

Težave z nizi

Po tem, ko smo se poučili o tem, kako so shranjeni nizi in kaj vse se še vsebujejo poleg vidnih znakov, se vrnimo k vremenu v Radovljici, podatkom z dražbe in izboru koles.

```
for vrstica in open("temperature.txt"):
    print(vrstica)
```

24

18

15

16

18

Vidimo prazne vrstice? Ki jih ne bi bilo, če bi izpisovali številke, ne nizov?

```
for vrstica in open("temperature.txt"):
    temp_c = int(vrstica)
    print(temp_c)
```

24

18

15

16

18

To se zgodi, ker vsaka vrstica, prebrana iz datoteke, vsebuje tudi znak za konec vrstice. Tako je, recimo, videti zadnja.

```
vrstica
'18\n'
```

To nas potem pesti pri nalogi z vaj, kjer je potrebno prešteti, kolikokrat se je nek lastnik štirih koles peljal s katerim izmed njih, recimo Cubom.

```
cube = 0
for vrstica in open("../vaje/kolesa.txt"):
    if vrstica == "Cube":
        cube += 1
```

cube

0

Že če pogledamo zadnjo vrstico, vidimo, da se je vozil tudi s Cubom, vendar ima vrstica na koncu, seveda, `\n`, zato primerjava `vrstica == "Cube"` vrne `False`.

```
vrstica
'Cube\n'
vrstica == "Cube\n"
True
Seveda lahko pišemo
cube = 0
for vrstica in open("../vaje/kolesa.txt"):
    if vrstica == "Cube\n":
        cube += 1

cube
46
```

in dobimo pravi rezultat, vendar nam gre (no, vsaj meni) to na živce. Menda ne obstaja kak način, da se znebimo tega trapastega `\n`?

Še hujše: kaj, če je pri zapisovanju nekonsistenten in včasih piše "Cube", včasih "cube" in včasih "CUBE"? Obstaja kakšna oblika primerjanja, ki ignorira razliko med velikimi in malimi črkami?

Aja, pa, zdi se mi, da še nisem omenil: njegova tipkovnica je nekaj pokvarjena in včasih dela dvojne n-je, tako da se v njegovi datoteki včasih pojavi tudi Stevenns, NNakamura in Cannyon (in Canyonn). Je mogoče na kak preprost način zamenjati vse dvojne n-je v nizu z enojnimi?

Kam gremo? V funkcije za delo z nizi. Pravzaprav v nekaj splošnejšega.

Metode nizov

Doslej smo spoznali le prgišče Pythonovih funkcij - le toliko, da smo se jih naučili klicati in da smo lahko kaj drobnega sprogramirali.

- Vemo za `print` in `input`. Tidve sta splošno uporabni funkciji, predvsem prva. (Druga bo hitro potonila iz spomina, ker v resnici ni tako zelo uporabna.) Podobno `open`.
- Vemo za nekaj funkcij za delo s števili, kot so `sqrt`, `sin` in `log`. Te so v resnici pospravljene v škatlici z imenom `math` in da smo jih lahko uporabili, smo jih morali pričarati s `from math import *`.
- Vemo za `int` in `float`. To v resnici nista funkciji, temveč tipa. Vendar so vsi tipi v Pythonu tudi na nek način funkcije. Kakorkoli, `int` in `float` ste očitno zelo splošni zadevi.

Python 3.12 ima v resnici samo 71 funkcij (med katerimi je kakšnih 20 tipov). Še več, izmed teh 71 funkcij sem tako, na hitro, naštel 20 funkcij, ki jih dejansko kolikor toliko redno uporabljam. Ostale so bolj kot ne eksotika. Ostale, uporabne funkcije - in Python jih ima na tisoče - so pospravljene po škatlicah.

Kakšnih škatlicah? Takole. Vzemimo nek niz, recimo

```
vrstica = "NNakamura\n"
```

Tako kot številke smo tudi nize na tablo narisali kot škatlice, ki vsebujejo nek podatek. Poleg tega vsebujejo te škatlice tudi funkcije za delo s tem podatkom. Kot smo omenili v vaji "Kolesa" se lahko odvečnega `\n` na koncu niza znebimo tako, da napišemo `vrstica = vrstica.strip()`. Ta `strip` je funkcija, ki se nahaja "znotraj" niza `vrstica`. In do tistega, kar je "znotraj", pridemo tako, da napišemo `vrstica`, nato piko (ki v vseh kontekstih v Pythonu - še nekaj jih bo - pomeni "znotraj") in potem ime tistega, kar je "znotraj", torej `strip`. Ker je to funkcija, jo pokličemo. In ker zanjo (zdajle) nimamo posebnih argumentov, pustimo oklepaja prazna.

```
vrstica
```

```
'NNakamura\n'
```

```
vrstica.strip()
```

```
'NNakamura'
```

Takšnim funkcijam znotraj škatlic rečemo metode, po angleško *method*. (Ime je, priznajmo, malo čudno. Zgodovinsko. Vseeno boljše od tega, kako so jih klicali v začetku: sporočilo, *message*. V kontekstu takratnega razmišljanja, opisovanja tega, kar se dogaja ob klicanju metod, je bilo to ime smiselno, danes pa niti med programerji ni veliko takšnih, ki bi vedeli za ta izraz.)

`strip`

`strip` je torej metoda nizov. Vsi nizi imajo `strip` (vsak svojega, drugega? kakor se vzame.) in njeno delo je, da vrne niz brez belega prostora na začetku in na koncu. Beli prostor so presledki, znak za novo vrstico in tabulatorji.

```
s = "    niz s    preveč presledki. \n Res.    \n\n "
```

```
s.strip()
```

```
'niz s    preveč presledki. \n Res.'
```

Vidimo: odstranil je vse presledke in nove vrstice na začetku in na koncu, tiste vmes pa je pustil pri miru.

Obstajata različici `lstrip` in `rstrip`, ki odstranita le presledke na začetku in na koncu. Kateri je kateri, si boste že zapomnili, saj znate angleško.

Metodi `strip` lahko podamo tudi argument, če želimo odstranjevati kaj drugega kot presledke. To bomo počeli redko, a vseeno pokažimo:

```
"....čemu služijo te pike?!...".strip(".")  
'čemu služijo te pike?!'
```

Mimogrede smo videli, da lahko metode kličemo tudi na *literalih* za nize ("dobesedno" navedenem nizu) ne le na nizih, ki so že poimenovani, kot je bil gornji `s`.

Tako se bomo torej znebili `\n` na koncu vrstice s kolesom.

```
vrstica.strip()  
'NNakamura'
```

Nobena metoda niza ne spreminja niza, temveč kvečjemu vrne nov niz. Nizi so nespremenljivi. Niz, na katerega se nanaša `vrstica`, je še vedno tak, kot je bil.

```
vrstica  
'NNakamura\n'
```

Metoda `strip` je zgolj vrnila nov niz in če hočemo, da niz, na katerega se nanaša `vrstica`, ne bo več vseboval `\n`, moramo imenu `vrstica` prirediti vrednost, ki jo vrne `vrstica.strip()`.

```
vrstica = vrstica.strip()  
  
vrstica  
'NNakamura'
```

Velike in male črke

Naš naslednji kamen spotike je bilo primerjanje, pri katerem zanemarjamo razliko med malimi in velikimi črkami. Takšno torej, po katerem so nizi `"canyon"`, `"Canyon"` in `"CANYON"` enaki.

Priznati moram, da funkcije, namenjene takšnemu primerjanju, nisem še nikoli uporabil, sem pa slutil, kje jo najti in jo tam ob zapisovanju teh zapiskov tudi hitro našel. Za tiste, ki bi jo utegnili potrebovati: imenuje se `locale.strcoll`. Kako, da tak postaran programer, kot sem jaz, ne ve zanjo? Preprosto: ta problem rešim po preprostejši bližnjici. `locale.strcoll` bi moral uporabiti le za urejanje nizov po abecedi, za ugotavljanje enakosti pa oba niza preprosto pretvorim v male (ali velike) črke.

```
vrstica.lower()  
'nnakamura'  
  
vrstica.upper()
```

```
'NNAKAMURA'
```

Ali celo

```
vrstica.capitalize()
```

```
'Nnakamura'
```

Spet: nobena od teh funkcij ne spreminja niza, vse zgolj vrnejo nov niz. Če hočem, da bo vrstica vsebovala niz, zapisan s samimi malimi črkami, potrebujem

```
vrstica = vrstica.lower()
```

```
vrstica
```

```
'nnakamura'
```

Zavoljo popolnosti povejmo še za eno metodo: `casefold`. Ta je podobna `lower`, vendar naredi še nekateri druge pretvorbe, recimo zamenja *ß* z *ss*, saj je "Straße" za tiste, ki znajo nemško, isto kot "strasse". Poleg tega, recimo, poskrbi, da so vsi šumniki zapisani na enak način (zapisati jih je mogoče z enim znakom ali kot kombinacijo *c/s/z* s strešico).

```
"Straße".casefold()
```

```
'strasse'
```

Zamenjava podnizov

Tudi metoda `replace` ima nerodno ime, saj ne spreminja niza, temveč vrne niz, v katerem je nek znak ali podniz zamenjan z drugim.

```
vrstica.replace("a", "e")
```

```
'nnekemure'
```

```
vrstica.replace("mu", "mjav")
```

```
'nnakamjavra'
```

```
vrstica.replace("am", "")
```

```
'nnakura'
```

Kar potrebujemo mi, je zamenjava dveh *nn*-jev z enim:

```
vrstica.replace("nn", "n")
```

```
'nakamura'
```

To dela tudi za Canyon:

```
"canyon".replace("nn", "n")
```

```
'canyon'
```

```
"canyonnn".replace("nn", "n")
```

```
'canyon'
"cannyonn".replace("nn", "n")
'canyon'
```

Vse skupaj

Ker vsaka od teh metod vrne nov niz, jih lahko "veriziramo".

```
vrstica = "NNakamura\n"
vrstica.strip().lower().replace("nn", "n")
'nakamura'
```

In zdaj lahko končno preštejemo te bicikle:

```
cube = stevens = nakamura = canyon = 0
for vrstica in open("../vaje/kolesa.txt"):
    vrstica = vrstica.strip().lower().replace("nn", "n")
    if vrstica == "cube":
        cube += 1
    if vrstica == "stevens":
        stevens += 1
    if vrstica == "nakamura":
        nakamura += 1
    if vrstica == "canyon":
        canyon += 1

print("Cube:", cube)
print("Stevens:", stevens)
print("Nakamura:", nakamura)
print("Canyon:", canyon)

Cube: 46
Stevens: 1
Nakamura: 23
Canyon: 30
```

Začetek in konec

Nizi imajo še par ducatov metod, vendar večine bodisi še ne moremo razumeti, bodisi niso prav splošno uporabne. Tu bomo navrgli le še dve, potem pa posvetili daljši razdelek zvezdi današnjega predavanja, metodi `split`.

Preprosti, a pogosto uporabni metodi sta `startswith` in `endswith`. Podamo jima niz in vrneta `True` ali `False`.

```
ime = "Benjamin"
```

```

ime.startswith("Ben")
True
ime.startswith("B")
True
ime.startswith("")
True
ime.startswith("tisto zgoraj je malo hecno")
False
ime.endswith("min")
True
ime.endswith("max")
False

```

Obe metodi bomo - kot vse, ki vračajo `False` ali `True` - seveda najpogosteje uporabljali v pogojih v `if` (in `while`, ko ga bomo poznali).

Razbijanje niza

Hja. V resnici je v direktoriju *vaje* samo precej poenostavljena različica datoteke s kolesarsko statistiko. V resnici je videti tako:

```

Cube,51,1216
Canyon,104,444
Cube,74,1508
Cube,79,936
Nakamura,17,95
(in tako naprej)

```

V vsaki vrstici so trije podatki, ločeni z vejicami: najprej ime kolesa, ki ga je imel neznani kolesar nek dan za izlet, nato prevožena razdalja in potem višinski metri. Kako prebrati to?

Za začetek vzemimo le eno vrstico.

```
vrstica = "Cube,51,1216"
```

Doslej zamolčana metoda nizov je `split`. Ta razbije nize na več nizov; kot argument ji podamo ločilo - v našem primeru vejico.

```
vrstica.split(",")
['Cube', '51', '1216']

```

Metoda je vrnila tri nize. Kaj pomenijo tisti oglati oklepaji, ni današnja tema. Motili nas ne bodo. Ker imamo tri nize, jih bomo priredili trem spremenljivkam.

```
kolo, razdalja, visina = vrstica.split(",")
```

Razumemo? Desno od enačaja imamo tri stvari, torej levo od njega potrebujemo tri spremenljivke.

Njihove vrednosti so, kot morajo biti.

```
kolo
```

```
'Cube'
```

```
razdalja
```

```
'51'
```

```
visina
```

```
'1216'
```

Pa ugotovimo, kako daleč se je možakar peljal in kakšna je skupna višina njegovih izletov.

```
skupna_razdalja = 0
skupna_visina = 0
for vrstica in open("kolesa.txt"):
    kolo, razdalja, visina = vrstica.split(",")
    skupna_razdalja += int(razdalja)
    skupna_visina += int(visina)
```

```
print("Skupna razdalja:", skupna_razdalja)
```

```
print("Skupna višina:", skupna_visina)
```

```
Skupna razdalja: 6359
```

```
Skupna višina: 98076
```

Meni se zdi to kar imenitno. Kar zapletene datoteke znamo že brati.

V zvezi s `split` je potrebno povedati še nekaj. Datoteke s podatki, ločene z vejicami, pogosto dobimo iz kakega Excela. Včasih pa so podatki ločeni s tabulatorji ali celo le s presledki - se pravi, belim prostorom. V tem primeru pokličemo `split` brez argumentov.

```
"Cube 45 1024".split()
```

```
['Cube', '45', '1024']
```

To je podobno, vendar ne popolnoma isto, kot če kot argument podamo presledek.

```
"Cube 45 1024".split(" ")
```

```
['Cube', '45', '1024']
```

Razlika je v tem, kako `split` obravnava zaporedne presledke v nizu.


```
"Cube 45 1024".split()
['Cube', '45', '1024']

"Cube 45 1024".split(" ")
['Cube', '', '', '45', '', '', '', '', '1024']
```

V prvem primeru upošteva zaporedne presledke kot en sam presledek, v drugem primeru, ko smo podali argument, pa je pet zaporednih presledkov obravnaval kot pet ločil - med njimi pa so bili prazni nizi. Tega navadno nočemo. Kadar vas torej zasrbi, da bi poklicali `split(" ")`, se obrzdajte in pokličite `split()`.