Računalništvo (fizika): 3. izpit

2. 7. 2018

Naloge rešujte na strežniku Tomo, `https://www.projekt-tomo.si`.

Brskanje po spletni učilnici, nalogah na strežniku Tomo, video zapiskih predavanj in *uradni* dokumentaciji za Python *je dovoljeno*. Brskanje po spletnih straneh, ki vsebujejo rešitve nalog in nasvete o tem, kako se rešuje naloge, *ni dovoljeno*. Komunikacija s komerkoli, razen z asistentom, *ni dovoljena*.

Čas reševanja je 180 minut. Doseženih 100 točk šteje za maksimalno oceno. Veliko uspeha!

1. naloga (25 točk)

Andrej je sestavil funkcijo, ki preveri, ali je dani niz palindrom, se pravi, da se bere enako naprej in nazaj:

```
def andrejev_palindrom(s):
    n = len(s)
    for i in range(0, (n+1)//2):
        if s[i] != s[n - i - 1]: return False
    return True
```

Sestavite funkcijo `palindrom`, ki deluje enako kot Andrejeva funkcija, vendar ne uporablja nobenih zank in nobenih pogojnih stavkov! Nasvet: namesto zanke uporabite rekurzijo, namesto pogojnih stavkov pa boolove operacije `and`, `or` itd.

Primeri uporabe:

```
>>> palindrom("pericarezeracirep")
True
>>> palindrom("ljubljana")
False
>>> palindrom("")
True
```

2. naloga (25 točk)

Plačilno-kreditna kartica je predstavljena z naslednjimi podatki:

1. naziv lastnika (neprazno zaporedje znakov dolžine največ 21)
2. vrsta kartice (debit ali credit)
3. izdajatelj (neprazno zaporedje znakov dolžine največ 16)
4. datum veljavnosti (mesec in leto)
5. številka kartice (16 števk, prva je lahko 0)

Na primer, kreditna kartica profesorja Bauerja je predstavljena s podatki:

```
Andrej Bauer
debit
Visa
05/21
4643 0400 0042 3451
```

a) (10 točk) V Pythonu sestavite razred `kartica`, s katerim predstavimo kreditno kartico. Implementirajte ustrezni konstruktor `__init__` in metodo `__repr__`. V komentar zapišite, kakšnega tipa mora biti vsak od atributov. Na primer, je številka kartice celo število, seznam števka, ali niz znakov?

b) (5 točk) Definirajte objekt `profesor` razreda `Kartica`, ki predstavlja kreditno kartico vašega profesorja. Definirajte še objekt `asistent` razreda `Kartica`, ki predstavlja *neveljavno* kartico, se pravi tako, ki *ne* zadošča zgoraj naštetim pogojem.

c) (10 točk) Razredu `Kartica` dodajte metodo `validiraj(self)`, ki preveri, ali je kartica veljavna. Metoda naj vrne `True`, če objekt `self` zadošča zgoraj naštetim pogojem, sicer `False`.

3. naloga (30 točk)

V trgovini so opazili, da nakup nekaterih izdelkov vodi tudi v nakup drugih izdelkov. Taki povezavi pravimo *asociacijsko pravilo*. Na primer, kdor kupi pivo, pogosto kupi tudi čips, kdor kupi šunko pa pogosto kupi tudi sir.

Trgovina nam je zadala nalogo, da s pomočjo podatkov o preteklih nakupih poiščemo vsa dovolj pogosta asociacijska pravila, torej vse pare izdelkov (x, y), da za dano verjetnost $p \in [0, 1]$ velja: kdor kupi x, ta z verjetnostjo vsaj p kupi tudi y.

Nakup ene stranke predstavimo kot množico izdelkov (posamezne količine vsakega izdelka ignoriramo), pretekle nakupe vseh strank pa kot seznam takih množic. Na primer, naslednji seznam predstavlja 11 nakupov:

```
primer = [{'pivo', 'cips', 'voda'},
          {'pivo', 'orescki'},
          {'cips', 'pivo'},
          {'pivo', 'orescki'},
          {'pivo', 'cips', 'orescki', 'kruh'},
          {'orescki', 'voda'},
          {'orescki', 'mleko'},
          {'pivo', 'cips', 'kruh'},
          {'voda', 'pivo', 'mleko'},
          {'pivo', 'cips'},
          {'voda', 'kruh', 'mleko'}]
```

Pri vrednosti $p = 0.8$ dobimo eno samo asociacijsko pravilo `('cips','pivo')`, ker je ob vsakem nakupu čipsa kupljeno tudi pivo. Pri vrednosti $p = 0.65$ dobimo asociacijska pravila

```
{('cips', 'pivo'), ('mleko', 'voda'), ('kruh', 'cips'), ('kruh', 'pivo')}
```

Na primer, od treh nakupov mleka dva vsebujeta tudi vodo. Ker je $2/3 > 0.65$, dobimo asociacijsko pravilo `('mleko','voda')`.

Sestavite funkcijo `asociacije(nakupi,p)`, ki vrne množico asociacijskih pravil za dani seznam nakupov in verjetnost `p`. Primeri uporabe:

```
>>> asociacije(primer, 0.8)
{('cips', 'pivo')}

>>> asociacije(primer, 0.6)
{('kruh', 'cips'), ('pivo', 'cips'), ('cips', 'pivo'), ('orescki', 'pivo'),
 ('kruh', 'pivo'), ('mleko', 'voda')}

>>> asociacije(primer, 0.65)
{('kruh', 'pivo'), ('cips', 'pivo'), ('kruh', 'cips'), ('mleko', 'voda')}
```

4. naloga (30 točk)

Aljaž in Sašo iz Makedonije v Slovenijo prevažata paradižnik. Pri veletrgovcu kupita zaboje paradižnika, ki jih nato zložita v kombi. Težava je v tem, da je lastnica kombija preprodajalka Mili, ki del kombija napolni s paprikami, preostanek prostora pa jima prepusti za paradižnike. Če za kakšne zaboje v kombiju zmanjka prostora, jih morata Aljaž in Sašo vrniti veletrgovcu po polovični ceni.

a) (15 točk) Aljaž nalaga zaboje v kombi tako, da jih razvrsti po teži, od najtežjega do najlažjega. Začenši z najtežjim zabojem, vsak zaboj naloži v kombi, če s tem še ne preseže dovoljene nosilnosti, sicer ga preskoči. Sestavite funkcijo `aljaz(zaboji,nosilnost)`, kjer je `zaboji` seznam mas zabojev in `nosilnost` največja dovoljena skupna masa, ki jo sme naložiti v kombi. Funkcija naj vrne par `(nalozeni,preostanek)`, kjer je `nalozeni` seznam mas naloženih zabojev in `preostanek` neizrabljena dovoljena masa.

Primer uporabe:

```
>>> aljaz([2, 32, 10, 12, 15], 40)
([32, 2], 6)
```

b) (15 točk) Sašo je perfekcionist, zato nalaga zaboje v kombi optimalno. Izmed vseh možnih podmnožic zabojev v kombi naloži tisto, ki ima največjo skupno maso, a še ne presega dovoljene skupne mase. Sestavite funkcijo `saso(zaboji,nosilnost)`, ki sprejme in vrne enake podatke kot funkcija `aljaz`, a deluje optimalno. Primer uporabe:

```
>>> saso([2, 32, 10, 12, 15], 40)
([15, 12, 10, 2], 1)
```