

Branje datotek

Delo z datotekami v Pythonu je tako preprosto, da ga že skoraj obvladamo.

Imejmo datoteko z imenom `planeti.txt` in takšno vsebino:

```
Merkur
Venera
Zemlja
Mars
Jupiter
Saturn
Uran
Neptun
Pluton
```

Če jo hočemo brati, jo moramo najprej odpreti.

```
datoteka = open("planeti.txt")
```

Kaj se je zgodilo tule? `open` je, očitno, funkcija, ki kot argument dobi ime datoteke, ki jo hočemo odpreti. Kot rezultat vrne ... hm, datoteko? Glede na to, da smo rezultat funkcije priredili spremenljivki `datoteka`, lahko s `print(datoteka)` poškilimo vanjo in izvemo naslednje:

```
print(datoteka)
```

```
<_io.TextIOWrapper name='planeti.txt' mode='r' encoding='UTF-8'>
```

Takole: `datoteka` je spremenljivka. Ni število (tipa `int` ali `float`), tudi ni tipa niz (`str`) ali seznam (`list`), temveč neko čudo tipa `_io.TextIOWrapper`. Pač zelo čudno ime tipa, predstavlja pa neko vrsto datoteke. Nadalje izvemo, da je datoteki, na katero se nanaša spremenljivka z imenom `datoteka`, ime `'planeti.txt'`, odprta je na način `'r'` (spoiler: to pomeni, da je odprta za branje, *read*) in da je besedilo shranjeno v formatu UTF-8 (kaj je to, bomo že še izvedeli).

Tole ni bilo nič posebej informativnega. Kaj lahko počnemo z datoteko? Lahko jo prehodimo z zanko `for`.

```
for vrstica in datoteka:
    print(vrstica)
```

```
Merkur
```

```
Venera
```

```
Zemlja
```

```
Mars
```

Jupiter

Saturn

Uran

Neptun

Pluton

Če torej podtaknemo datoteko zanki `for`, jo bo zanka brala vrstico za vrstico.

A gremo še enkrat?

```
for vrstica in datoteka:
    print(vrstica)
```

Ne moremo. Ko preberemo datoteko do konca, je prebrana. Če jo hočemo ponovno brati, jo moramo tudi ponovno odpreti. (To ni povsem res, vendar je pri besedilnih datotekah *dokaj res*.)

Datoteke se v bistvu vedejo kot generatorji, ki smo jih spoznali pred dvema tednoma. `next` pa te reči.

Zakaj pa so vmes prazne vrstice? V datoteki jih vendar ni. Vse bo jasno, če datoteko preberemo nekoliko drugače. Kot vemo od prejšnjih predavanj, imajo sezname in nizi razne metode, ki z njimi počnejo različne stvari. Prav tako imajo svoje metode datoteke. Ena od njih je `read`, ki prebere celotno datoteko in vrne njeno vsebino kot niz.

```
datoteka = open("planeti.txt")
datoteka.read()
```

```
'Merkur\nVenera\nZemlja\nMars\nJupiter\nSaturn\nUran\nNeptun\nPluton'
```

(Mimogrede, metodi `read` lahko kot argument podamo številko, ki pove, koliko znakov datoteke naj prebere. Če jo pokličemo brez argumentov, pa prebere celo datoteko.)

Vidimo, da datoteka, jasno, vsebuje znake `\n`, ki povedo, kje se začneja nova vrstica. Ko beremo datoteko z zanko `for`, dobivamo celotne vrstice, skupaj z `\n`, torej, recimo `'Merkur\n'`. Če poskusimo izpisati `print('Merkur\n')`, na koncu niza izpišemo prazno vrstico, še eno pa sam od sebe doda `print`. Odtod prazne vrstice.

Dva nauka. Prvi: `strip` je tvoj prijatelj - še prav posebej pri branju datotek.

```
datoteka = open("planeti.txt")
```

```
for vrstica in datoteka:
    print(vrstica.strip())
```

Merkur
Venera
Zemlja
Mars
Jupiter
Saturn
Uran
Neptun
Pluton

Drugi: kadar po vrsticah berete besedilne datoteke, jih berite z zanko `for`. Torej tako

```
for vrstice in datoteka:
```

in ne tako

```
for vrstice in datoteka.read().split("\n"):
```

Drugo ni samo daljše in nepotrebno, temveč lahko tudi ne deluje. Če si že morate brez potrebe zapletati življenje, potem uporabite vsaj

```
for vrstice in datoteka.read().splitlines()
```

Poleg metode `read` ima datoteka še kar nekaj metod, med katerimi za zdaj omenimo `readline`, ki prebere eno vrstico, in `readlines`, ki prebere vse vrstice in vrne seznam.

Datoteke vedno beremo po vrsti. Kot pove njihovo ime, datoteke tečejo.

```
datoteka = open("planeti.txt")
```

```
datoteka.readline()
```

```
'Merkur\n'
```

```
datoteka.readline()
```

```
'Venera\n'
```

```
datoteka.readline()
```

```
'Zemlja\n'
```

```
datoteka.readlines()
```

```
['Mars\n', 'Jupiter\n', 'Saturn\n', 'Uran\n', 'Neptun\n', 'Pluton']
```

Najprej smo prebrali tri vrstice in nato ostale. Vračanja nazaj ni. (Je, ampak ... pustimo.) In ko smo na koncu, smo na koncu.

```
datoteka.readline()
```

```
''
```

```
datoteka.readlines()
```

[]

Spodobi se, da povemo še ze metodo: `close`. Z njo datoteko zapremo. Zapiranje je potrebno kadar želimo operacijskemu sistemu (da, ne Pythonu, temveč Windowsom oz. macOS-u oz. Linuxu) povedati, da te datoteke ne beremo več. To je pomembno predvsem zato, ker bi se kak drug program morda odločil v datoteko pisati, vendar mu sistem tega ne bo pustil, če je datoteka odprta.

V Pythonu zapiranje datoteke ni povsem nujno. Če datoteko odpremo v funkciji, se bo kar sama od sebe zaprla, ko bo funkcije konec. Pravijo, da na to ni lepo računati, vendar vsaj sam to pogosto počnem. Datoteko rad odprem kar v zanki `for`, takole

```
for vrstice in open("planeti.txt"):
```

Saj res ni potrebe, da bi jo tlačili v kakšno posebno spremenljivko.

To je pomembno predvsem zato, ker bi se kak drug program morda odločil v datoteko pisati... Pa znamo mi pisati v datoteko?

Pisanje datotek

Funkcija `open` sprejme poleg imena datoteke, ki jo želimo odpreti, še nekaj argumentov. Drugi po vrsti vsebuje način, na katerega želimo odpreti datoteko. Ta je lahko "r", "w" ali "a". Način "r" smo že videli zgoraj in vemo, da se nanaša na branje. Potem je očitno, da se "w" na pisanje. Način "a" pa pomeni dodajanje na konec datoteke "a".

Če torej napišemo

```
datoteka = open("bogovi.txt", "w")
```

ustvarimo novo datoteko z imenom *bogovi.txt*. Če je takšna datoteka že obstajala ... tough luck. Zdaj obstaja prazna datoteka s tem imenom. Da uganemo, kako se imenuje metoda za pisanje v datoteko, ni potrebno biti Einstein.

```
datoteka.write("Hermes")
```

6

6?! Zakaj je metoda `write` vrnila 6? Povedala je, koliko znakov je napisala v datoteko. Zakaj? Pojma nimam.

```
datoteka.write("Afrodita")
```

8

```
datoteka.write("Ares")
```

4

```
datoteka.write("Zeus")
```

4

```
datoteka.write("Kronos")
```

6

Pri vseh teh vrnjenih številkah je smešno to, da Python v datoteko v resnici (najbrž) še ni napisal ničesar. Če odpremo datoteko, bomo najbrž odkrili, da je prazna. V resnici ne piše vsake drobnarije posebej, temveč čaka, da se bo malo nabralo. Zagotovo pa bo zapisal, vse kar je treba, če datoteko zapremo.

```
datoteka.close()
```

Po tem ne moremo več pisati vanjo.

```
datoteka.write("Caelus")
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-20-b7e03db93281> in <module>  
----> 1 datoteka.write("Caelus")
```

ValueError: I/O operation on closed file.

Pač pa lahko datoteko odpremo s PyCharmom ali čim podobnim. In uvidimo, da v njej piše:

```
HermesAfroditaAresZeusKronosKronos
```

Ups. Ja, `write` ni `print` (hvalabogu). `write` ne dodaja praznih vrstic.

Popravni izpit:

```
datoteka = open("bogovi.txt", "w")  
datoteka.write("Hermes\n")  
datoteka.write("Afrodita\n")  
datoteka.write("Ares\n")  
datoteka.close()
```

Če zdaj odpremo datoteko, dobimo, kar smo (najbrž) želeli:

```
Hermes  
Afrodita  
Ares
```

Metoda `write` se od `printa` razlikuje še po nečem: medtem ko `print` dobi poljubno število argumentov poljubnega tipa, sprejme `write` kot argument en in natančno en niz. Če bi hoteli *izpisati* prvih deset števil in njihove kvadrate, bi pisali

```
for i in range(10):  
    print(i, i ** 2)
```

```
0 0  
1 1  
2 4
```

```
3 9
4 16
5 25
6 36
7 49
8 64
9 81
```

Če bi jih hoteli zapisati v datoteko, bi jih morali najprej preoblikovati v nize.

```
k = open("kvadrati.txt", "w")
for i in range(10):
    k.write(f"{i} {i ** 2}")
k.close()
```

Trenutni direktorij

Kje mora biti datoteka, da jo bo `open` našel? Ali, kadar pišemo: kam, v kateri direktorij, jo bo shranil?

Vsak program, ki trenutno teče, ima določen "trenutni direktorij". V začetku mu ga na nek način določi operacijski sistem (kako, je odvisno od sistema, programa in bogvečesa še). V Pythonu ga izvemo s funkcijo `os.getcwd()`.

```
import os
```

```
os.getcwd()
```

```
'/Users/janez/Desktop/predavanja/p1/predavanja'
```

Kratika "cwd" pomeni "current working directory". Spremenimo ga z `os.chdir()`:

```
os.chdir("/Users/jananovak/Desktop")
```

Če po tem odpremo datoteko v načinu "w", se bo le-ta pojavila v direktoriju `/Users/jananovak/Desktop`, kar bi utegnilo biti Janino namizje.

Ime datoteke je lahko relativno ali absolutno.

Relativna imena začnemo brez poševnice. Če je trenutni direktorij `Users/janez/Downloads` in napišemo `open("programi/naloga.py")`, bo Python poskusil odpreti datoteko `/Users/jananovak/Desktop/programi/naloga.py`. Takšno ime je torej relativno glede na trenutni direktorij.

Če pa ime datoteke začnemo s poševnico, gre za absolutno ime, od, s kleno slovensko besedo, korenskega imenika oziroma mape. Napišemo lahko torej, na primer, `open("/Users/jananovak/Desktop/programi/naloga.py")`.

Ob poševnicah povejmo še, da dandanes vsi operacijski sistemi sprejemajo običajno, "napredno" poševnico, `/`. Windowsi (oziroma MS-DOS, na katerem temeljijo) so dolgo uporabljali vzvratno poševnico `\`, ki pa povzroča same sitnosti - kot smo izvedeli, ko smo govorili o nizih.

Kako je zapisano besedilo

Računalnik v resnici shranjuje samo številke. Da lahko shranimo besedilo, se zmenimo, da vsak znak predstavimo z določeno številko. V ta namen je bilo razvitih več standardov, od katerih je preživel bolj ali manj samo ASCII. Ta pravi, recimo, da veliki A predstavimo s številko 65, B s 66, C z 67... Mali a je 97, b 98 ... in tako naprej. Svoje kode imajo tudi drugi znaki: presledek je 32, pika je 46, oklepaj 40... Vsak znak, ki ga lahko natipkamo ali kako drugače napišemo, mora imeti neko kodo. Problem: ASCII je (bil) sedembitni standard, pri čemer so prvih 32 znakov zasedle kontrolne kode, kot recimo prehod v novo vrsto, tabulator, zvonček in še kup eksotičnih reči, ki so jih potrebovali za teleprinterje. Tako so proste le kode od 32 do 127, s čimer lahko zapišemo 96 znakov. To nekako zadošča za angleško abecedo, ločila in še par malenkosti, za trde in mehke č-je pa se Američani, ki so sestavljali standard, niso posebej menili. Kitajci pa si niti s celotnim naborom 96 mest ne bi mogli veliko pomagati.

ASCII je doživel kup razširitev. Že v sedembitni različici smo ga v rajnki domovini popravili v YUSCII, v katerem smo žrtvovali nekaj znakov, kot so zaviti oklepaji in vzvratne poševnice in jih zamenjali s šumniki. Microsoft in IBM sta kasneje sestavila vsak svoj standard, takoimenovane kodne tabele; gre za razporede, v katerih je "spodnji del", znaki s kodami do 127, enak kot v izvirnem ASCII, v gornjem delu - kode od 128 do 256 - pa so različni drugi znaki. V naših krajih se je najbolj prijela tabela CP1250, namenjeni srednje- in vzhodnoevropskim jezikom, v kateri ima, recimo, č kodo 232. V zahodnoevroski tabeli, CP1252, č-ja ni, na mestu 232 pa je znak è. Poleg teh, Microsoftovih tabel, obstajajo IBMove, kjer je Slovanom namenjen IBM 852; da bi bilo programiranje še preprostejše, obstaja tudi tretji standard, ISO-8859, ki nam je namenil tabelo ISO-8859-2, ki je podobna, vendar ne povsem enaka CP1250.

Ko torej računalnik prikazuje kako besedilo, mora vedeti, v katerem kodnem naboru je zapisan, se pravi, kako razumeti, recimo, številko 232. Če je besedilo v CP1252, je to è, če v CP1250, pa č. In kako naj to ve? V splošnem ne more! Spletne strani navadno vsebujejo glavo, v kateri je zapisan podatek o uporabljenem kodnem naboru. Če tega ni, se bodo šumniki pokazali prav ali pa tudi ne. In še huje: vse besedilo je napisano v istem kodnem naboru, zato se lahko zgodi, da ne bo moglo vsebovati tako črke č kot è. (To morda razloži kaj v zvezi s podnapisi filmov, ki napačno sugerirajo, da so na pikniku jedle *èvapèièè s èebulo?*)

Obenem pa s tem Kitajci še vedno ne morejo biti zadovoljni, saj jim dodatnih 128 znakov bore malo pomaga.

Takole smo se hecali do pred nekaj leti, ko je prišel v širšo rabo standard Unicode. Ta je narejen takole: vsakemu znaku je prirejena ena koda. Kod je 232, kar je čez in čez dovolj tudi za Kitajce (vsak ima lahko svojo, če hoče!). Nekatere znake je mogoče pisati tudi na več načinov: obstaja, recimo, znak za strešico in č lahko zapišemo kot č (koda 269) ali kot strešico (309) in c (99). Da bi lahko zapisal 232 kod, bi potrebovali po štiri bajte na znak. Američanov to ne bi posebej

osrečilo, saj bi se dolžina njihovih besedil početrila in bi se spraševali, kaj je bilo pravzaprav narobe s starim dobrim ASCII. Zato so se snovalci standarda domislili različnih načinov, kako "stisniti" zapis, da bo zahteval manj kot štiri bajte na znak. Med njimi je daleč najbolj razširjen UTF-8, občasno pa vidimo še UTF-16. UTF-8 je sestavljen zelo zvito: če imamo besedilo v starem dobrem ASCIIju (brez neangleških znakov, torej brez kakšnih šumnikov ali francoskih naglasov), se lahko delamo, da je v UTF-8. Čim vsebuje šumnike (recimo po standardu CP1250), pa ni več združljiv z UTF-8 in moramo, če ga hočemo pravilno prebrati, vedeti, po katerem standardu je kodirano.

Smo prišli z dežja pod kap? Smo imeli prej polno kodnih naborov (CP1250 do CP1258, bogve koliko, pa IBMovi nabori, pa ISO-8859), zdaj pa imamo kup UTF-jev? Niti ne. Razlika je v tem, da so stari nabori določali, kakšne kode imajo posamezni znaki. Po novem imajo vsi znaki določene kode, obstaja le par načinov zapisa teh kod. Med njimi pa vsaj na zahodu prevladuje UTF-8.

Če imamo torej neko besedilo - recimo v neki datoteki, ali pa na neki spletni strani -, ki ga želimo prebrati, bo ponavadi zapisano v UTF-8 ali CP1250. Če je angleško in vsebuje le angleške znake, je to eno in isto. Če gre za slovensko besedilo, pa moramo vedeti, kako je zapisano. Če gre za novejšo besedilo, je verjetno v UTF-8, če je starejše kot kakih deset let ali pa prihaja iz vira, s katerim ima kaj opraviti kakšen domači amaterskih izdelovalec spletnih strani, pa CP-1250. CP-1250 se žal še vedno kar pogosto pojavlja tudi v bližini Windowsov. Vedno ga lahko tudi poskusimo najprej prebrati kot UTF-8; če to uspe, je najbrž v resnici UTF-8, sicer ga preberemo kot CP1250.

Kako Pythonu povemo, kako naj bere besedilo? Metoda `open` ima še par argumentov. Tretji nas ne zanima in naj ima vrednost `-1`, četrti pa poda vrsto zapisa. Ime mu je `encoding`. Da preskočimo tretjega, `encoding` navadno podamo poimensko.

```
open("planeti.txt", "r", encoding="utf8")
```

```
<_io.TextIOWrapper name='planeti.txt' mode='r' encoding='utf8'>
```

Pa če Pythonu ne povemo, kot kaj želimo brati besedilo - kot kaj ga bo poskusil brati? To je odvisno od nastavitve sistema - zato imajo Windowsi nastavek "System locale". Tam sprašuje po "Current language for non-Unicode programs"; če imate nastavljeno slovenščino, bo privzeti kodni nabor CP1250. Na Ubuntuju in Macu je kot privzeto kodiranje nastavljen UTF-8. Kako je s tem pri vas, vam pove funkcija `locale.getpreferredencoding()`.

Kaj pri branju in pisanju datotek pravzaprav naredi ta nastavek? Tole: ko zapišemo nek niz v datoteko, ga mora Python pretvoriti v zaporedje števil. Ta nastavek pove, kako - v katero številko naj se spremeni posamezni znak. (Pozoren študent, ki kaj ve, ve tudi, da so tudi nizi v Pythonu, torej spremenljivke tipa `str`, že shranjene kot zaporedje številke. Drži, vendar Python interno shranjuje nize pač v nekem standardu in nič nas ne briga, v katerem (v resnici v UTF-16 ali UTF-32, saj bi bil UTF-8 za tole zelo nepraktičen), pri zapisovanju v

datoteko pa pretvori v tak standard, kot želimo. Pri branju pa je ravno obratno: ko Python bere datoteko, mora spreminjati številke v znake niza; ta nastavitev pove, kako.

Tule bi se lahko pogovorili še marsikaj. Predvsem ne vsebujejo vse datoteke besedil. Vendar so te stvari že precej tehnične in nekatere tudi specifične za Python, tako da se tu morda ustavimo. Več pa v dodatnih zapiskih