Računalniški praktikum (fizika) - Vaje

Rok Kuk - kontakt@rokuk.org

2023-11-30

Kazalo

O strani

Na tej strani so zbrani zapiski za vaje predmeta računalniški praktikum v 1. letniku študija fizike na Fakulteti za matematiko in fiziko Univerze v Ljubljani.

Zapiski so mišljeni le kot opora pri izvajanju vaj in ne obsegajo čisto vseh obravnavanih vsebin. Zapiski zato ne morejo nadomestiti obiskovanja predavanj in vaj.

---

1 Namestitev okolja za vaje

Da lahko na svojem računalniku uporabljate Python in rešujete naloge je potrebno namestiti nekaj programov. Spodaj je opisan okvirni postopek in pogoste težave. Če imate težave, je opis problema dobro pogooglati, sicer pa lahko seveda vprašate sošolce, pišite asistentu ali postavite vprašanje na forumu.

1.1 Namestitev Pythona

1. Namestite Python (najbolje kar verzijo 3.12) s te strani (zavihek `Downloads`): https://www.python.org. Ko poženete program za namestitev, v oknu, ki se odpre, odkljukajte "Add Python 3.x to PATH". Nato nadaljujte z namestitvijo (opcija "Install now").

   Če uporabljate Windows 7 ali še starejši Windows, boste morali namestiti starejšo verzijo Pythona (npr. 3.8.9 ali manj). Najdete jo tu: https://www.python.org/downloads/

2. Preverite ali se je Python uspešno namestil. Odprite program Ukazni poziv (Windows) ali Terminal (macOS/Linux), ki je že na vašem računalniku. V okno, ki se odpre vpišite ukaz `python --version` in pritisnite tipko Enter. Če je Python uspešno nameščen, bi se vam v novi vrstici moralo izpisati `Python 3.x.y` (kjer je `x.y` verzija nameščenega Pythona).

   Če ste na Windowsu in ukaz `python --version` ne izpiše verzije, ampak se vam izpiše napaka ali "Python not found", poskusite ukaz `py --version`.

1.2 Namestitev Visual Studio Code

3. Namestite Visual Studio Code: https://code.visualstudio.com
4. Namestite Python extension. Odprite Visual Studio Code. Morda se vam bo v oknu VSCode pojavil zavihek z naslovom `Get Started`, ki ga lahko kar zaprete. Na levem robu okna kliknite na Extensions (ikona s štirimi kvadratki), vpišite `Python`, izberite `Python` in na desni kliknite `Install`. Morda se bo odprlo okno `Get Started`, ki ga lahko zaprete.

5. Dobro je, da si nekje na računalniku ustvarite mapo, v katero boste shranjevali vso vašo kodo. V VSCode v meniju File kliknite `Open Folder...` in izberite to mapo. Morda se bo pojavilo okno, ki vas sprašuje, če zaupate avtorju datotek v tej mapi: kliknite Yes, ker sebi zaupate. Ustvarite novo datoteko (meni File > New File), jo shranite (meni File > Save) in jo poimenujte `test.py` (na Windowsu izberete `Save as type: Python`). Vsebina datoteke se vam odpre kot zavihek v VSCode. Vanj vpišite `print("Pozdravljen svet!")`. V desnem zgornjem kotu bi morali imeti gumb v obliki puščice, s katerim lahko poženete napisani program. Če ga nimate, si lahko namestite extension z imenom `Code Runner`. Sicer lahko program poženete tudi tako, da desno kliknete kjerkoli v območju urejevalnika besedila in nato v meniju, ki se pojavi, izberete `Run Python File in Terminal`. Ko poženete program, bi se vam v oknu terminala moralo izpisati "Pozdravljen svet!".
6. Priporočam, da vklopite tudi "linter". To je program, ki je del VSCode in v vaši kodi sproti preverja ali ste se kje zmotili. Ne ujame vseh možnih napak, mnoge zatipke pa zazna in vas nanje opozori, še preden poženete program, tako da jih podčrta. VSCode pritisnite Ctrl+Shift+P in vpišite "linter" (brez navednic), kliknite na `Python: Select Linter`. Pojavi se meni z več možnostmi za linter. Priporočam `flake8`, ki nas poleg napak opozori tudi na kršitve priporočil za stil PEP8. Izberete ga s tipko Enter. VSCode vas bo desno spodaj obvestil, da ta linter ni nameščen; namestite ga tako, da kliknete `Install` v tem obvestilu. Po nekaj sekundah bi se moral namestiti. Lahko ga preizkusite tako, da v datoteko s končnico .py nekaj narobe napišete npr. `prnt("Pozdravljen svet!")`

VSCode ima veliko funkcionalnosti, ki nam lahko pomagajo pri programiranju. Več o tem piše v uradni dokumentaciji: https://code.visualstudio.com/docs/editor/codebasics

1.3 Namestitev paketa Numpy

Potrebovali bomo še Pythonov paket Numpy. To je neke vrste dodatek za Python, ki nam omogoča lažje delo z vektorji in tabelami. Več o tem na prihodnjih vajah. Dodatne module za Python lahko nameščamo in odstranjujemo z modulom `pip`. To je modul, ki bi se moral avtomatsko namestiti skupaj s Pythonom.

7. Najprej v Ukazni poziv / Terminal vpišite in izvršite ukaz `python -m pip --version` (če ste na Windowsu boste morda dobili napako v stilu "python ne obstaja", v tem primeru poskusite izvršiti ukaz `py -m pip --version`). Izpisati bi se vam moralo nekaj podobnega `pip X.Y.Z from ... (python 3.N.N)`.

   Več o tem na https://pip.pypa.io/en/stable/getting-started/

Če se vam izpiše to, lahko nadaljujete na 8. korak, sicer berite naprej. Če se vam izpiše nekaj v stilu "pip ne obstaja" ali "command not found", morate namestiti Pythonov modul

`pip`. To naredite z ukazom `python -m ensurepip --upgrade` (oz. na Windowsu morda `py -m ensurepip --upgrade`).

   Več informacij o nameščanju modula `pip` je na https://pip.pypa.io/en/stable/installation/#python

8. Namestite Pythonov modul numpy. To naredite z ukazom `python -m pip install numpy` (oz. na Windowsu bo morda treba uporabiti `py -m pip install numpy`).

1.4 Pogoste težave

- Če pri poganjanju programa dobite napako `no module named numpy`, to pomeni, da Numpy ni nameščen. Če ste ga že namestili, je morda težava, da ste ga namestili za napačno verzijo Pythona (glej spodnjo alinejo). Če ga še niste namestili, poskusite zgoraj opisani postopek.
- Če imate na računalniku nameščenih več verzij Pythona (to so npr. mnogi računalniki z macOS) se lahko "python" v ukazni vrstici / terminalu nanaša na drugo verzijo, kot je tista, ki jo uporabljate za poganjanje svojih programov v VSCode. Katera verzija se uporablja v VSCode lahko izberete tako, da odprete katerokoli datoteko s končnico .py in kliknete na "Python 3.x.y" na spodnjem robu okna. Pokaže se okno z vsemi nameščenimi verzijami. Da namestite Numpy za določeno verzijo Pythona lahko poskusite ukaz `python3.10 -m pip install numpy`, kjer 3.10 zamenjate z želeno verzijo Pythona. Na Windowsu bi moral delovati ukaz `py -3.10 -m pip install numpy`.

2 Uvod v Python

Gradiva za to poglavje so:

- https://automatetheboringstuff.com/2e/chapter1/
- https://automatetheboringstuff.com/2e/chapter2/ (do poglavja `while loop statements`)
- https://automatetheboringstuff.com/2e/chapter3/
- poglavje Osnovni koncepti programiranja
- poglavje Izdelava samostojnih programov in pogojni stavki
- poglavje Funkcije

2.1 Števila

Za seštevanje uporabimo `+`, za odštevanje `-`, za množenje `*`, za potence `**`, za deljenje `/`, za celoštevilsko deljenje `//`, za ostanek pri deljenju `%`. Za vrstni red računanja operacij (če jih kombiniramo) veljajo enaka pravila kot v matematiki.

```
celidel = 20 // 8
ostanek = 20 % 8
print(celidel, ostanek)
```

```
2 4
```

Za zaokroževanje števila `stevilo` uporabimo funkcijo `round(stevilo, d)`, ki število zaokroži na `d` decimalnih mest.

2.2 Logične operacije

Logične operacije s ključnimi besedami `and`, `or` in `not` ustrezajo operacijam v matematiki.

- `a and b` je True, če sta `a` in `b` enaka True, sicer je False
- `a or b` je False, le če sta `a` in `b` enaka False
- `not a` je True, če je `a` enak False, sicer je True

Logične operacije lahko kombiniramo. Vrstni red operacij lahko določimo z oklepaji. Sicer ima operator `and` prednost pred `or`, `not` pa ima prednost pred obema.

```
a = True
b = True
c = False
print((a or b) and (a or c))
```

```
True
```

V logičnih operacijah se število 0 obnaša kot False, ostala števila pa kot True.

2.3 Relacijski operatorji

Če relacija velja ima izraz vrednost True, sicer pa False.

- primerjava števil `a < b` ali `a <= b`
- preverjanje enakosti `a == b`
- preverjanje različnosti (nista enaka) `a != b`

Logične operacije in relacije so binarne. Binarna operacija se izvede med tem, kar je na levi, in tem, kar je na desni.

2.4 Nekoliko kompleksnejši primer

Če želimo preveriti ali je spremenljivka `mesec` enaka 6 ali 7, ni prav, če napišemo

```
mesec = 4
rezultat = mesec == 6 or 7
print(rezultat)
```

```
7
```

V zgornjem izrazu se najprej izvede primerjava med `mesec` in 6. Ker 4 ni enako 6, nam to da False. Nato se izvede operacija `or` med False in 7. Ker se 7 obnaša kot True bi pričakovali, da dobimo True. Operacija `or` deluje tako, da vrne prvo vrednost, ki ni False. Ponavadi primerjamo True in False vrednosti, zato ima operacija rezultat True, če je ena od vrednosti True. Na levi strani je pri nas False, na desni pa 7, zato ima celoten izraz desno od enačaja vrednost 7 (ker je prva vrednost, ki ni False).

Pravilna rešitev bi bila

```
mesec = 4
rezultat = (mesec == 6) or (mesec == 7)
print(rezultat)
```

```
False
```

3 Zanke

Gradivi za to poglavje sta

- https://automatetheboringstuff.com/2e/chapter2/ (poglavji `while loop statements` in `for loops`)
- poglavje Zanke

3.1 Zanka for

Zanko for uporabimo, ko **vemo kolikokrat želimo nekaj ponoviti**.

```
zacetek = 3
konec = 10
korak = 2
for i in range(zacetek, konec, korak):
    print(i)
```

```
3
5
7
9
```

Če tretjega parametra ne podamo, bo korak 1. Če podamo le en parameter bo šla zanka od 0 do vrednosti tega parametra.

3.2 Zanka while

Zanko while uporabimo, ko želimo zanko izvajati, **dokler je nek pogoj izpolnjen**.

```
i = 3
while i < 10:
    print(i)
    i += 2
```

```
3
5
7
9
```

V zgornjem primeru bi lahko `i < 10` nadomestili s kakršnokoli logično operacijo, ki vrne True ali False. Zanka se izvaja, dokler je pogoj True.

> **Pozor**
>
> Paziti moramo, da ne napišemo neskončne zanke, kjer je pogoj vedno True! V tem primeru se izvajanje programa samo nikoli ne konča. Izvedbo programa lahko v Ukaznem pozivu / Terminalu prekinemo s kombinacijo tipk `Ctrl+C`.

4 Seznami in nizi

Gradiva za to poglavje so:

- https://automatetheboringstuff.com/2e/chapter4/
- poglavje Tabele
- poglavje Nizi

4.1 Indeksiranje

Indeksi morajo biti cela števila, sicer pride do napake. Indeksi se začnejo z 0 (prvi element). Negativni indeksi so indeksi šteti s konca seznama proti začetku (-1 je indeks zadnjega elementa, -2 predzadnjega). Če podamo indeks, ki je večji od indeksa zadnjega elementa, Python javi `IndexError`.

```
spam = ['cat', 'bat', 42, True, 'dog']
print(spam[2])
print(spam[-2])
```

```
42
True
```

Podsezname dobimo s sintakso `seznam[zacetni_indeks:koncni_indeks:korak]`. Nobeno od teh treh števil ni obvezno.

```
spam = ['cat', 'bat', 42, True, 'dog']
print(spam[1:4:2])
print(spam[:3])
print(spam[::2])
print(spam[::-1])
```

```
['bat', True]
['cat', 'bat', 42]
['cat', 42, 'dog']
['dog', True, 42, 'bat', 'cat']
```

Podobno je za nize.

```
spam = "besedna zveza"
print(spam[2])
print(spam[1:4])
print(spam[:3])
print(spam[::2])
print(spam[::-1])
```

```
s
ese
bes
bsdazea
azevz andeseb
```

4.2 Dolžina

Število elementov seznama ali število znakov v nizu dobimo s funkcijo `len(seznam_ali_niz)`.

```
c = [1, 2, 3, "abeceda"]
print(len(c))

d = "terminologija"
print(len(d))
```

```
4
13
```

4.3 Združevanje

Več seznamov / več nizov staknemo s plusom.

```
sez1 = [1, 2, 3]
sez2 = [4, 5, 6]
print(sez1 + sez2)

niz1 = "a"
niz2 = "b"
print(niz1 + niz2)
```

```
[1, 2, 3, 4, 5, 6]
ab
```

Sezname lahko združimo tudi z metodo `extend`. Za zgornji primer: `seznam1.extend(seznam2)`.

4.4 Preverjanje vsebovanja

S ključno besedo `in` preverimo ali seznam vsebuje nek element, kar lahko uporabimo npr. v if stavku. Podobno lahko z `in` preverimo ali niz vsebuje nek znak.

```
print('cat' in ['cat', 'bat', 42, True, 'dog'])
print('c' in 'beseda')
```

```
True
False
```

4.5 Zanke

Po elementih seznama / znakih niza se lahko sprehodimo z zanko for.

```
spam = ['cat', 'bat', 42, True, 'dog']
for element in spam:
    print(element)
```

```
cat
bat
42
True
dog
```

```
spam = "abc"
for element in spam:
    print(element)
```

```
a
b
c
```

Ko v zanki potrebujemo indekse, pride prav spodnji pristop, kjer do elementov seznama dostopamo z indeksi.

Primer: zanima nas razlika med zaporednimi elementi seznama.

```
seznam = [1, 2, 3, 5, 6, 7]
for i in range(len(seznam) - 1):
    print(seznam[i+1] - seznam[i])
```

```
1
1
2
1
1
```

Tretji pristop k sprehodu po elementih seznama / znakih niza pa je funkcija `enumerate` (glej dno te strani za primer).

4.6 Spreminjanje

Vrednost elementa v seznamu lahko spremenimo.

```
spam = ['cat', 'bat', 42, True, 'dog']
spam[1] = 'aardvark'
print(spam)
```

```
['cat', 'aardvark', 42, True, 'dog']
```

V nizu znakov ne moremo tako spreminjati! Uporabimo pa lahko podsezname:

```
s = "abcdef"
index = 3
s = s[:index] + "ž" + s[index + 1:]
print(s)
```

```
abcžef
```

Elemente seznama lahko zbrišemo s ključno besedo `del`.

```
spam = ['cat', 'bat', 42, True, 'dog']
del spam[2]
print(spam)
```

```
['cat', 'bat', True, 'dog']
```

4.7 Dodajanje elementov

Element lahko dodamo na konec seznama z metodo `append`.

```
spam = ['cat', 'bat', 42, True, 'dog']
spam.append(3)
print(spam)
```

```
['cat', 'bat', 42, True, 'dog', 3]
```

Pri nizih ne moremo uporabiti metode append, lahko pa dodamo elemente z operatorjem `+`. Tudi tu lahko uporabimo okrajšavo `+=`.

```
s = "abcd"
s += "e" # oziroma s = s + "e"
print(s)
```

```
abcde
```

4.8 Nekaj uporabnih funkcij

- `len(seznam_ali_niz)` vrne število elementov seznama oz. število znakov v nizu
- `enumerate(seznam)` vrne zaporedje parov, v katerih so na drugem mestu vrednosti iz podanega seznama, na prvem mestu pa so njihovi indeksi. Funkcija vrne poseben tip - da dobimo seznam, moramo ta tip pretvoriti s funkcijo `list()`. V for zanki lahko uporabimo `enumerate` brez `list`.

```
print(list(enumerate(["a", "b", "c"])))
```

```
[(0, 'a'), (1, 'b'), (2, 'c')]
```

```
for indeks, element in enumerate(["a", "b", "c"]):
    print(indeks, element)
```

```
0 a
1 b
2 c
```

- `zip(seznam1, seznam2, ...)` vrne zaporedje naborov istoležnih elementov v podanih seznamih. Funkcija vrne poseben tip - da dobimo seznam, moramo ta tip pretvoriti s funkcijo `list()`. V for zanki lahko uporabimo `zip` brez `list`.

```
print(list(zip('xyz', [10, 20, 30], [4, 5, 6])))
```

```
[('x', 10, 4), ('y', 20, 5), ('z', 30, 6)]
```

```
for x in zip('xyz', [10, 20, 30], [4, 5, 6]):
    print(x)
```

```
('x', 10, 4)
('y', 20, 5)
('z', 30, 6)
```

5 Delo z objekti

Gradivi za to poglavje sta

- https://automatetheboringstuff.com/2e/chapter6/
- poglavje Nizi

Uporabna je tudi dokumentacija za različne tipe https://docs.python.org/3/tutorial/datastructures.html

Metod za sezname in nize je veliko. Spodaj je naštetih nekaj najpogosteje uporabljanih. Celoten seznam je v uradni dokumentaciji: https://docs.python.org/3/library/stdtypes.html#string-methods

Ponavadi lahko z Googlom hitro najdemo metodo, ki jo potrebujemo, če opišemo, kaj želimo narediti (npr. s `python count characters in string` hitro najdemo `count()` in primere uporabe).

5.1 Pretvarjanje med tipi

Spremenljivko `a` lahko pretvorimo v drug tip s funkcijami:

- `int(a)` vrne celo število
- `float(a)` vrne decimalno število
- `str(a)` vrne niz
- `list(a)` vrne seznam
- `tuple(a)` vrne nabor
- `set(a)` vrne množico

5.2 Metode za nize

- `niz.count(znak)` vrne število pojavitev znaka v nizu
- `niz.index(znak)` vrne indeks, na katerem se znak prvič pojavi; če ne obstaja sproži napako
- `niz.replace(niz1, niz2)` vrne niz, kjer so podnizi enaki `niz1` zamenjani z `niz2`
- `niz.lower()` in `niz.upper()` vrne niz, kjer iz malih črk naredi velike ali obratno

- `niz.islower()` in `niz.isupper()` vrne True, če je niz iz samih malih črk oz. velikih črk
- `niz.strip()` vrne niz, kjer z leve in desne strani odstrani "whitespace characters" (presledki, tab, `\n`). Lahko podamo neobvezni argument, s katerim določimo, katere znake naj odstrani z leve in desne. Obstajata tudi metodi `.rstrip()` in `.lstrip()`, ki odstranjujeta le z leve in desne.

```
print('    Hello, World    \n'.strip())
```

```
Hello, World
```

- `"locilo".join(seznam)` združi elemente seznama v niz in postavi `locilo` med posamezne elemente

```
print('ABC'.join(['Moje', 'ime', 'je', 'Rok']))
```

```
MojeABCimeABCjeABCRok
```

- `niz.split(locilo)` vrne seznam, kjer so elementi posamezni deli niza, ki jih ločuje `locilo`. Privzeta vrednost za `locilo` je presledek.

```
print("Moje ime je Rok.".split())
```

```
['Moje', 'ime', 'je', 'Rok.']
```

5.3 Metode za sezname

- `sez.append(element)` doda element na konec seznama
- `sez.extend(sez2)` na konec seznama `sez` pristavi seznam `sez2`, na kratko `sez += sez2`
- `sez.insert(i, x)` na mesto z indeksom `i` vstavi element `x`
- `sez.remove(x)` iz seznama odstrani prvo pojavitev elementa `x`
- `sez.pop(i)` odstrani element na indeksu `i` in ga vrne; če `i` ne podamo je to zadnji element
- `sez.index(x)` vrne prvi indeks, na katerem se nahaja vrednost `x`
- `sez.count(x)` vrne število pojavitev `x` v seznamu
- `sez.sort()` razvrsti seznam števil po velikosti / nizov po abecedi

5.4 Sortiranje seznamov

Za sortiranje lahko uporabimo funkcijo `sorted(seznam)`, ki vrne sortiran seznam ali pa metodo `seznam.sort()`, ki spremeni originalni seznam, tako, da je sortiran. Metoda `.sort()` ne vrne nič. Obema funkcijama lahko podamo neobvezni argument `reversed=True`, ki seznam sortira v nasprotnem vrstnem redu.

```
seznam = [9, 1, 4, 5, 6, 4] #<<
print("prej:   ", seznam)
sortiran_seznam = sorted(seznam) #<<
print("potem:  ", seznam)
print("vrnjeno:", sortiran_seznam)
```

```
prej:    [9, 1, 4, 5, 6, 4]
potem:   [9, 1, 4, 5, 6, 4]
vrnjeno: [1, 4, 4, 5, 6, 9]
```

```
seznam = [9, 1, 4, 5, 6, 4] #<<
print("prej:   ", seznam)
a = seznam.sort() #<<
print("potem:  ", seznam)
print("vrnjeno:", a)
```

```
prej:    [9, 1, 4, 5, 6, 4]
potem:   [1, 4, 4, 5, 6, 9]
vrnjeno: None
```

6 Slovarji

Gradivi za to poglavje sta

- https://automatetheboringstuff.com/2e/chapter5/
- poglavje Slovar

6.1 Dostopanje do vrednosti

Do vrednosti dostopamo podobno kot pri seznamih, le da namesto indeksov uporabimo ključe.

```
s = {'a': 6, 'b': 'test', 123: True}
print(s['b'])
```

```
test
```

Če ključ ne obstaja pride do napake. Temu se lahko izognemo z metodo `get`

```
s = {'a': 6, 'b': 'test', 123: True}
print(s.get('a'))
print(s.get('ž', 100000))
```

```
6
100000
```

6.2 Spreminjanje in dodajanje

```
s = {'a': 6, 'b': 'test', 123: True}
s['a'] = 10
s['abcd'] = False
print(s)
```

```
{'a': 10, 'b': 'test', 123: True, 'abcd': False}
```

Nov par ključ-vrednost dodamo tako kot piše v tretji vrstici zgornjega primera. Pri dodajanju ni pomembno, če ključ še ne obstaja v slovarju, ampak se neobstoječi ključ doda, vrednost pa se nastavi na vrednost desno od enačaja.

6.3 Brisanje

```
s = {'a': 6, 'b': 'test', 123: True}
del s['a']
print(s)
```

```
{'b': 'test', 123: True}
```

6.4 Metode

- `s.get(kljuc, privzeta_vrednost)` vrne vrednost, ki ustreza ključu `kljuc`, če ključ ne obstaja v slovarju vrne None; lahko podamo še neobvezni parameter `privzeta_vrednost`, ki jo vrne, če ključ ne obstaja; glej primer na vrhu strani
- `s.pop(kljuc)` iz slovarja odstrani par s ključem `kljuc` in vrne vrednost
- `s.update(s2)` k slovarju `s` doda pare slovarja `s2`
- `s.values()` za uporabo v zankah; vrne vrednosti v slovarju; s funkcijo `list` pretvorimo v seznam
- `s.keys()` za uporabo v zankah; vrne ključe v slovarju; s funkcijo `list` pretvorimo v seznam
- `s.items()` za uporabo v zankah; vrne nabore ključev in vrednosti; s funkcijo `list` pretvorimo v seznam

6.5 Preverjanje vsebovanja

Kot pri seznamih in nizih uporabimo ključno besedo `in`, da preverimo ali se nek ključ/vrednost/par nahaja v slovarju.

```
s = {'a': 6, 'b': 'test', 123: True}
print('test' in s.values())
print('b' in s.keys())
print(('a', 6) in s.items())
```

```
True
True
True
```

6.6 Zanke

Zanka po vrednostih v slovarju.

```
s = {'a': 6, 'b': 'test', 123: True}
for v in s.values():
    print(v)
```

```
6
test
True
```

Zanka po ključih v slovarju.

```
s = {'a': 6, 'b': 'test', 123: True}
for k in s.keys(): # lahko tudi "for k in s:"
    print(k)
```

```
a
b
123
```

Zanka po parih v slovarju.

```
s = {'a': 6, 'b': 'test', 123: True}
for kljuc, vrednost in s.items():
    print(kljuc, '->', vrednost)
```

```
a -> 6
b -> test
123 -> True
```

7 Datoteke

Gradivo za to poglavje je https://automatetheboringstuff.com/2e/chapter9/

Če vas zanima več, tudi https://automatetheboringstuff.com/2e/chapter10/

Primer branja iz datoteke z vaj: temp.txt (Vir: ARSO), temp.csv (Vir: ARSO), temp.py

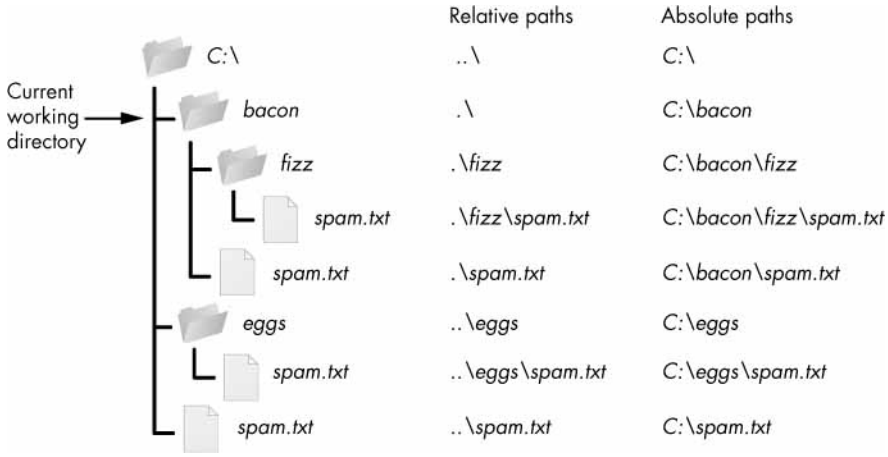
7.1 Datotečni sistem

Datoteke so shranjene na različnih nosilcih (npr. trdi disk, SSD, DVD, USB ključ, ...). Prostor, ki je na voljo na nosilcu lahko razdelimo na več ločenih delov, ki jim rečemo particije. Primer: trdi disk z 1000 GB bi lahko razdelili na dve particiji `C` z 100 GB in `D` z 900 GB. Vsaka particija na nosilcih, ki so priklopljeni na računalnik, dobi svojo črko (to velja za Windows, drugje je drugače), npr. USB ključi so pogosto pod `E` ali `F`.

Na računalniku datoteke organiziramo v mape, ki so lahko gnezdene. Na vrhu imamo korensko mapo (root folder). Na Linux in macOS je to `/` na Windowsu pa `C:\`, kjer je `C` ime particije.

7.1.1 Absolutna in relativna pot

Vsaki datoteki ustreza ena absolutna pot. To je "naslov", pod katero jo lahko najdemo. Primer: `C:\bacon\fizz\spam.txt`. Pot vsebuje vse mape, v katerih se datoteka nahaja, ločene z `\` (na Windowsu; na Linux in macOS je ločilo `/`), ime datoteke, piko in končnico datoteke, ki določa njen tip.



Slika 7.1: Vir: https://automatetheboringstuff.com/2e/chapter9/ (Al Sweigart, CC BY-NC-SA 3.0)

Relativna pot do datoteke je pot glede na neko drugo mapo. Za zgornji primer: glede na mapo `bacon` je relativna pot do datoteke `.\fizz\spam.txt`.

> Pika pomeni trenutno mapo.

Če bi bila trenutna mapa `eggs`, bi bila relativna pot do prejšnje datoteke glede na `eggs` enaka `..\bacon\fizz\spam.txt`.

> Dve piki pomenita eno mapo višje v hierarhiji (parent folder) glede na trenutno mapo.

Če bi želeli iti dve mapi višje bi uporabili `..\..`, npr. iz mape `fizz` v mapo `eggs` pridemo z `..\..\eggs`, itd.

Za podrobnejši razlago in več primerov glej gradivo: https://automatetheboringstuff.com/2e/chapter9/

7.1.2 *Delo z ukaznim pozivom

Podobno kot v Raziskovalcu (File Explorer) se tudi v ukaznem pozivu (Terminal) v nekem trenutku nahajamo v neki mapi (ang. Current working directory ali CWD). Ta mapa je vedno napisana na začetku vrstice. V ukaznem pozivu najprej napišemo ukaz nato parametre, ki jih želimo podati, ločene s presledki. Ukaz izvedemo s tipko Enter.

V neko mapo se lahko premaknemo z ukazom `cd`, ki mu kot argument podamo pot (relativno ali absolutno do mape, v katero se želimo premakniti.

Ukaz `dir` izpiše vse datoteke in mape, ki se nahajajo v trenutni mapi.

Glej tudi: https://ucilnica.fmf.uni-lj.si/mod/page/view.php?id=2505

7.1.3 *Mape in datoteke v Pythonu

Za delo z datotečnim sistemom je na voljo modul `os`. Posamezne funkcije, njihove parametre in uporabo lahko poiščete v uradni dokumentaciji ali drugod na spletu. Spodaj parametri funkcij niso napisani!

Nekaj najbolj uporabnih:

- `os.getcwd()` vrne trenutno mapo (CWD)
- `os.chdir()` nastavi trenutno mapo na podano pot
- `os.listdir()` vrne seznam poti do datotek in map, ki se nahajajo v mapi, do katere vodi pot
- `os.mkdir()` ustvari novo mapo, ki se nahaja na podani poti
- `os.rename()` preimenuje mapo, prvi parameter je pot mape, drugi pa nova pot (z novim imenom)
- `os.remove()` izbriše datoteko, ki se nahaja na podani poti
- `os.rmdir()` izbriše prazno mapo, ki se nahaja na podani poti

Za delo s potmi je na voljo modul `os.path`, kjer so pogosto uporabljane funkcije:

- `os.path.exists()` vrne True, če podana pot obstaja
- `os.path.join()` stakne dve poti v eno, pri čemer ustrezno poskrbi za prava ločila glede na OS
- `os.path.abspath()` vrne absolutno pot, ki ustreza podani relativni poti (glede na trenutno mapo)
- `os.path.relpath()` vrne relativno pot, ki ustreza podani absolutni poti (glede na trenutno mapo)
- `os.path.isfile()` vrne True, če pot vodi do datoteke
- `os.path.isdir()` vrne True, če pot vodi do mape

7.2 Pisanje

Datoteko odpremo v načinu za pisanje `mode="w"` in uporabimo metodo `write()`, ki zapiše niz v datoteko. Znak `\n` pomeni novo vrstico. Če želimo zapisati znak `\` moramo v Pythonu napisati `\\`. Več o uporabi posebnih znakov v Pythonu: https://www.w3schools.com/python/gloss_python_escape_characters.asp

```
pot_do_datoteke = "datoteka.txt"
with open(pot_do_datoteke, mode="w", encoding="utf-8") as dat:
    dat.write("To je ")
    dat.write("en stavek.\nTo je drugi.")
```

```
datoteka.txt
To je en stavek.
To je drugi.
```

Namesto `dat.write("niz")` se lahko uporablja tudi `print("niz", file=dat)`, kjer odprto datoteko podamo kot parameter.

7.3 Branje

7.3.1 read()

Datoteko odpremo v načinu za branje `mode="r"` in uporabimo metodo `read()`, ki vrne celotno vsebino datoteke naenkrat v obliki niza.

```
with open("datoteka.txt", mode="r", encoding="utf-8") as datoteka:
    vsebina = datoteka.read()
print(vsebina)
```

```
To je en stavek.
To je drugi.
```

Uporaba argumenta `mode` je opisana na dnu strani. Klicu `open` lahko podamo tudi neobvezni argument `encoding`, ki poda kodno tabelo, v kateri je napisana datoteka. Privzeta vrednost tega argumenta je na šolskih (in najverjetneje tudi vaših) Windows računalnikih `windows-1252`, kar je nekoliko zastarel standard. Zato je dobra praksa uporaba parametra `encoding="utf-8"`, s čimer uporabimo Unicode, ki se danes uporablja skoraj povsod. Na macOS in Linux je vrednost `utf-8` že privzeta.

7.3.2 readlines()

Z metodo `readlines()` dobimo seznam, v katerem so posamezne vrstice iz datoteke.

```
with open("datoteka.txt", mode="r", encoding="utf-8") as datoteka:
    vrstice = datoteka.readlines()
print(vrstice)
```

```
['To je en stavek.\n', 'To je drugi.']
```

7.3.3 for zanka

Po vrsticah datoteke lahko gremo s for zanko.

```
vrstice = []
with open("datoteka.txt", mode="r", encoding="utf-8") as datoteka:
    for line in datoteka:
        vrstice.append(line)
print(vrstice)
```

```
['To je en stavek.\n', 'To je drugi.']
```

7.4 Mode

je neobvezni argument funkcije `open()`. Privzeta vrednost je `mode="rt"`. Zato nam v zgornjih primerih ni bilo treba pisati `t` (je že privzet poleg druge črke, ki jo podamo (`r` ali `w`)). S posameznimi črkami povemo, kaj želimo z datoteko početi.

oznaka	opis	opomba
r	branje	če datoteka ne obstaja, sproži napako
w	pisanje	če datoteka ne obstaja, ustvari novo; če obstaja, izbriše prejšnjo vsebino datoteke
a	append	če datoteka ne obstaja, ustvari novo; če obstaja, *NE* izbriše prejšnje vsebine
x	ustvari datoteko, pisanje	če datoteka že obstaja, sproži napako
+	pisanje in branje	
t	za delo s tekstovnimi datotekami	npr. .txt, .csv, .tex, .html, .py
b	za delo s binarnimi datotekami	npr. slike

Nekaj lastnosti je zbranih v spodnji tabeli:

lastnost \ kombinacija črk	r	r+	x	x+	w	w+	a	a+
branje	x	x		x		x		x
pisanje		x	x	x	x	x	x	x

lastnost črk \ kombinacija	r	r+	x	x+	w	w+	a	a+
datoteka mora obstajati	x	x						
datoteka ne sme obstajati			x	x				
zbriše prejšnjo vsebino datoteke					x	x		
pisanje na konec datoteke							x	x

K zgornjim kombinacijam lahko dodamo še `t` ali `b`.

8 Numpy

Gradivi za to poglavje sta:

- https://numpy.org/doc/stable/user/absolute_beginners.html
- https://numpy.org/doc/stable/user/quickstart.html

Povezave in osnovna navodila za namestitev knjižnice numpy na domači računalnik so v Poglavje 1.3 .

> Primeri z vaj
>
> https://slides.com/rokkuk/numpy-1
> https://slides.com/rokkuk/numpy-2

Vizualna reprecentacija operacij z Numpy seznami: http://jalammar.github.io/visual-numpy/

8.1 Ustvarjanje tabel

Navaden seznam pretvorimo v Numpy seznam s funkcijo `np.array(seznam)`.

```
import numpy as np

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(a)
print(a.shape)
print(a.ndim)
print(a.size)
```

```
[[1 2 3 4]
 [5 6 7 8]]
(2, 4)
2
8
```

Sezname lahko združimo z `np.concatenate(sez1, sez2)`, ki vrne staknjen seznam. To je počasna operacija, zato je raje ne uporabljajmo.

Če je le mogoče vnaprej ustvarimo nov seznam znane dolžine z eno od spodnjih metod:

- `np.zeros(oblika)` parameter `oblika` je nabor celih števil, ki predstavlja obliko tabele npr. (3, 2) ustvari tabelo z tremi vrsticami in dvema stolpcema
- `np.ones(oblika)`
- `np.linspace(zacetek, konec, num)` vrne enakomerno razporejena števila na intervalu
- `np.arange(zacetek, konec, korak)` vrne števila ločena za korak (če ni podan, je 1)
- `np.fromfunction(ime_funkcije, oblika)`
- `seznam.reshape(oblika)` spremeni obliko seznama

```
print(np.zeros((3, 2))) # tri vrstice, dva stolpca
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
```

8.2 Rezine

Do elementov tabel dostopamo podobno kot pri običajnih seznamih (`zacetek:konec:korak`), le da to naredimo za vsako dimenzijo posebej ločeno z vejicami.

```
tabela = np.arange(0, 49).reshape(7, 7)
print(tabela)

razrezano = tabela[0:4,2:7:2] # prve štiri vrstice, vsak drugi stolpec od tretjega do sedmega
print(razrezano)
```

```
[[ 0  1  2  3  4  5  6]
 [ 7  8  9 10 11 12 13]
 [14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34]
 [35 36 37 38 39 40 41]
 [42 43 44 45 46 47 48]]
[[ 2  4  6]
 [ 9 11 13]
 [16 18 20]
 [23 25 27]]
```

Če želimo vse elemente v neki dimenziji napišemo `:`. Tako lahko dobimo posamezne stolpce.

```
print(tabela[:,3]) # četrti stolpec
```

```
[ 3 10 17 24 31 38 45]
```

Ko imamo enkrat izbrane želene vrstice in stolpce, lahko te vrednosti shranimo v spremenljivko (kot je to zgoraj pri `razrezano`). Lahko pa pa na ta izbrana mesta v tabeli shranimo neke druge vrednosti. Shranjujemo lahko tudi cele tabele naenkrat:

```
minitabela = np.arange(100, 112).reshape(4,3)
print(minitabela)

tabela[0:4,2:7:2] = minitabela # na izbrana mesta shranimo vrednosti iz minitabela
print(tabela)
```

```
[[100 101 102]
 [103 104 105]
 [106 107 108]
 [109 110 111]]
[[  0   1 100   3 101   5 102]
 [  7   8 103  10 104  12 105]
 [ 14  15 106  17 107  19 108]
 [ 21  22 109  24 110  26 111]
 [ 28  29  30  31  32  33  34]
 [ 35  36  37  38  39  40  41]
 [ 42  43  44  45  46  47  48]]
```

8.3 Uporabne funkcije

Glej predvsem uradno dokumentacijo: https://numpy.org/doc/stable/reference/routines.sort.html
Vsaka funkcija ima opis parametrov in zelo nazorne primere uporabe.

Pri mnogih funkcijah lahko podamo neobvezni parameter `axis=x`, kjer je `x` številka osi, po kateri želimo operacijo izvesti (0, 1, 2, ...).

> **Pozor**
>
> Vrednost True se obnaša kot 1 in False se obnaša kot 0 ter obratno.

- `np.any(tabela)` vrne True, če je vsaj en element True
- `np.all(tabela)` vrne True, če so vsi elementi True
- `np.nonzero(tabela)` vrne indekse neničelnih elementov v vsaki dimenziji posebej (koordinate teh elementov)
- `np.flatten(tabela)` vrne "flat" obliko tabele (enodimenzionalni seznam zaporednih elementov)
- `np.flatnonzero(tabela)` vrne indekse neničelnih elementov v "flat" obliki tabele (zaporedni indeks)
- `np.where(pogoj, x, y)` vrne elemente iz `x`, kjer je pogoj izpolnjen, sicer vrne ustrezni element iz `y`; pogoj se ovrednoti za vsak element posebej; glej primere v dokumentaciji!
- `tabela.T` vrne transponiramo tabelo (to pomeni, da so elementi zrcaljeni preko diagonale); deluje tudi za nekvadratne tabele

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
print(a.T)
```

```
[[1 2 3]
 [4 5 6]]
[[1 4]
 [2 5]
 [3 6]]
```

8.4 Matematika

Dokumentacija: https://numpy.org/doc/stable/reference/routines.math.html

- `np.abs(tabela)`
- `np.sum(tabela)`
- `np.cumsum(tabela)`
- `np.prod(tabela)`
- `np.log(tabela)`
- `np.exp(tabela)`
- `np.sin(tabela)`
- itd.

9 Drugo

9.1 Nabori

Nabori so urejeni in nespremenljivi. Definira se jih z običajnimi oklepaji `()`. Do elementov dostopamo z indeksi od 0 naprej, kot pri seznamih. Naboru ne moremo dodati novih elementov, ko je enkrat ustvarjen. Glej tudi: https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences

Vrednosti v naborih lahko "odpakiramo" (*ang.* unpacking) nazaj v spremenljivke.

```
sadje = ("jabolko", "banana", "češnja")
zeleno, rumeno, rdece = sadje
print(zeleno)
print(rumeno)
print(rdece)
```

```
jabolko
banana
češnja
```

Nabor lahko odpakiramo v manj spremenljivk, kot je elementov nabora, če eni od spremenljivk pred ime dodamo `*`. V to spremenljivko se bo shranil seznam presežnih elementov. Glej npr. https://www.w3schools.com/python/python_tuples_unpack.asp

9.2 Množice

Množice niso urejene in so nespremenljive. Definira se jih z zavitimi oklepaji. Prazno množico lahko dobimo tako, da pokličemo funkcijo `set()`. Vrednosti v množici so unikatne (ne moremo dodati dveh enakih). Do elementov lahko dostopamo z zanko. Nove elemente lahko dodamo z `mnozica.add(element)`. Glej tudi: https://docs.python.org/3/tutorial/datastructures.html#sets

9.3 *Rekurzija

Dober članek o uporabi rekurzije za izračun Fibbonaccijevega zaporedja v Pythonu. Poglavje *Using Recursion and a Python Class* lahko preskočite. https://realpython.com/fibonacci-sequence-python/

9.4 *Uvažanje modulov

Delo z moduli lepo opisuje dokumentacija (predvsem prva polovica strani).

9.5 *Merjenje časa izvajanja programa

Glej https://docs.python.org/3/library/timeit.html

9.6 *Izpeljani seznami/slovarji/množice

Če želimo ustvariti seznam s kvadrati celih števil, lahko to naredimo z zanko:

```
kvadrati = []
for i in range(10):
    kvadrati.append(i * i)
print(kvadrati)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Na kratko lahko tak seznam ustvarimo z izpeljanim seznamom:

```
kvadrati2 = [i * i for i in range(10)]
print(kvadrati2)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Dodamo lahko tudi pogoj:

```
poved = 'the rocket came back from mars'
samoglasniki_poved = [i for i in poved if i in 'aeiou']
print(samoglasniki_poved)
```

```
['e', 'o', 'e', 'a', 'e', 'a', 'o', 'a']
```

Podoben pristop lahko uporabimo tudi za slovarje in množice.

```
kvadrati3 = {i: i * i for i in range(10)}
print(kvadrati3)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

```
poved = "life, uh, finds a way"
samoglasniki_poved = {i for i in poved if i in 'aeiou'}
print(samoglasniki_poved)
```

```
{'e', 'i', 'u', 'a'}
```

Part I

Dodatno

10 Kako deluje računalnik?

Pri programiranju je pogosto koristno vedeti, kako deluje računalnik. To je zelo obsežna tema, o kateri bi se lahko učili leta. Kljub temu lahko razumemo osnovne ideje delovanja računalnikov že v nekaj urah. Če vas tema bolj poglobljeno zanima, lahko na spletu najdete veliko specializiranih gradiv in univerzitetnih učbenikov.

Spodaj sem skušal zbrati čim bolj jedrnata gradiva na spletu, ki poljudno in dobro razložijo nekatere osnovne ideje. Nekatere teme so dobro razložene v poglavju Sistemi v učbeniku https://lusy.fri.uni-lj.si/ucbenik/book/1301/index.html.

10.1 Crash Course Computer Science

Obširen tečaj osnov računalništva. Vzame nekaj ur, priporočam predvsem prvih 21 videov https://www.youtube.com/watch?v=tpIctyqH29Q&list=PLH2l6uzC4UEW0s7-KewFLBC1D0l6XRfye&index=1.

Če nimate časa za vse to, si lahko pogledate nekaj kratkih videov spodaj (po 10-20 min). Niso mišljeni kot študijsko gradivo – bolj kot zanimivosti.

10.2 Posamezne teme

Spodaj je navedenih nekaj tem o delovanju računalnika. Predlagam ogled v takem vrstnem redu:

1. **Computer basics - What is inside a computer?** Komponente računalnika. Osnovno - lahko preskočite. https://www.youtube.com/watch?v=HB4I2CgkcCo
2. **Exploring How Computers Work - Sebastian Lague** Izvrstna demonstracija tega, kako elektronska vezja omogočajo računanje. *Če si boste pogledali le eno stvar, si poglejte tole.* https://www.youtube.com/watch?v=QZwneRb-zqA
3. **What's Your Computer Actually Doing? - Tom Scott** Poenostavljen opis kako računalniški procesor izvaja naše programe. https://www.youtube.com/watch?v=Z5JC9Ve1sfI

4. **How do computers read code?** Kako naš program pove računalniku kaj mora narediti? *Če si boste pogledali le dve stvari, si poglejte še tole.* https://www.youtube.com/watch?v=QXjU9qTsYCc

5. **Learn how computers add numbers and build a 4 bit adder circuit - Ben Eater** Razlaga vezja za seštevanje in demonstracija, kako ga sestaviti. https://www.youtube.com/watch?v=wvJc9CZcvBc

6. **How Do Computers Remember? - Sebastian Lague** Demonstracija elektronskih vezij, ki lahko shranijo vrednosti. https://www.youtube.com/watch?v=I0-izyq6q5s&t=613s

7. **Visualizing Binary Data with 7-segment displays - Sebastian Lague** Demonstracija, kako lahko z vezji prikažemo binarna števila na zaslonu. https://www.youtube.com/watch?v=hEDQpqhY2MA

8. **Physics of Computer Chips - Dr. Phil Moriarty** Intervju z nanofizikom, ki opiše, kako so narejene komponente procesorja. https://www.youtube.com/watch?v=xkLAhU74f3s

9. **Izdelava čipov** Tole je sicer PR video, a dobro pokaže kompleksnost in proces izdelave čipov. https://www.youtube.com/watch?v=bor0qLifjz4

10. **Why do computers use 1's and 0's?** Kaj je binarni številski sistem in kako se uporablja za reprecentacijo števil in črk. https://www.youtube.com/watch?v=Xpk67YzOn5w

11 LaTeX

LaTeX je program za urejanje besedil. Odlikuje se v izdelavi matematičnih dokumentov, zato je standard za znanstvene članke in poročila v matematiki in fiziki. Omogoča tudi avtomatsko generiranje bibliografij.

V slovenščini je na voljo priročnik Ne najkrajši uvod v LaTeX, za začetek pa je morda še primernejši Praktičen uvod v LaTeX.

Dobra referenca je Overleaf, ki omogoča tudi ustvarjanje LaTeX dokumentov preko spletnega vmesnika. Preko spleta je urejanje nekoliko nerodno in počasno, zato priporočam, da si namestite enega od urejevalnikov (glej spodaj).

LaTeX je potrebno najprej namestiti. Na Windowsu je verjetno najbolje namestiti MiKTeX, na macOS pa MacTeX.

LaTeX dokumente lahko ustvarjate v *Visual Studio Code*, če namestite Extension z imenom Latex Workshop. Na voljo so tudi drugi specializirani urejevalniki, kot je TeXstudio (meni najljubši), LyX (omogoča način urejanja podoben Wordu) ali Overleaf (omogoča hkratno delo več oseb preko spleta).

Zelo uporabno orodje je Mathpix, ki omogoča pretvorbo enačb napisanih na roko ali v PDF datotekah v LaTeX ukaze.

Priročna je tudi spletna stran Table generator, ki omogoča generiranje kode za LaTeX tabele iz Excelovih ali .csv datotek.

12 Risanje grafov

Za risanje grafov s Pythonom je najbolj priljubljena knjižnica matplotlib. Za začetek je uporaben Tutorial, dobra referenca pa so Primeri.

S knjižnico matplotlib je mogoče grafe shraniti v različnih formatih. Pred risanjem grafov je dobro nastaviti velikost slike s `plt.figure(figsize=(sirina, visina))`, kjer sta višina in širina v inčih.

Če pišemo v LaTeXu, imamo dve glavni možnosti:

1. Sliko shranimo v vektorskem formatu (npr. `.pgf`) To lahko naredimo s klicem funkcije `plt.savefig(pot.pgf)`. Sliko nato dodamo v LaTeX dokument z `\input{pot.pgf}`, znotraj okolja `figure` pred tem pa pri vrhu datoteke uvozimo `\usepackage{pgf}`. Prednost vektorske slike je, da slika ohrani ostrino ne glede na to, koliko povečana je, in da so fonti številk in oznak na sliki enaki kot v LaTeXu ter prave velikosti. Da se tudi nastaviti, da je širina slike npr. ravno polovico širine dokumenta. Vse skupaj je opisano tu: https://timodenk.com/blog/exporting-matplotlib-plots-to-latex Za veliko grafov z veliko točkami (>10000) je bolje uporabiti rastrski format (npr. `.png`), sicer lahko stavljenje dokumenta z LaTeXom traja več minut.
2. Sliko shranimo v rastrskem formatu (npr. `.png`). To lahko naredimo s klicem funkcije `plt.savefig(pot.png, dpi=300)`. Parameter `dpi` (dots per inch) dolča resolucijo slike. Ponavadi zadošča 300 ali manj. Sliko nato dodamo v LaTeX dokument z `\includegraphics{pot.png}` znotraj okolja `figure`. Privzeto fonti ne bodo taki kot v LaTeXu, kar ni najlepše. Prave fonte lahko nastavimo z

```
plt.rcParams.update({"text.usetex": True})
```

13 Sortiranje

Za osnove sortiranja glej Delo z objekti

Bolj podrobno je sortiranje opisano v dokumentaciji: https://docs.python.org/3/howto/sorting.html

Funkciji za sortiranje lahko podamo tudi neobvezni argument `key=ime_funkcije`, s katerim podamo ime druge funkcije, ki sprejme elemente seznama in vrne tiste elemente, po katerih želimo seznam sortirati. Primer sortiranja po tretjem elementu nabora:

```
def sortirna_funkcija(element_seznama):
    return element_seznama[2]

student_tuples = [
    ('john', 'A', 15),
    ('jane', 'B', 12),
    ('dave', 'B', 10),
]
print(sorted(student_tuples, key=sortirna_funkcija))
```

```
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

Sortiramo lahko tudi po več komponentah nabora, če v funkciji vrnemo nabor komponent, po katerih želimo sortirati. npr. funkcija

```
def sortirna_funkcija(element_seznama):
    return  (element_seznama[1], element_seznama[2])
```

bi seznam najprej sortirala po drugem elementu nabora, nato pa še po tretjem (tisti elementi seznama, ki imajo enak drugi element nabora, bi bili sortirani še po tretjem).