Datoteke

Osnove programiranja

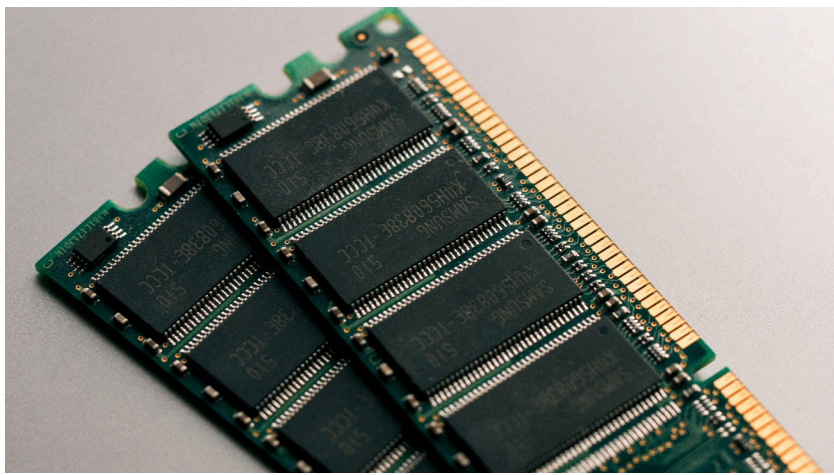
Nejc Ilc

Kje živijo podatki?

Naprave za shranjevanje podatkov

- glavni pomnilnik

Pravimo mu tudi RAM (ang. *random access memory*). Ko računalnik ugasnemo, se vsi podatki v glavnem pomnilniku izgubijo. Hm …



Fotografija: Harrison Broadbent [vir]

- magnetni disk

Pogost v starejših sistemih ali v podatkovnih centrih (ker je poceni). Podatki, zapisani na njem, se ohranijo tudi po izklopu računalnika. Je med "počasnejšimi".

Fotografija: Benjamin Lehman [vir]

Kje živijo podatki?

Naprave za shranjevanje podatkov

- polprevodniški pogon

Pravimo mu tudi SSD (ang. *solid state drive*). Je bistveno hitrejši od magnetnega diska in nima premikajočih delov. Ohišje 2.5" ali M.2.

Fotografija: Gigazine [vir]

- izmenjljive naprave

Kartice SD, "ključki USB", zunanji diski, … Temeljijo na podobni tehnologiji kot polprevodniški pogoni - blikovni pomnilnik (ang. *flash memory*).

Fotografija: Photo Mix [vir]

Datotečni sistem

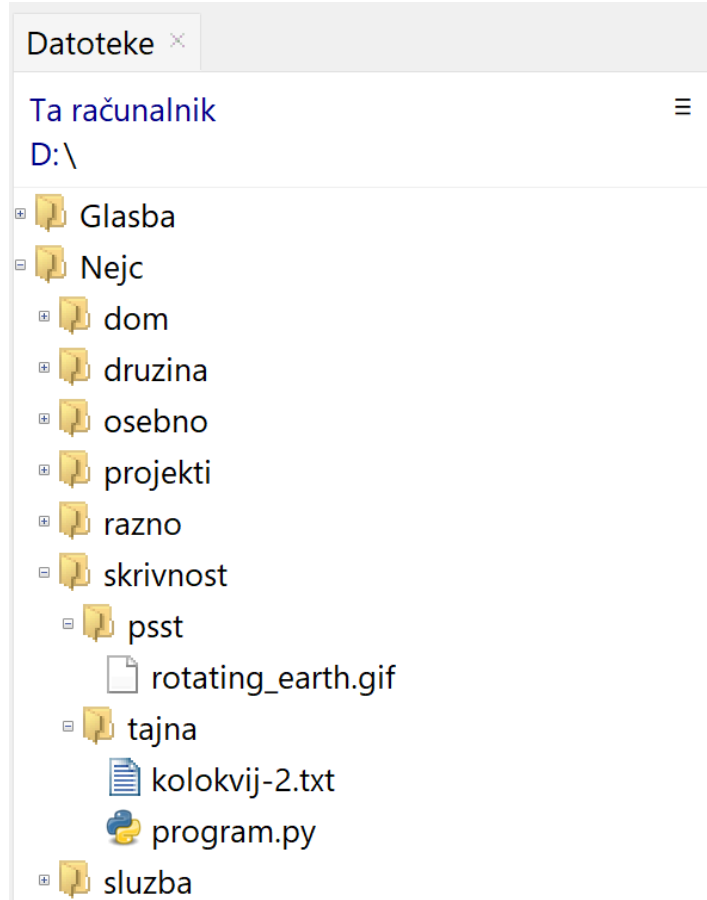
Zemljevid shrambe podatkov

Naprave za shranjevanje podatkov (glavni pomnilnik računalnika v to kategorijo ponavadi ne spada) morajo vedeti, kje je zapisan določen podatek.

Podatke pakiramo v enote, ki jim pravimo *datoteke*. Poznamo navadne datoteke, ki imajo svoje ime in končnico (`.txt`, `.py`, `.docx`, `.jpg`, `.png`, `.pdf`, ...) - ta oznanja njihovo vrsto.

Obstajajo pa tudi malo drugačne datoteke, ki so vsebniki za navadne datoteke - pravimo jim *imeniki* oziroma *mape*.

Datoteke so urejene v *drevesno strukturo*: korenina je ime pogona (v Windows recimo `D:`), vsak imenik je razcep veje, navadne datoteke so listi drevesa. To je torej datotečni sistem.



Kaj je v datoteki?

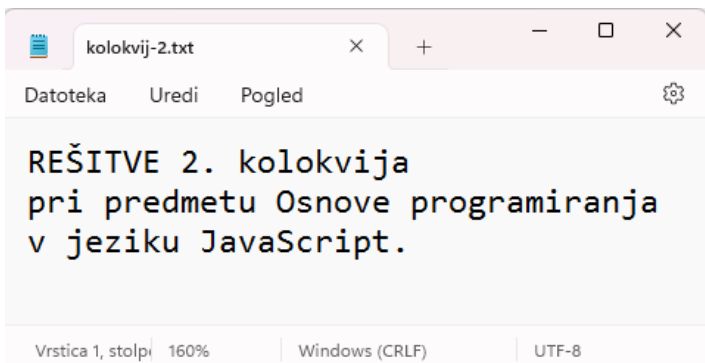
Odprimo dva tipa datotek in poglejmo

`kolokvij-2.txt`

Do te datoteke pridemo po tej *poti*:

`D:\Nejc\skrivnost\tajna\kolokvij-2.txt`

Ko dvokliknemo na datoteko, se nam odpre program Beležnica (*Notepad*), ki prikaže vsebino:



`rotating_earth.gif`

Pot do datoteke:

`D:\Nejc\skrivnost\psst\rotating_earth.gif`

Kaj je *zares* v datoteki `kolokvij-2.txt`?

Vsebina datoteke je v shrambi zapisana z ničlami in enicami, torej z *biti*. Preberimo prvih 96 bitov.

```
01010010 01000101 11000101 10100000 01001001 01010100
01010110 01000101 00100000 00110010 00101110 00100000
```

Osem bitov skupaj tvori en *bajt* (oktet, ang. *byte*).

Zapišimo krajše s šestnajstiškimi števili:

```
52 45 c5 a0 49 54 56 45 20 32 2e 20
```

V desetiškem številskem sistemu je to enako:

```
82 69 197 160 73 84 86 69 32 50 46 32
```

Sedaj pa te vrednosti bajtov izpišimo kot znake (funkcija `print()` tako dela):

```
RE\xc5\xa0ITVE 2.
```

Prvi znak je `'R'`, ki ga kodiramo s številko `82` (šestnajstiško `52`, dvojiško `01010010`).

V Pythonu lahko pretvarjamo med različnimi interpretacijami - med znaki in števili:

```
>>> ord('R')
82
>>> chr(82)
'R'
```

V datoteki so torej biti, ki pa jih lahko interpretiramo tudi kot znake. Ali pa kaj drugega.

Kaj je *zares* v datoteki `rotating_earth.gif`?

Zapis z biti:

```
01000111 01001001 01000110 00111000 00111001 01100001
10010000 00000001 10010000 00000001 11110111 00000000
```

Šestnajstiško:

```
47 49 46 38 39 61 90 01 90 01 f7 00
```

Pretvorba v znake:

```
GIF89a\x90\x01\x90\x01\xf7\x00
```

Vidimo, da začetek datoteke vsebuje števila, ki jih lahko pretvorimo v berljive znake `GIF89a`, temu pa sledijo bajti, ki imajo "čudne" vrednosti, npr. `90`, `01` ipd.

Datoteka `.gif` ni *besedilna datoteka*, ampak je mišljena za shranjevanje slikovnega gradiva. Pravimo, da je *binarna datoteka*.

Binarnih datotek ponavadi ne skušamo interpretirati kot besedilo, ampak jih s prikazujemo/obdelujemo s posebnimi programi, ki znajo brati *format* te datoteke. V primeru datoteke GIF ima prvih nekaj bajtov naslednji pomen:

- prvih 6 bajtov govori o različici formata datoteke GIF, tukaj gre za zadnjo različico `GIF89a`.
- naslednja dva bajta pomenita širino slike. Sestavimo jih od leve proti desni: `int('0190', 16) → 400` slikovnih točk
- naslednja dva bajta pomenita višino slike ...

Delo z datotekami v Pythonu

Odpiranje, branje, pisanje, zapiranje

Dogovor: obravnavali bomo samo besedilne datoteke - to so datoteke, katerih vsebino lahko uspešno interpretiramo kot znake.

Tipično zaporedje pri delu z datotekami gre tako:

1. Datoteko odpremo s funkcijo `open()`.
2. Iz datoteke beremo z metodo `read()` ali vanjo pišemo z `write()` oziroma `print()`.
3. Datoteko zapremo z metodo `close()`.

Pravilo: kar odpreš tudi zapri.

Če pustimo datoteko odprto, do nje ne morejo dostopati drugi procesi.

`na_hitro.py`: odprimo datoteko `kolokvij-2.txt`, preberimo njeno vsebino in jo izpišimo na zaslon:

```
# Pot do datoteke (relativna glede na pot programa)
pot = 'skrivnost/tajna/kolokvij-2.txt'
# open() vrne objekt, ki predstavlja tok podatkov.
# Če ne podamo drugih argumentov, bo datoteka odprta
# samo za branje.
datoteka = open(pot)
# Z metodo read() preberemo vsebino datoteke v niz.
vsebina = datoteka.read()
# Izpišemo vsebino na zaslon
print(vsebina)
# Zapremo datoteko
datoteka.close()
# Izpis:
```

```
REŠITVE 2. kolokvija
pri predmetu Osnove programiranja
v jeziku JavaScript.
```

Stavek `with` stavek_with.py

Za varno delo z datotekami

Na prejšnji strani smo videli kratek primer programa za branje datoteke. Navadno pa v Pythonu tega ne počnemo tako. Kaj rado se namreč zgodi, da pride pri odpiranju/branju/pisanju datoteke do napake. V tem primeru se zapiranje datoteke ne bo zgodilo, kar je slabo.

Zakaj slabo? Če datoteke po uporabi ne zapremo, lahko ostane *zaklenjena* in drugi procesi ne morejo delati z njo. Lahko se tudi zgodi, da se izgubijo podatki, ki smo jih želeli pisati v datoteko.

Da v primeru napak datoteke ne puščamo odprte, je bil iznajden stavek `with`, ki poskrbi, da se datoteka zapre, ko se konča blok kode v stavku ali ko pride do napake.

```
pot = 'skrivnost/tajna/kolokvij-2.txt'

with open(pot) as datoteka:
    vsebina = datoteka.read()
    print(vsebina)
```

Ko se zaključi blok stavka `with`, se datoteka samodejno zapre. Pravimo, da sprostimo vire, ki smo jih zasedli z odpiranjem datoteke. Tudi drugi procesi zdaj lahko odpirajo, spreminjajo ali brišejo to datoteko.

Pri delu z datotekami bomo torej uporabljali stavek `with`.

Odpiranje datoteke

Fukcija `open()`

`'r'`: samo za branje

Če želimo besedilno datoteko odpreti samo za branje, je dovolj, da funkciji `open()` podamo pot do datoteke. Pot je lahko podana:

- absolutno; gremo od korenine do lista

  `pot = 'D:/Nejc/skrivnost/tajna/kolokvij-2.txt'`

- relativno glede na mapo, iz katere kličemo naš program; če smo denimo v mapi `D:/Nejc/skrivnost/tajna`:
  `pot = 'kolokvij-2.txt'`

Denimo, da se nahajamo v mapi `D:/Nejc`. (Mimogrede, na operacijskem sistemu Windows bi to pot napisali tudi kot `D:\Nejc`.) Definirajmo relativno pot do datoteke in jo odprimo za branje:

```
pot = 'skrivnost/tajna/kolokvij-2.txt'
datoteka = open(pot)
```

Z argumentom `mode` lahko eksplicitno povemo, na kakšen način želimo odpreti datoteko. Za branje je to tako:

```
datoteka = open(pot, mode='r')
```

Če datoteka, na katero kaže `pot`, ne obstaja, dobimo napako `FileNotFoundError`.

Odpiranje datoteke

Fukcija `open()`

`'w'`: samo za pisanje

Če želimo v datoteko pisati, moramo argument `mode` funkcije `open()` nastaviti na `'w'`.

```
datoteka = open(pot, mode='w')
```

Delovanje:

- če datoteka, podana s `pot`, še ne obstaja, se bo ustvarila,
- če že obstaja, se bo prepisala. Njena stara vsebina se bo torej izbrisala. Pisali bomo na začetek datoteke.

`'a'`: samo za pisanje na konec

Če želimo dodajati vsebino na konec datoteke, argument `mode` določimo kot `'a'`.

```
datoteka = open(pot, mode='a')
```

Delovanje:

- če datoteka, podana s `pot`, še ne obstaja, se bo ustvarila,
- če že obstaja, se njena vsebina ne bo izbrisala, ampak bomo pisali na konec datoteke.

Odpiranje datoteke

Za branje in pisanje

Datoteko lahko odpremo za branje in pisanje hkrati: drugemu argumentu (`mode`) funkcije `open()` dodamo `+`.

`open(pot, 'r+')`

- Če datoteka `pot` še ne obstaja, dobimo napako.
- Če datoteka `pot` že obstaja, ne izbriše njene vsebine.
- Ob odprtju datoteke smo na njenem začetku.

`open(pot, 'w+')`

- Če datoteka `pot` še ne obstaja, jo ustvari.
- Če datoteka `pot` že obstaja, izbriše njeno vsebino.
- Ob odprtju datoteke smo na njenem začetku.

`open(pot, 'a+')`

- Če datoteka `pot` še ne obstaja, jo ustvari.
- Če datoteka `pot` že obstaja, ne izbriše njene vsebine.
- Ob odprtju datoteke smo na njenem koncu.

Odpiranje datoteke `kodiranje.py`

Še nekaj pomembnega

Kodiranje znakov

Če smo bili pozorni, je bil izpis pri začetnem kratkem primerčku nekoliko "pokvarjen". Črka 'Š' ni izpisana pravilno.

```
pot = 'skrivnost/tajna/kolokvij-2.txt'

with open(pot, mode='r') as datoteka:
    vsebina = datoteka.read()
    print(vsebina)
```

Izpis:

```
REĹ ITVE 2. kolokvija
pri predmetu Osnove programiranja
v jeziku JavaScript.
```

Kodiranje znakov je postopek, ki določenemu znaku priredi številčno vrednost. Poznamo ogromno možnih načinov (npr. `'ascii'`, `'cp1250'`, `'utf8'`, `'utf16'`). Pri odpiranju datoteke je zaželeno, da določimo kodiranje, v katerem je zapisana datoteka.

```
pot = 'skrivnost/tajna/kolokvij-2.txt'
with open(pot, mode='r', encoding='utf8') as datoteka:
    vsebina = datoteka.read()
    print(vsebina)
```

Izpis:

```
REŠITVE 2. kolokvija
pri predmetu Osnove programiranja
v jeziku JavaScript.
```

Branje iz datoteke `branje.py`

`datoteka.read()`

Vrne vsebino datoteke kot niz.

```
with open(pot, mode='r', encoding='utf8') as datoteka:
    vsebina = datoteka.read()
    # Nočemo dodatne nove vrstice na koncu vsebine
    # datoteke, zato end=''
    print(vsebina, end='')
```

`datoteka.readlines()`

Vrne vsebino datoteke kot seznam vrstic.

```
with open(pot, mode='r', encoding='utf8') as datoteka:
    seznam_vrstic = datoteka.readlines()
    for vrstica in seznam_vrstic:
        print(vrstica, end='') # Nočemo dodatnega \n
```

`datoteka.readline()`

Vrne naslednjo vrstico kot niz.

```
with open(pot, mode='r', encoding='utf8') as datoteka:
    prva_vrstica = datoteka.readline()
    vrstica = prva_vrstica
    # Na koncu datoteke bomo dobili prazno vrstico
    while vrstica:
        print(vrstica, end='')
        vrstica = datoteka.readline()
```

Zanka `for`

```
with open(pot, mode='r', encoding='utf8') as datoteka:
    for vrstica in datoteka:
        print(vrstica, end='')
```

Pisanje v datoteko `pisanje.py`

Ustvarimo nekaj novega

Če želimo v datoteko pisati, jo moramo najprej odpreti za pisanje. Argument `mode` funkcije `open()` je lahko `'w'`, `'a'`, `'r+'`, `'w+'`, ali `'a+'`. Vsak ima seveda svoje finte.

`datoteka.write(niz)`

Niz lahko v datoteko pišemo z metodo `write()`.

Najprej odprimo datoteko samo za pisanje.

```
pot = 'nekaj-novega.txt'
with open(pot, mode='w', encoding='utf8') as datoteka:
    datoteka.write('Moja nova štorija')
```

Če datoteka `nekaj-novega.txt` še ne obstaja, jo zgornji klic funkcije `open()` ustvari. Z metodo `write()` v datoteko zapišemo niz `'Moja nova štorija'`.

Sedaj ponovno odprimo datoteko samo za pisanje. Ker že obstaja, se bo njena vsebina izbrisala.

```
with open(pot, mode='w', encoding='utf8') as datoteka:
    datoteka.write('Tvoja zgodba')
```

Datoteka ima sedaj vsebino `Tvoja zgodba`.

Dodajmo še nekaj na konec datoteke (`mode='a'`).

```
with open(pot, mode='a', encoding='utf8') as datoteka:
    datoteka.write('\nje super.')
```

Vsebina datoteke:

```
Tvoja zgodba
je super.
```

Pisanje v datoteko

pisanje.py

Pišemo lahko tudi s `print()`

`print(niz, file=datoteka)`

V datoteko lahko pišemo tudi s funkcijo `print()`. Ima namreč argument `file`, ki ga do sedaj še nismo spoznali. Z njim povemo, kam naj se naredi izpis.

Privzeto je `file=None`, kar funkcija `print()` razume kot izpis na *standardni izhod* - to je v terminal. `None` se v resnici znotraj funkcije nadomesti s `sys.stdout`, ki je objekt za opis standardnega izhoda. Kratek dokaz:

```
>>> import sys
>>> print('Juhuhu', file=None)
Juhuhu
>>> print('Juhuhu', file=sys.stdout)
Juhuhu
```

Lahko pa izpis usmerimo v datoteko, ki jo moramo pred tem odpreti za pisanje.

```
recenzent = 'Marko Pišmevuh'
ocena_knjige = 4.9999999

with open(pot, mode='a', encoding='utf8') as datoteka:
    # Pišemo samo novo vrstico
    print(file=datoteka)
    # Na koncu tega izpisa je tudi nova vrstica
    print('Recenzent:', recenzent, file=datoteka)
    # U, kaj pa f-niz?
    print(f'Ocena: {ocena_knjige:.1f}.', file=datoteka)
```

Vsebina datoteke:

```
Tvoja zgodba
je super.
Recenzent: Marko Pišmevuh
Ocena: 5.0.
```

Primer `jedilnik.py`

Primer branja iz datoteke, obdelave podatkov in zapisa v datoteko

Šef restavracije sestavlja jedilnik v posebnem programu, iz katerega lahko podatke tudi izvozi v datoteko CSV. ki je videti tako:

```
01.05.2023,mesni cmoki
01.05.2023,bograč
01.05.2023,ocvrte sardele
02.05.2023,zelenjavna obara
02.05.2023,svinjski trakci z rižem
02.05.2023,ričet
03.05.2023,piščanjčji file z rižem
03.05.2023,marinirane perutničke iz peči
03.05.2023,pica klasika
04.05.2023,bučkini polpeti
04.05.2023,pašta fižol
04.05.2023,pleskavica z lepinjo
05.05.2023,file brancina
05.05.2023,puranja rulada z mlinci
05.05.2023,ričet
```

Naša naloga je, da napišemo program, ki bo prebral datoteko CSV (`jedilnik.csv`) in celoten jedilnik v lepši obliki izpisal v izhodno datoteko (`jedilnik.txt`):

```
Jedilnik od 1. 5. 2023 do 5. 5. 2023

1. 5. 2023
  - mesni cmoki
  - bograč
  - ocvrte sardele

2. 5. 2023
  - zelenjavna obara
  - svinjski trakci z rižem
  - ričet

...
```