01 ponavljanje-2020

January 28, 2024

0.1 Ponavljanje: Naloge s prvega izpita (januar 2021)

0.1.1 5. naloga: Papajščina

Študent Brane si je za skrivno komuniciranje s kolegi izmislil način, s katerim kodira svoja sporočila. Vsako besedo v stavku je spremenil tako, da je za vsakim samoglasnikom dodal `p` *in ponovil tisti samoglasnik. Tako je npr. svoje ime* `'brane'` *preoblikoval v* `'brapanepe'`.

Napišite funkcijo `razkrij(stavek)`*, ki za podani zakodiran stavek (besede ločene s presledkom) vrne originalno sporočilo.*

```
>>> razkrij('upuvopod v propograpamipirapanjepe')
uvod v programiranje
>>> razkrij('kdapaj bopo koponepec vapaj')
kdaj bo konec vaj
```

Rešitev Niz bo potrebno prepisati v nov niz in preskakovati, kar je odveč.

Beremo torej podani stavek in počnemo tole: - ko vidimo soglasnik, ga le prepišemo - ko vidimo samoglasnik, ga prepišemo in preskočimo naslednji dve črki.

[7]:
```
def razkrij(stavek):
    novi_stavek = ""
    i = 0
    while i < len(stavek):
        if stavek[i] not in "aeiou":
            novi_stavek += stavek[i]
        else:
            novi_stavek += stavek[i]
            i += 2
        i += 1
    return novi_stavek
```

Deluje?

[12]:
```
razkrij('upuvopod v propograpamipirapanjepe')
```

[12]:
```
'uvod v programiranje'
```

[9]:
```
razkrij('kdapaj bopo koponepec vapaj')
```

[9]: `'kdaj bo konec vaj'`

Uporabili smo zanko `while`. Ko gremo prek nizov, seznamov in podobnih reči bomo, če se le da, uporabili `for`. Tu pa rešitev s `for` ni prav preprosta (a jo bomo vseeno poskusili, malo kasneje!), zato torej `while`. V osnovi imamo torej

```
i = 0
while i < len(stavek):
    ... naredi, kar moraš
    i += 1
```

Kako preverimo, ali je črka `stavek[i]` samoglasnik? Očitno bi lahko napisali

```
if stavek[i] == "a" or stavek[i] == "e" or stavek[i] == "i" or stavek[i] == "o" or stavek[i] ==
```

O tem dobi človek kar malo musklfibra. Če že morate, potem pišite:

```
crka = stavek[i]
if crka == "a" or crka == "e" or crka == "i" or crka "o" or crka "u":
```

Zgoraj, v funkciji, pa smo uporabili drug trik. Sestavili smo niz samoglasnikov, `aeiou` in se vprašali, ali črko `stavek[i]` najdemo v tem nizu. Če je ne (`not in`), je soglasnik (ali presledek ali kako ločilo).

Opazimo, da sta prvi vrstici znotraj `if` in `else` enaki. Ponavljanje vrstic ni preveč elegantno, sploh pa, če se je ponavljanja tako preprosto znebiti. Izvira iz tega, da smo že po slovensko nerodno povedali, kaj pravzaprav počnemo. Boljše bi bilo:

Beremo torej podani stavek in počnemo tole: - znak prepišemo v novi niz in nato - če je bil to soglasnik, nadaljujemo z naslednjim znakom - če je bil samoglasnik, pa preskočimo naslednja dva (torej gremo naprej za tri namesto za enega)

Tako dobimo, recimo, tole:

[15]:
```
def razkrij(stavek):
    novi_stavek = ""
    i = 0
    while i < len(stavek):
        novi_stavek += stavek[i]
        if stavek[i] not in "aeiou":
            i += 1
        else:
            i += 3
    return novi_stavek
```

Še vedno deluje?

[16]:
```
razkrij('upuvopod v propograpamipirapanjepe')
```

[16]: `'uvod v programiranje'`

[13]:
```
razkrij('kdapaj bopo koponepec vapaj')
```

[13]:
```
'kdaj bo konec vaj'
```

Še vedno deluje.

Malo krajše bi bilo tako:

[14]:
```
def razkrij(stavek):
    novi_stavek = ""
    i = 0
    while i < len(stavek):
        novi_stavek += stavek[i]
        if stavek[i] in "aeiou":
            i += 2
        i += 1
    return novi_stavek
```

Pri samoglasniku gremo naprej za dva znaka, v vsakem primeru pa še za enega.

Zdaj pa poskusimo z zanko `for`. Tako, za vajo.

```
def razkrij(stavek):
    novi_stavek = ""
    for crka in stavek:
        novi_stavek += crka
        if crka in "aeiou":
            ... kaj pa zdaj?!
```

Problem pri zanki `for` in takšnih nalogah je, da je zanka `for` trmasta. Šla bo prek vseh elementov. Enega za drugim. Ne moremo je prepričati ne, naj katerega preskoči, ne, naj se vrne nazaj, kjer je že bila. (Razloge za to bomo morda spoznali čez mesec ali dva.)

Tule se lahko rešimo le tako, da si zapomnimo, da je potrebno toliko in toliko znakov preskočiti. (Vsaj jaz se ne domislim druge rešitve.)

[18]:
```
def razkrij(stavek):
    novi_stavek = ""
    preskoci = 0
    for crka in stavek:
        if preskoci > 0:
            preskoci -= 1
        else:
            novi_stavek += crka
            if crka in "aeiou":
                preskoci = 2
    return novi_stavek
```

Najprej preverimo, da tole res deluje.

[19]:
```
razkrij('upuvopod v propograpamipirapanjepe')
```

[19]:
```
'uvod v programiranje'
```

[20]:
```
razkrij('kdapaj bopo koponepec vapaj')
```

[20]:
```
'kdaj bo konec vaj'
```

Kako deluje? Spremenljivka `preskoci` beleži, koliko naslednjih znakov je potrebno preskočiti.

Najprej poglejmo drugi `if`. Ta pravi: ko naletiš na samoglasnik (ki si ga prepisal v novi niz), si zapomni, da je potrebno naslednja dva znaka preskočiti, `preskoci = 2`.

Zdaj pa prvi `if`. Ta pravi: če je število znakov, ki jih je potrebno preskočiti, večje od 0, bomo tale znak preskočili - torej le zmanjšamo `preskoci` za `1`. Sicer pa prepišemo znak in preverimo, da ne gre morda za samoglasnik.

0.1.2 4. naloga: Karte

Karte lahko v programski kodi predstavimo tako, da jih zapišemo z dvema znakoma. Prvi označuje vrsto karte in je lahko katerikoli znak iz niza `'A23456789TJQK'`, drugi pa njeno barvo `'HDCS'`, pri čemer uporabljamo prve črke angleških poimenovanj.

Napišite `preveri(karte)`, ki za podan niz kart preveri, če so glede na prej opisano pravilo ustrezno zapisane. Če so vse pravilno zapisane vrne `True`, če pa je med njimi vsaj ena takšna, ki ni, pa `False`.

```
>>> preveri('AC AD AH AS KD')
True
>>> preveri('2C 4D D3 4H 2D 2H')
False
>>> preveri('1M AH 2H 3H H5')
False
```

Rešitev Najprej napišimo funkcijo, ki za podano karto preveri, ali je pravilna, se pravi, ali gre za niz z dvema črkama, pri čemer je prva črka ena izmed `'A23456789TJQK'`, drugi pa njeno barvo `'HDCS'`.

[21]:
```
def je_karta(karta):
    return len(karta) == 2 and karta[0] in 'A23456789TJQK' and karta[1] in␣
 ↪'HDCS'
```

Preverimo.

[22]:
```
je_karta("4S")
```

[22]:
```
True
```

[23]:
```
je_karta("4T")
```

[23]:
```
False
```

[24]:
```
je_karta("Benjamin")
```

[24]:
```
False
```

Preden gremo naprej, še enkrat poglejmo funkcijo. Vzemimo neko karto, npr

[25]:
```
karta = "4S"
```

Kaj je rezultat izraza `len(karta) == 2`? Operator `==` primerja to, kar je na levi (`len(karta)`) in na desni (`2`). Če sta stvari enaki, vrne `True`, sicer `False`.

[26]:
```
len(karta) == 2
```

[26]:
```
True
```

Podobno operator `in` pogleda, ali se levi operand nahaja znotraj desnega, se pravi, ali je `karta[0]` znotraj niza `'A23456789TJQK'` in `karta[1]` znotraj niza `HDCS`.

[27]:
```
karta[0] in 'A23456789TJQK'
```

[27]:
```
True
```

[28]:
```
karta[1] in 'HDCS'
```

[28]:
```
True
```

Vsi trije deli izraza torej vrnejo `True`. V gornjem `return`-u so združeni z `and`-i, torej imamo `True and True and True`, kar je seveda `True`.

[31]:
```
True and True and True
```

[31]:
```
True
```

Funkcija mora torej vrniti vrednost tega izraza.

Če se strinjamo z vsem, kar smo pravkar namodrovali - zakaj bi potem večina študentov sprogramirala tole?

[33]:
```
def je_karta(karta):
    if len(karta) == 2 and karta[0] in 'A23456789TJQK' and karta[1] in 'HDCS':
        return True
    else:
        return False
```

Ta izraz, `len(karta) == 2 and karta[0] in 'A23456789TJQK' and karta[1] in 'HDCS'` že ima natančno tisto vrednost, ki jo želimo vrniti, namreč `True` ali `False`, zato gornje nima smisla. Zadošča

[34]:
```
def je_karta(karta):
```

```
    return len(karta) == 2 and karta[0] in 'A23456789TJQK' and karta[1] in␣
↪'HDCS'
```

Zdaj, ko je to napisano in tudi na široko razloženo, pa končno napišimo še funkcijo, ki jo zahteva naloga. Podani niz razbijmo na karte in jih, eno za drugo, preverjajmo. Če naletimo na napačno, spričajmo, da je napačna. Sicer veselo nadaljujmo do konca in šele, ko smo preverili vse, vrnimo `True`.

[35]:
```
def preveri(karte):
    for karta in karte.split():
        if not je_karta(karta):
            return False
    return True
```

[36]:
```
preveri('AC AD AH AS KD')
```

[36]: True

[37]:
```
preveri('2C 4D D3 4H 2D 2H')
```

[37]: False

[38]:
```
preveri('1M AH 2H 3H H5')
```

[38]: False

Tipična napaka študentov začetnikov je

[39]:
```
def preveri(karte):
    for karta in karte.split():
        if not je_karta(karta):
            return False
        else:
            return True
```

To preveri le prvo karto. Za spodnji niz vrne `True`, dasiravno je tretja karta napačna.

[41]:
```
preveri('2C 4D D3 4H 2D 2H')
```

[41]: True

O dogajanju se hitro prepričamo z dodatnim izpisom.

[42]:
```
def preveri(karte):
    for karta in karte.split():
        print("Preverjam:", karta)
        if not je_karta(karta):
            return False
```

```
        else:
            return True
```

[43]:
```
preveri('2C 4D D3 4H 2D 2H')
```

```
Preverjam: 2C
```

[43]:
```
True
```

Očitno je preveril le `"2C"` in takoj, preveč navdušeno nad tem, da je pravilna, vrnil `True`.

Čez mesec ali dva, nekako takrat, ko bomo morda izvedeli, kako deluje zanka `for`, se bomo to naučili sprogramirati tudi hitreje:

[48]:
```
def preveri(karte):
    return all(je_karta(karta) for karta in karte.split())
```

[49]:
```
preveri('AC AD AH AS KD')
```

[49]:
```
True
```

[50]:
```
preveri('2C 4D D3 4H 2D 2H')
```

[50]:
```
False
```

[51]:
```
preveri('1M AH 2H 3H H5')
```

[51]:
```
False
```

0.1.3 3. Naloga: Strava

Študent Mirko se je odločil, da se bo športno aktiviral in bo začel teči. V seznamu si posamezne treninge beleži s terko, v kateri prva vrednost predstavlja število kilometrov, druga pa čas v minutah. Če nek dan ni šel tečt to označi s terko `(0,0)`.

Napišite funkcijo `statistika(s)`, ki za tak podani seznam vrne terko s tremi vrednostmi: število dni, ko je pretekel več ali enako 5 km; najdaljše zaporedje dni, ko ni tekel; seznam dni, ko je tekel enako ali hitreje od 6 min na kilometer (prva vrednost v seznamu se nanaša na dan s številko 1).

```
>>> statistika([(3,20), (4,25), (0,0), (5,30)])
1, 1, [4]
>>>  statistika([(3,15), (5,25), (0,0), (0,0), (0,0), (8, 40), (0, 0)])
2, 3, [1, 2, 6]
>>>  statistika([(10,55), (9,40), (12,60), (4,35), (5,40)])
4, 0, [1, 2, 3]
```

Rešitev Tu gre pravzaprav za tri naloge in nihče nam ne prepoveduje sprogramirati treh funkcij - in četrte, ki vrne rezultate vseh treh.

Najprej napišimo funkcijo, ki prešteje “dolge” teke.

[52]:
```
def dolgi(teki):
    dolgih = 0
    for dolzina, cas in teki:
        if dolzina >= 5:
            dolgih += 1
    return dolgih
```

[53]:
```
dolgi([(3,15), (5,25), (0,0), (0,0), (0,0), (8, 40)])
```

[53]: 2

Ta naloga je bila preprosta, pogledati pa se splača predvsem, kako smo napisali zanko `for`: ker so `teki` seznam parov `(dolzina, cas)`, smo pisali `for dolzina, cas in teki`.

Naprej je preprosto: če je dolžina večja ali enaka 5, povečamo števec `dolgih` tekov za 1.

Zdaj pa "hitri" teki. Tu bomo potrebovali dolžino in čas, poleg tega pa še indeks.

Spomnimo se, da `enumerate` "spremeni" seznam v seznam parov indeks, element. Če imamo nek seznam `s`, bi z `for x in s` dobili vse elemente, s `for i, x in enumerate(s)` pa pare indeksov in elementov. Tu pa je naš seznam že seznam parov, torej ga `enumerate` spremeni v seznam indeksov in parov. Če imamo brez `enumerate` zanko `for dolzina, cas in teki`, imamo z `enumerate` zanko `for i, (dolzina, cas) in enumerate(teki)`. Ne spreglejte oklepajev.

Ker želimo šteti od `1` in ne od `0`, dodamo funkciji `enumerate` še argument `start=1`.

[84]:
```
def hitri(teki):
    indeksi = []
    for i, (dolzina, cas) in enumerate(teki, start=1):
        if cas / dolzina >= 6:
            indeksi.append(i)
    return indeksi
```

Tole ne deluje.

Prva past, v katero smo se zataknili, je matematična.

[85]:
```
hitri([(3,15), (5,25), (0,0), (0,0), (0,0), (8, 40)])
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-85-58155ee12e2f> in <module>
----> 1 hitri([(3,15), (5,25), (0,0), (0,0), (0,0), (8, 40)])

<ipython-input-84-f8162784a4cd> in hitri(teki)
      2     indeksi = []
      3     for i, (dolzina, cas) in enumerate(teki, start=1):
----> 4         if cas / dolzina >= 6:
      5             indeksi.append(i)
      6     return indeksi
```

```
ZeroDivisionError: division by zero
```

Ker si nesrečnik zabeleži dneve, ko ni tekel, z `(0, 0)`, dobimo tu deljenje z 0. Takšne je potrebno preskočiti.

[86]:
```
def hitri(teki):
    indeksi = []
    for i, (dolzina, cas) in enumerate(teki, start=1):
        if dolzina > 0 and cas / dolzina >= 6:
            indeksi.append(i)
    return indeksi
```

Oba pogoja damo v isti `if`. Ne pišite

```
if dolzina > 0:
    if cas / dolzina >= 6:
        ...
```

Namesto dveh gnezdenih `if` lahko zelo pogosto (nikakor pa ne vedno) uporabimo `and`.

Druga past je fizikalna. Ali tekaška. Pri teku navadno ne merimo hitrosti, temveč tempo. Ne zanima nas, koliko kilometrov naredimo v uri, temveč, koliko minut potrebujemo za kilometer. Tekač nikoli ne teče z 10 kilometri na uro, temveč s šestimi minutami na kilometer. Čeprav je to eno in isto. Posledično je tekač hiter, ko so številke majhne, ne visoke. Hiter je bil recimo, norec, ki je pretekel maraton s hitrostjo manj kot treh minut na kilometer (se pravi, dve uri), ne nekdo, ki ga je odšepal s šestimi minutami na kilometer.

Pogoj v gornji funkciji je torej potrebno obrniti: ne `>= 6`, temveč `<= 6`.

[87]:
```
def hitri(teki):
    indeksi = []
    for i, (dolzina, cas) in enumerate(teki, start=1):
        if dolzina > 0 and cas / dolzina <= 6:
            indeksi.append(i)
    return indeksi
```

Zadnja funkcija je število zaporednih dni abstinence. Ta je nekoliko težja.

[88]:
```
def zaporedje(teki):
    naj_dolzina = 0
    dolzina = 0
    for razdalja, _ in teki:
        if razdalja == 0:
            dolzina += 1
            if dolzina > naj_dolzina:
                naj_dolzina = dolzina
        else:
            dolzina = 0
    return naj_dolzina
```

Zapomniti si je potrebno najdaljši čas abstinence (`naj_dolzina`) in dolžino trenutne abstinence (`dolzina`). V zanki se dogaja tole: če je razdalja `0`, se pravi, če ni tekel, povečamo trenutno dolžino abstinence za 1 in preverimo ali je to več kot največ, kar smo videli doslej. Če je tekel, pa je dolžina abstinence (spet) `0`.

Poglejte zanko `for`. Spet bi pisali `for razdalja, cas in teki`. Vendar spremenljivke `cas` v resnici ne potrebujemo. No, `for razdalja in teki` ne bi delovalo (vsaj ne tako, kot bi hoteli), saj bi bila `razdalja` v tem primeru cela terka. V zanki moramo torej našteti dve spremenljivke. Ker druge ne potrebujemo, pa ji damo ime `_`. To ime je uveljavljeno, običajno ime za spremenljivko, katere vrednost nas v resnici ne zanima, vendar jo moramo napisati, da se v njej "izgubi" neka nepotrebna vrednost.

Končno napišimo še funkcijo, ki jo hoče naloga: `statistika` bo poklicala vse tri funkcije in vrnila njihove rezultate.

[89]:
```
def statistika(teki):
    return dolgi(teki), zaporedje(teki), hitri(teki)
```

[90]:
```
statistika([(3,20), (4,25), (0,0), (5,30)])
```

[90]:
```
(1, 1, [4])
```

[91]:
```
statistika([(3,15), (5,25), (0,0), (0,0), (0,0), (8, 40)])
```

[91]:
```
(2, 3, [1, 2, 6])
```

[92]:
```
statistika([(10,55), (9,40), (12,60), (4,35), (5,40)])
```

[92]:
```
(4, 0, [1, 2, 3])
```

0.1.4 Naloga 2: Simon

Igra »Simon reče« gre tako, da karkoli reče tisti, ki je Simon, ostali naredijo. Npr. Simon reče: »Simon reče zaploskajte«, nato pa vsi zaploskajo. Če Simon na začetku ne reče »Simon reče«, pa tega ne smejo narediti.

Napišite funkcijo ***`simon(niz)`****, ki za podani niz preveri ali je ustrezen, se pravi, ali se začne s »Simon reče« ali ne. V primeru, da je, naj vrne del niza z navodilom, drugače pa* ***`False`****.*

```
>>> simon('Simon reče zaploskajte')
zaploskajte
>>> simon('Simon reče primite se za nos')
primite se za nos
>>> simon('Dvignite levo roko')
False
```

Rešitev Če se spomnimo, da imajo nizi funkcijo `startswith`, se s to nalogo ne bomo veliko zamudili.

[103]:
```
def simon(navodilo):
    zacetek = "Simon reče "
    if navodilo.startswith(zacetek):
        return navodilo[len(zacetek):]
    return False
```

[104]:
```
simon('Simon reče zaploskajte')
```

[104]:
```
'zaploskajte'
```

[105]:
```
simon('Simon reče primite se za nos')
```

[105]:
```
'primite se za nos'
```

[106]:
```
simon('Dvignite levo roko')
```

[106]:
```
False
```

Iz nekega zanimivega razloga deluje celo:

[107]:
```
def simon(navodilo):
    zacetek = "Simon reče "
    return navodilo.startswith(zacetek) and navodilo[len(zacetek):]
```

[108]:
```
simon('Simon reče zaploskajte')
```

[108]:
```
'zaploskajte'
```

[109]:
```
simon('Simon reče primite se za nos')
```

[109]:
```
'primite se za nos'
```

[110]:
```
simon('Dvignite levo roko')
```

[110]:
```
False
```

0.1.5 Naloga 1: Dvojiški zapis

Napišite funkcijo binarno(stevilo), ki za podano celo število vrne niz z binarno vrednostjo tega števila.

Spomnimo: binarno vrednost desetiškega števila dobimo tako, da ponavljamo postopek deljenja s številom dva dokler količnik ni enak 0. Ostanke preberemo v obratnem vrstnem redu.

Primer za število 25:

- *25 deljeno z 2 = 12, ostanek: 1*
- *12 deljeno z 2 = 6, ostanek: 0*
- *6 deljeno z 2 = 3, ostanek: 0*

- *3 deljeno z 2 = 1, ostanek: 1*
- *1 deljeno z 2 = 0, ostanek: 1*

Količnik je 0, zato končamo. Rezultat so ostanki, ki jih preberemo v obratnem vrstnem redu: 11001.

```
>>> binarno(10)
1010
>>> binarno(19)
10011
>>> binarno(42)
101010
```

Rešitev Le sledimo receptu: pripravimo niz, v katerega bomo zlagali ostanke. Dokler število ni enako 0, v niz z ostanki dodamo ostanek po deljenju z 2, število pa celoštevilsko (`//`, ne `/`) delimo z 2. Na koncu vrnemo niz z ostanki, obrnjen naokrog.

Če je začetno število enako 0, pa vrnemo "0". Z negativnimi pa se ne bomo ukvarjali.

[115]:
```
def binarno(i):
    if i == 0:
        return "0"

    s = ""
    while i > 0:
        s += str(i % 2)
        i //= 2
    return s[::-1] or "0"
```

[116]:
```
binarno(10)
```

[116]: '1010'

[117]:
```
binarno(19)
```

[117]: '10011'

[118]:
```
binarno(42)
```

[118]: '101010'