicina - lani.get(darilo, 0)
        if delta != 0:
            razlika[darilo] = delta

    for darilo, kolicina in lani.items():
        if darilo not in letos:
            razlika[darilo] = -kolicina

    return razlika

napredek({"avto": 3, "lok": 5, "bomboni": 3, "dron": 1},
         {"avto": 8, "lok": 3, "bomboni": 3, "čokolada": 2})
```

[14]:
```
{'avto': 5, 'lok': -2, 'čokolada': 2, 'dron': -1}
```

To se ne počne. Poglejte tole:

[15]:
```
lani = {}
letos = {"avto": 8, "lok": 3, "bomboni": 3, "čokolada": 2}
print("Napredek:", napredek(lani, letos))
print("Lani:", lani)
```

```
Napredek: {'avto': 8, 'lok': 3, 'bomboni': 3, 'čokolada': 2}
Lani: {'avto': 0, 'lok': 0, 'bomboni': 0, 'čokolada': 0}
```

Funkcija je zapackala slovar `lani`, vanj je svojevoljno dodajala nove elemente. To se ne dela!

Si predstavljate, da bi bila funkcija `sum`, ki vrne vsoto elementov seznama, napisana tako?

[16]:
```
def sum(s):
    v = 0
    while s != []:
        v += s.pop()
    return v
```

Saj dela ... samo seznam izprazni!

[17]:
```
s = [2, 4, 5]
print("Vsota, prvič:", sum(s))
print("Seznam po vsoti:", s)
print("Vsota, drugič:", sum(s))
```

```
Vsota, prvič: 11
Seznam po vsoti: []
Vsota, drugič: 0
```

Funkcije ne smejo nikoli, nikoli, nikoli spreminjati seznamov, slovarjev ... česarkoli, kar so dobile kot argument. Razen, seveda, če je točno to njihov namen. Za hrbtom pa - ne.

0.4 Rešitve, kot se jih bomo učili kmalu

```python
def prestej(s):
    return {x: s.count(x) for x in s}

def sestej(darila):
    return {darilo: sum(y for x, y in darila if x == darilo) for darilo, _ in darila}

def vseh_daril(darila):
    return sum(darila.values())

def razlicnih_daril(darila):
    return len(darila)

def cena(darila, cene):
    return sum(stevilo * cene[darilo] for darilo, stevilo in darila.items())

def napredek(lani, letos):
    return {x: letos.get(x, 0) - lani.get(x, 0) for x in set(letos) | set(lani) if letos.get(x
```