UPORABA IKT V NARAVOSLOVJU IN TEHNIKI

Predavanje 12

Python: seznam, funkcije in datoteke

doc.dr. Mira Trebar

Vsebina

- Python
  - Seznam (list, range, break)
  - Funkcije
    - Definicija
    - Spremenljivke
    - Vračanje vrednosti
  - Datoteke
    - Branje
    - Pisanje

Seznam - uvod

- Kaj je seznam?
  - Je urejena zbirka objektov, kjer lahko nastopa karkoli.
  - Podoben je nizu, le da v nizu nastopajo samo črke,
  - Seznam dobimo tako, da objekte zapremo v oglate oklepaje.

```
>>> seznam = [1, 2, 3, 4, 10, -20]
>>> osebaStarost= ['Janez', 20, 'Mojca', 19.5]
>>> prazenSeznam = []
```

- Delo s seznami – elementi

```
 >>> osebaStarost[0]
'janez'
>>> osebaStarost[1]
20
>>> osebaStarost[-1]
19.5
```

Seznam seznamov

- Element seznama je seznam
  - ukazna vrstica
  - programsko
- Delo s seznami seznamov

```
>>> osebaLeto = [['J', 20], ['M', 19.5]]
[['J', 20], ['M', 19.5']]
>>> osebaLeto = [
... ['Janez', 20],
... ['Mojca', 19.5]
... ]
[['Janez', 20], ['Mojca', 19]]
koordinate = [[0, 0, 0], [0, 0, 1],
              [0, 1, 0], [0, 1, 1],
              [1, 0, 0], [1, 0, 1],
              [1, 1, 0], [1, 1, 1]]

>>> osebaLeto[1]
['Mojca', 19.5]
>>> osebaLeto[1][0]
'Mojca'
>>> osebaLeto[1][0][-1]
'a'
```

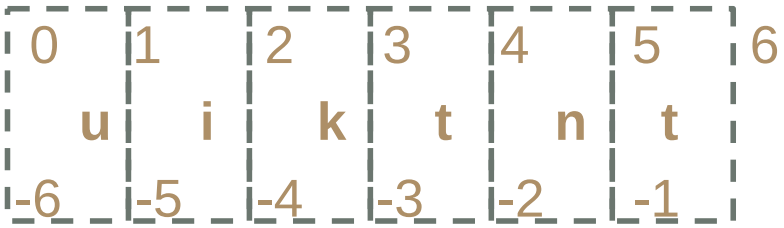
Seznami in nizi

- Delo s seznami in nizi
  - elementi so označeni od 0 dalje
  - izluščimo lahko samo del
    - [od: do]
    - zadnji element ni vključen!!!
  - ugotovimo njegovo dolžino - **len()**

```
>>> seznam = [1, 2, 3, 4, 10, -20]
>>> seznam[1:3]
[2, 3]
>>> len(seznam)
6
>>> niz = 'uiktnt'
>>> niz[0:4]
'uikt'
>>> niz[1:4]
'ikt'
>>> niz[2:]
'ktnt'
>>> niz[:4]
'uikt'
>>> seznam[10]
Traceback (most recent call last):
  File "<interactive input>", line 1, in
<module>
IndexError: list index out of range
```

Seznam - označevanje

- izpisujemo znake od do



  - kako izluščiti znaka 'kt'?
    - številke izbiramo tako, da so želeni znaki med njimi

```
>>> niz = 'uiktnt'
>>> niz[2:4]
'kt'
>>> niz[2:-2]
'kt'
>>> niz[-4:4]
'kt'
>>> niz[-4:-2]

>>> niz1=niz[0:3]+niz[3:5]
>>> niz1
'uiktn'
```

    - lepljenje elementov v seznamu

Seznam – list, range

- Malo daljši koraki
  - dodamo še tretjo številko

    [od:do:?]

    ? =2 – vsak drugi element

  - objekti **`range`** -spremeniti jih moramo v sezname `- list`

  - seštevanje (lepljenje) seznamov

```
>>> stevke ='0123456789'
>>> stevke
'0123456789'
>>> stevke[0:11:2]
'02468'
>>> stevke[1:11:2]
'13579'
>>> stevke[::2]
'02468'

>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(9, -1, -1))
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

>>> [1, 2, 3] + [3, 4, 5]
[1, 2, 3, 3, 4, 5]
```

prg14.py

Seznami in zanka for (izpis)

- Izbira elementov iz seznama - z zanko for

  for x in [ ]:

    stavki

```
>>> for x in [5, 3, -1]:
... print(x)
...
5
3
-1
```

```
imena = ['Mojca', 'Janez', 'Karmen', 'Jaka']
for ime in imena:
   if ime[-1] == 'a':
       print(ime, 'kuha kosilo.')
   else:
       print(ime, 'bere revijo.')

Mojca kuha kosilo.
Janez bere revijo.
Karmen bere revijo.
Jaka kuha kosilo.
```

prg16.py

Programi (1)

- Še nekaj uporabnih programčkov za elemente v seznamu
  - vsota in povprečje elementov v seznamu

```
meritve = [1.1, 2.0, 3.4, 4.7, 5.3]

vsota = 0
for x in meritve:
    vsota = vsota + x
povprecje = vsota / len(meritve)

print('Vsota:', vsota)
print('Povprečje:', povprecje)
```

prg17.py

Programi (2)

- Najmanjši in največji element v seznamu

```
meritve = [1.1, 2.0, 3.4, 4.7, 5.3]

xMin = meritve[0]
xMax = meritve[0]
for x in meritve:
    if x < xMin:
        xMin = x
    if x > xMax:
        xMax = x
print('Min:', xMin)
print('Max:', xMax)
```

prg22a.py

Programi (3)

- Še enkrat poglejmo imena s črko c. Če bi nas zanimalo samo, ali obstaja ime s črko c ali ne, bi napisali takole

```
imena = ['Mojca', 'Janez', 'Cene', 'Jana']

imeImaC = False
for ime in imena:
    print(ime)
    if ('c' in ime) or ('C' in ime):
        imeImaC = True

if imeImaC == True:
    print('Vsaj eno ime ima črko "c".')
else:
    print('V nobenem imenu ni črke "c".')
```

prg22b.py

Programi (4)

- No, takoj ko najdemo vsaj eno ime, ki ima "c" lahko zaključimo. To storimo z ukazom break

```
imena = ['Mojca', 'Janez', 'Cene', 'Jana']

imeImaC = False
for ime in imena:
    print(ime)
    if ('c' in ime) or ('C' in ime):
        imeImaC = True
        break

if imeImaC == True:
    print('Vsaj eno ime ima črko "c".')
else:
    print('V nobenem imenu ni črke "c".')
```

Zanka v zanki - poštevanka

- kaj naredi `end=''` v stavku **`print`**?

```
meja = 10
for i in range(1, meja + 1):
    for j in range(1, meja + 1):
        print(i*j, end=' ')
    print()
```

- še lepše

```
meja = 10
for i in range(1, meja + 1):
    for j in range(1, meja + 1):
        niz = str(i * j)
        presledki = 3 – len(niz)
        print(' ' * presledki + niz, end=' ')
    print()
```

prg23a.py, prg23b.py

Zanke - Urejanje po velikosti (1)

- Pomagajmo si z mehurčki (ang. bubblesort)
  - primerjamo sosednja elementa v seznamu
  - manjši element je lažji (tako kot mehurček), zato ga premaknemo nad večjega (težjega)
  - postopek ponovimo za vse pare

  - Primer:
    - http://web.engr.oregonstate.edu/~minoura/cs261/javaProgs/sort/BubbleSort.html
    - BubbleSort

Zanke - Urejanje po velikosti (2)

- potrebujemo pomožno spremenljivko

```
TAKOLE NE GRE
>>> x = 5
>>> y = 3
>>> [x, y]
[5, 3]
>>> x = y
>>> y = x
>>> [x, y]
[3, 3]
```

```
TAKOLE PA GRE
>>> x = 5
>>> y = 3
>>> [x, y]
[5, 3]
>>> pomozna = x
>>> x = y
>>> y = pomozna
>>> [x, y]
[3, 5]
```

Zanke - Urejanje po velikosti (3)

- Spomnimo se kako urediti seznam po velikosti?
  - Priprave: zamenjava vrednosti prvega in drugega elementa v seznamu. Elementi v seznamu so označeni od 0 naprej

```
>>> seznam = [17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
>>> seznam
[17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
>>> zacasno = seznam[0]
>>> zacasno
17
>>> seznam[0] = seznam[1]
>>> seznam
[10, 10, 8, 13, 5, 4, 6, 20, 15, 18]
>>> seznam[1] = zacasno
>>> seznam
[10, 17, 8, 13, 5, 4, 6, 20, 15, 18]
```

Zanke - Urejanje po velikosti (4)

- Kako urediti seznam po velikosti?
  - Izpišimo elemente v seznamu

```
seznam = [17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
for i in seznam:
    print(i)
```

  - Napišimo še mesto v seznamu, na katerem je element

```
seznam = [17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
ii = 0
for i in seznam:
    print('mesto:', ii, 'vrednost:', i)
    ii = ii + 1
```

  - Veliko lepše in pravilneje je takole

```
seznam = [17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
dolzina = len(seznam)
for i in range(dolzina):
    print('mesto:', i, 'vrednost:', seznam[i])
```

Zanke - Urejanje po velikosti (5)

- Kako urediti seznam po velikosti?
  - Izpišimo vse številke od 9 … 1 v zanki **`for`**

```
for j in range(9, 0, -1):
    print(j)
```

Tako se premika puščica pri simulaciji. Spreglejmo, da gre puščica tudi na polje označeno z 0

  - Primerjamo po dve števili, zato zraven izpišimo še sosednje število

```
for j in range(9, 0, -1):
    print(j, j-1)
```

Zanke - Urejanje po velikosti (6)

- Kako urediti seznam po velikost
  - V vsakem naslednjem prehodu gre puščica za en element manj visoko
  - Tako, izpisali smo vse pare, ki jih moramo primerjati
  - Znamo to narediti lepše?
    - Poglejmo drugi parameter v funkciji range

```
for j in range(9, 0, -1):
    print(j, j-1)
for j in range(9, 1, -1):
    print(j, j-1)
for j in range(9, 2, -1):
    print(j, j-1)
for j in range(9, 3, -1):
    print(j, j-1)
for j in range(9, 4, -1):
    print(j, j-1)
for j in range(9, 5, -1):
    print(j, j-1)
for j in range(9, 6, -1):
    print(j, j-1)
for j in range(9, 7, -1):
    print(j, j-1)
for j in range(9, 8, -1):
    print(j, j-1)
```

Zanke - Urejanje po velikosti (6)

- Kako urediti seznam po velikosti?
  - Vsi pari na kratko

```
for i in range(9):
    for j in range(9, i, -1):
        print(j, j-1)
```

  - Vzemimo, da moramo primerjati števila v seznamu z `dolzina` elementi. V tem primeru napišemo:

```
dolzina = 10
for i in range(dolzina-1):
    for j in range(dolzina-1, i, -1):
        print(j, j-1)
```

Zanke - Urejanje po velikosti (7)

- Kako urediti seznam po velikosti?
  - Zdaj pa dodajmo seznam in poleg oznak (položaja v seznamu) izpišimo še vrednosti elementov v seznamu

```
seznam = [17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
dolzina = len(seznam)

for i in range(dolzina-1):
    for j in range(dolzina-1, i, -1):
        print('Primerjamo', end=' ')
        print('element:', j, 'vrednost:', seznam[j], end=' ')
        print('in', end=' ')
        print('element:', j-1, 'vrednost:', seznam[j-1])
```

Zanke - Urejanje po velikosti (8)

- Kako urediti seznam po velikosti?
  - Če je številka z manjšim indeksom večja od tiste z večjim indeksom, ju moramo zamenjati

```
seznam = [17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
dolzina = len(seznam)

for i in range(dolzina-1):
    for j in range(dolzina-1, i, -1):
        print('Primerjamo', end=' ')
        print('element:', j, 'vrednost:', seznam[j], end=' ')
        print('in', end=' ')
        print('element:', j-1, 'vrednost:', seznam[j-1])
        if seznam[j] < seznam[j-1]:
            zacasno = seznam[j-1]
            seznam[j-1] = seznam[j]
            seznam[j] = zacasno
```

prg24a.py

Zanke - Urejanje po velikosti (9)

- Kako urediti seznam po velikosti?

```
seznam = [17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
dolzina = len(seznam)

for i in range(dolzina-1):
    for j in range(dolzina-1, i, -1):
        # poučen izpis
        for num in range(dolzina):
            print(' '*(2-len(str(seznam[num]))), seznam[num], end=' ')
        print('\n' + ' '*4*(j-1) + '--- ---')
        input()
        # po potrebi zamenjajmo elementa v seznamu
        if seznam[j] < seznam[j-1]:
            zacasno = seznam[j-1]
            seznam[j-1] = seznam[j]
            seznam[j] = zacasno

print('Urejen seznam:', seznam)
```

prg24b.py

Funkcije (1)

- Če bi obstajala funkcija `izpisiSeznam`, bi bil prejšnji primer bolj pregleden uporabili bi jo lahko tudi večkrat

```
seznam = [17, 10, 8, 13, 5, 4, 6, 20, 15, 18]
dolzina = len(seznam)

for i in range(dolzina-1):
    for j in range(dolzina-1, i, -1):
        # poucen izpis
        izpisiSeznam()
        # po potrebi zamenjajmo elementa v seznamu
        if seznam[j] < seznam[j-1]:
            zacasno = seznam[j]
            seznam[j] = seznam[j-1]
            seznam[j-1] = zacasno
            izpisiSeznam()

print('Urejen seznam:', seznam)
```

prg24b.py

Funkcije (2)

- Python take funkcije ne pozna, pripraviti jo moramo sami.
- Stvar je preprosta – nekje nad prvim klicem, običajno čisto na vrhu programa, napišemo naslednje

```
def izpisiSeznam():
    for num in range(dolzina):
        print(' '*(2-len(str(seznam[num]))), seznam[num], end=' ')
    print('\n' + ' '*4*(j-1) + '--- ---')
    input()
```

prg25.py

Funkcije - spremenljivke

- V funkcijah lahko uporabljamo **spremenljivke**
  - spremenljivke, definirane v funkciji, so lokalne; obstajajo samo takrat, ko se funkcija izvaja
  - spremenljivke, definirane zunaj funkcijo so globalne
  - kadar ima lokalna spremenljivka enako ime kot globalna spremenljivka, lokalna spremenljivka zakrije globalno spremenljivko
- Kaj izpiše naslednji program?

```
def funkcija():
    i = 2
    print('v funkciji:', i, j)

i = j = 1
print('pred klicem:', i, j)
funkcija()
print('po klicu:', i, j)
```

Rešitev

```
pred klicem: 1 1
v funkciji: 2 1
po klicu: 1 1
```

prg26a.py

Funkcije

- Funkcija ima lahko vhodne parametre
- Napišimo funkcijo, ki nariše enakostranični trikotnik s poljubno višino

```
# funkcija narise enakostranicni trikotnik z visino visina
def narisiTrikotnik(visina):
    for i in range(visina):
        print(' ' * (40 – 2*i), '*' * (4*i + 1))
```

- Funkcijo lahko v programu uporabimo večkrat

```
narisiTrikotnik(3)
narisiTrikotnik(5)
narisiTrikotnik(7)
```

in tako narišemo novoletno jelko

prg26b.py

Funkcije

- Novoletno jelko znamo narisati še bolj elegantno
  - lokalni parameter: visina
  - parameter funkcije je lahko tudi spremenljivka;
    ta spremenljivka ima lahko drugačno ime: v

```
# funkcija narise enakostranicni trikotnik z visino visina
def narisiTrikotnik(visina):
    for i in range(visina):
        print(' ' * (40 - 2*i), '*' * (4*i + 1))

# risanje jelke
for v in range(3, 8, 2):
    narisiTrikotnik(v)
```

prg26c.py

Funkcije

- Funkcija seveda lahko kliče tudi drugo funkcijo

```
# funkcija narise enakostranicni trikotnik z visino visina
def narisiTrikotnik(visina):
    for i in range(visina):
        print(' ' * (40 - 2*i), '*' * (4*i + 1))

# risanje jelke
def jelka():
    for v in range(3, 8, 2):
        narisiTrikotnik(v)

# klic
jelka()
```

prg27.py

Funkcije

- Funkcije nam znajo tudi **vrniti kakšno vrednost**
- Če v funkciji potrebujemo kakšno knjižnico, si jo seveda lahko naložimo

```
from random import *

# funkcija vrne enega od dobrih moz
def dobriMoz():
    i = randint(0, 2)
    seznam = ['Miklavz', 'Bozicek', 'dedek Mraz'];
    return seznam[i]

# klic
print('Obiskal nas bo ', dobriMoz(), '.', sep='')
```

prg28a.py

Funkcije

- Prave matematične funkcije nam znajo iz vhodnih parametrov kaj izračunati in vrniti rezultat
- Funkcija vrne tisto, kar zapišemo za stavkom **`return`**

```
from math import *
# funkcija vrne ploščino trikotnika (Heronov obrazec)
def ploscinaTrikotnika(a, b, c):
    s = (a + b + c) / 2.0
    p = sqrt( s * (s-a) * (s-b) * (s-c) )
    return p
# vpis podatkov
stranica1 = float(input('Vnesi dolžino prve stranice'))
stranica2 = float(input('Vnesi dolžino druge stranice'))
stranica3 = float(input('Vnesi dolžino tretje stranice'))
# klic
t = ploscinaTrikotnika(stranica1, stranica2, stranica3)
print('Ploščina trikotnika je', t)
```

prg28b.py

Funkcije

- Vrnejo lahko tudi več stvari, seveda v obliki seznama

```
from math import *
# funkcija vrne ploščino in obseg trikotnika
def ploscinaInObsegTrikotnika(a, b, c):
    s = (a + b + c) / 2.0
    p = sqrt( s * (s-a) * (s-b) * (s-c) )
    o = a + b + c
    return [p, o]
# vpis podatkov
stranica1 = float(input('Vnesi dolžino prve stranice'))
stranica2 = float(input('Vnesi dolžino druge stranice'))
stranica3 = float(input('Vnesi dolžino tretje stranice'))
# klic
t = ploscinaInObsegTrikotnika(stranica1, stranica2,
stranica3)
print('Ploščina trikotnika je', t[0], 'obseg pa', t[1])
```

prg29a.py

Funkcije

- Funkcija naj pove ali je število praštevilo
  - Praštevila so vsa števila, ki so deljiva samo sama s sabo in z 1
- V funkciji lahko stavek **`return`** uporabimo večkrat

```
def prastevilo(n):
    if (type(n) != int) or (n <= 1):
        return False
    pra = True
    for i in range(2, n):
        if n % i == 0:
            pra = False
            break
    return pra
for i in range(1, 11):
    if prastevilo(i) == True:
        print('Število', i, 'je praštevilo.')
    else:
        print('Število', i, 'ni praštevilo.')
```

prg29b.py

Funkcije

- Funkcija naj pove ali je število praštevilo
  - Krajša različica
- Stavek **`return`** namesto stavka **`break`**

```
def prastevilo(n):
    if (type(n) != int) or (n <= 1):
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

for i in range(1, 11):
    if prastevilo(i) == True:
        print('Število', i, 'je praštevilo.')
    else:
        print('Število', i, 'ni praštevilo.')
```

prg30.py

Funkcije

- Kaj pa funkcija, ki kliče samo sebe (rekurzija)
- Primer: faktoriela
  - varianta 1: n! = n (n-1) (n-2) …1
  - varianta 2: n! = n (n-1)!, 0! = 1! = 1

```
def faktoriela1(n):
    prod = 1
    for i in range(1, n+1):
        prod = prod * i
    return prod
def faktoriela2(n):
    if n <= 1:
       return 1
    else:
        return n * faktoriela2(n-1)
print('5! =', faktoriela1(5), 'ali 5! =', faktoriela2(5))
```

Datoteke

- Do sedaj smo podatke, ki jih je od nas zahteval program, vnašali preko tipkovnice, nato pa nam je program rezultate izpisal na zaslon
- Pri bolj resnih obdelavah je lepo, če računalnik podatke prebere iz ene datoteke in jih nato zapiše v drugo
- Ogledali si bomo enostavno delo z datotekami

prg31.py

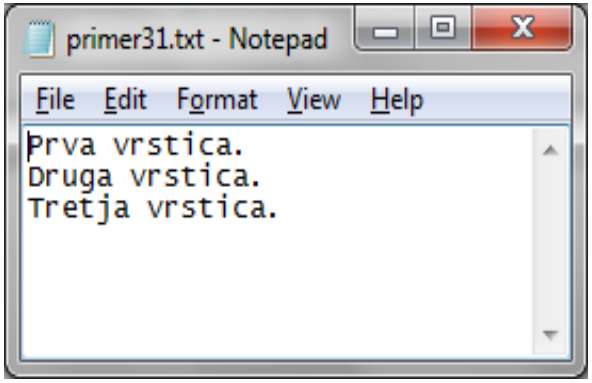
Datoteke

- Pisanje na zaslon …

```
print('Prva vrstica.')
print('Druga vrstica.')
print('Tretja vrstica.')
```

- … in pisanje v datoteko

```
# spremenljivko datoteka povežemo z dejansko
# datoteko na disku in jo odpremo za pisanje
datoteka = open('C:\\temp\\primer31.txt', 'w')

# v datoteko pišemo
print('Prva vrstica.', file=datoteka)
print('Druga vrstica.', file=datoteka)
print('Tretja vrstica.', file=datoteka)

# na koncu datoteko zapremo
datoteka.close()
```



Datoteke

- Pisanje v datoteko
  - Datoteko najprej odpremo. To storimo s funkcijo open
  - Vsi primeri v nadaljevanju predpostavljajo, da so datoteke shranjene v mapo c:\temp
  - Za Python je \ poseben znak, zato ga v nizu pravilno razume šele, ko ga napišemo dvakrat zapored, torej \\
  - Pri klicu povemo
    - pot do datoteke vključno z njenim imenom
    - ter način dostopa
      - 'w' – kadar želimo vanjo pisati na novo, od začetka (ang. write)
      - 'r' – kadar želimo iz nje brati (ang. read)
      - 'a' – kadar želimo dodajati stvari na konec obstoječe datoteke (ang. append)
      - 'r+' ali 'w+' – kadar želimo hkrati brati iz datoteke in pisati vanjo

Datoteke

- Pisanje v datoteko
  - v datoteko pišemo z ukazom `print`
  - za razliko od pisanja na zaslon namesto
    ```
    print(...)
    ```
    zapišemo
    ```
    print(..., file=datoteka)
    ```
  - za pisanje v datoteko veljajo enaka pravila kot za pisanje na zaslon
  - nazadnje datoteko z malce čudnim klicem
    ```
    f.close()
    ```
    zapremo. Potem, ko je zaprta, jo lahko odpremo v kakšnem drugem programu.

prg32a.py

Datoteke

- Branje iz datoteke
  - Python bere datoteke vrstico po vrstico
  - Preberimo datoteko, ki smo jo ustvarili v prejšnjem primeru
    - Kadar vemo koliko vrstic ima datoteka je stvar zelo preprosta

```
# zdaj pa te tri vrstice preberimo in izpišimo na zaslon
v1, v2, v3 = open('c:\\temp\\primer31.txt', 'r')
print(v1)
print(v2)
print(v3)
```

prg32b.py

Datoteke

- Branje iz datoteke
  - Če števila vrstic ne poznamo, postopamo takole

```
# branje poljubnega števila vrstic
datoteka = open('c:\\temp\\primer31.txt', 'r')

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

  - Kaj pa zapiranje datoteke pri branju
    - pri pisanju je to nujno, saj se mnogokrat šele takrat vsebina dejansko zapiše
    - pri branju je to lepo, nič hudega pa ni, če to malenkost prepustimo Pythonu, da jo opravi namesto nas kadar mu to ustreza

prg33.py

Datoteke

- Primer: jelko iz primera 26c narišimo v datoteko
  - Funkcijama za risanje smo dodali povezavo na datoteko

```
# funkcija narise enakostranični trikotnik z višino visina
def narisiTrikotnik(visina, d):
    for i in range(visina):
        print(' ' * (40 - 2*i), '*' * (4*i + 1), file=d)

# risanje jelke
def jelka(d):
    for v in range(3, 8, 2):
        narisiTrikotnik(v, d)

# klic
datoteka = open('c:\\temp\\primer33.txt','w')
jelka(datoteka)
datoteka.close()
```

prg34a.py

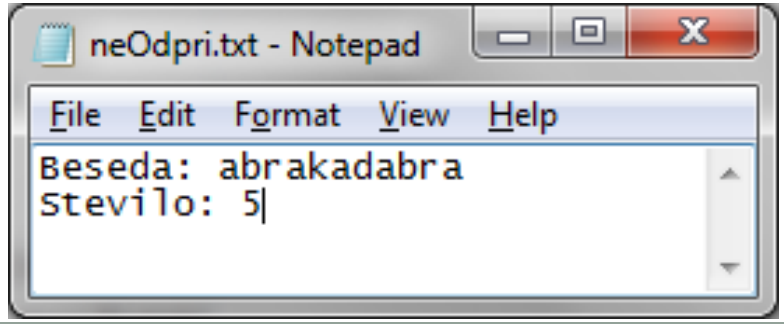
Datoteke

- Primer: ugibanje gesla iz naloge 21 še enkrat

```
# vstopna kontrola še enkrat

skrivnost = 'uiktnt'
poskusov = 3

while poskusov > 0:
    geslo = input('Kdo gre?')
    if geslo == skrivnost:
        break
    poskusov = poskusov - 1
    print('Imaš še', poskusov, 'poskusov.')

if poskusov > 0:
    print('Vstopi, prosim!')
else:
    print('Kar zunaj ostani!')
```

Datoteke

- Primer: ugibanje gesla iz naloge 21 še enkrat
  - Skrivnostno besedo in število poskusov raje preberimo iz datoteke




```
>>> nizSkrivnost, nizPoskusov = open('C:\\temp\neOdpri.txt','r')
>>> nizSkrivnost
'Beseda: abrakadabra\n'
>>> nizSkrivnost[8:-1]
'abrakadabra'
>>> nizPoskusov
'Stevilo: 5'
>>> nizPoskusov[8:]
' 5'
>>> int(nizPoskusov[8:])
5
```

prg34b.py

Datoteke

- Primer: ugibanje gesla iz naloge 21 še enkrat

```
# vstopna kontrola z geslom zapisanim v datoteki
nizSkrivnost, nizPoskusov = open('C:\\temp\\neOdpri.txt','r')
skrivnost = nizSkrivnost[8:-1]
poskusov = int(nizPoskusov[8:])
while poskusov > 0:
    geslo = input('Kdo gre?')
    if geslo == skrivnost:
        break
    poskusov = poskusov - 1
    print('Imaš še', poskusov, 'poskusov.')
if poskusov > 0:
    print('Vstopi, prosim!')
else:
    print('Kar zunaj ostani!')
```

prg35a.py

Datoteke

- Primer: graf funkcije $y = x^2$ na območju [-2, 2]
  - Narišimo ga!
  - V Pythonu tega ne znamo. Zato bomo rezultate shranili v tabelo, jo uvozili v Excel in graf narisali v njem.
  - Najprej izračunajmo vrednosti.

```
# vhodni podatki
obmocjeMin = -2
obmocjeMax = 2.01
korak = 0.1

# izračun
x = obmocjeMin
while (x < obmocjeMax):
    y = x ** 2
    print(x, y)
    x = x + korak
```

prg35b.py

Datoteke

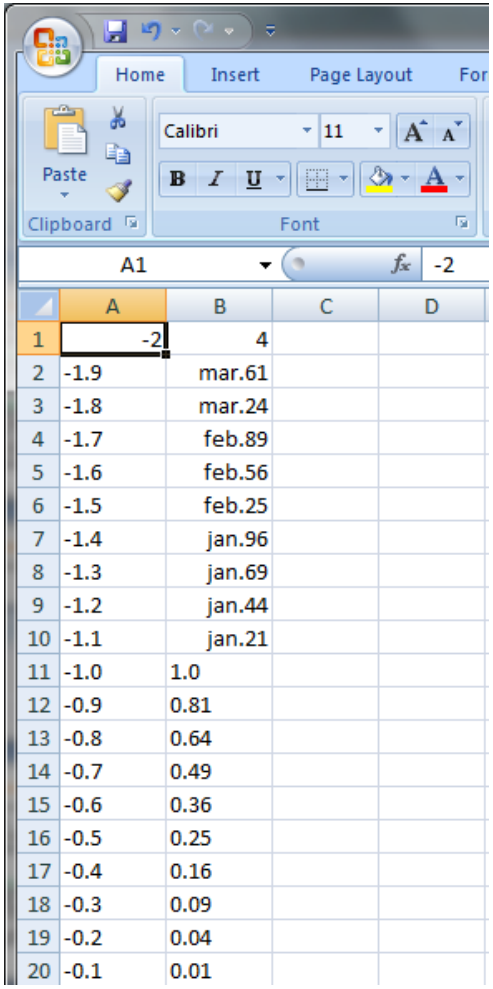
- Primer: graf funkcije $y = x^2$ na območju [-2, 2]
  - znak \t v nizu predstavlja tabulator

```
# vhodni podatki
obmocjeMin = -2
obmocjeMax = 2.01
korak = 0.1
imeDatoteke = 'c:\\temp\\izracun.txt'
# izračun in izpis v datoteko
datoteka = open(imeDatoteke, 'w')
x = obmocjeMin
while (x < obmocjeMax ):
    y = x ** 2
    print(x, '\t', y, file=datoteka)
    x = x + korak
datoteka.close()
```

  - Rezultati so zapisani v datoteki c:\temp\izracun.txt

Datoteke

- Zdaj pa v Excel …
- Graf funkcije $y = x^2$ na območju [-2, 2]
  - Če spregledamo podrobnosti pri uvažanju datoteke c:\temp\izracun.txt, in za vsak stolpec posebej ne povemo, da je decimalno ločilo pika, dobimo nekaj takega
  - … kar pa ni tisto, kar smo pričakovali.



	A	B	C	D
1	-2	4		
2	-1.9	mar.61		
3	-1.8	mar.24		
4	-1.7	feb.89		
5	-1.6	feb.56		
6	-1.5	feb.25		
7	-1.4	jan.96		
8	-1.3	jan.69		
9	-1.2	jan.44		
10	-1.1	jan.21		
11	-1.0	1.0		
12	-0.9	0.81		
13	-0.8	0.64		
14	-0.7	0.49		
15	-0.6	0.36		
16	-0.5	0.25		
17	-0.4	0.16		
18	-0.3	0.09		
19	-0.2	0.04		
20	-0.1	0.01		

prg35c.py

Datoteke

- Primer: graf funkcije $y = x^2$ na območju [-2, 2]
  - Klikanja ne maramo, torej …
  - … številke zapišemo kot nize,
  - v nizih zamenjamo piko v vejico in
  - popravljeni niz zapišemo v datoteko

  - Najprej poglejmo, kako v nizu zamenjati piko z vejico,

```
# funkcija spremeni pike v nizu v vejice
def pikaVVejico(niz):
    novNiz = ''
    for znak in niz:
        if znak == '.':
            znak = ','
        novNiz = novNiz + znak
    return novNiz
```

prg35c.py

Datoteke

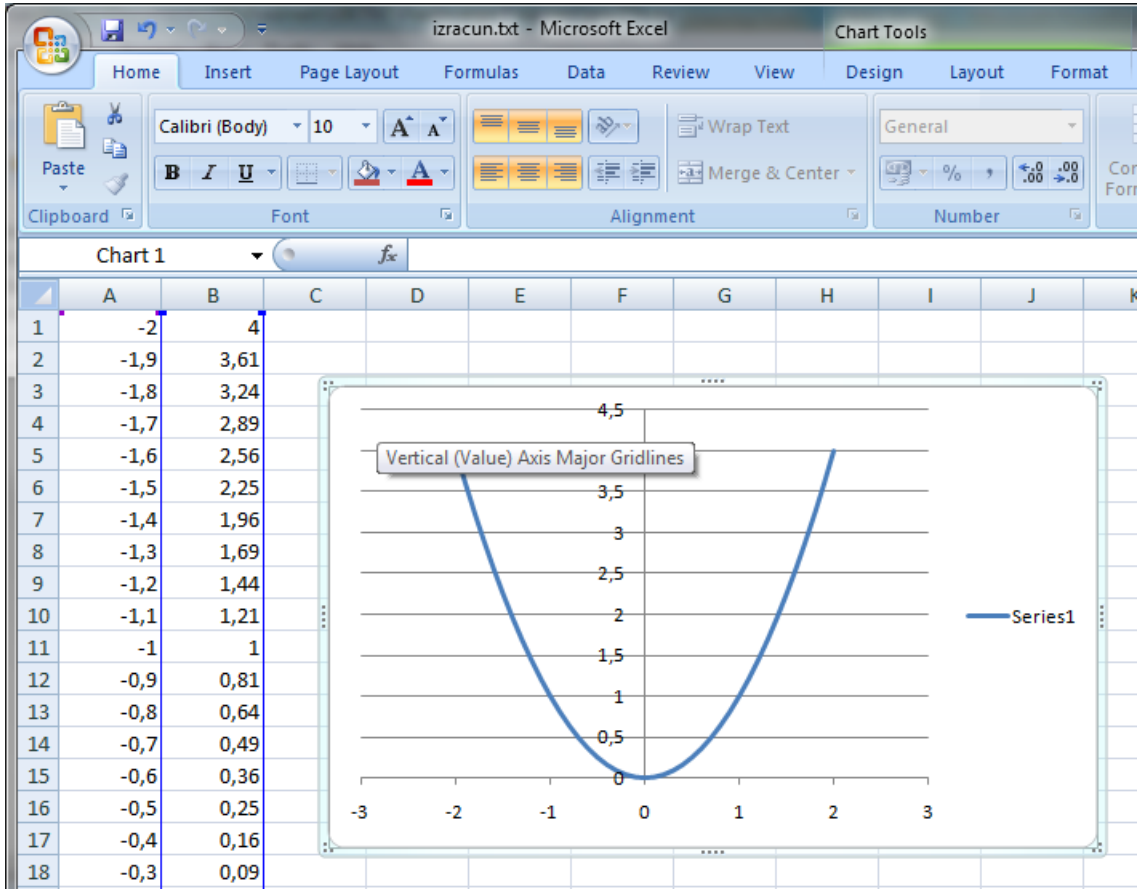
- Primer: graf funkcije $y = x^2$ na območju [-2, 2]
  - nato pa funkcijo uporabimo v našem programu.

```
# vhodni podatki
obmocjeMin = -2
obmocjeMax = 2.01
korak = 0.1
imeDatoteke = 'c:\\temp\\izracun.txt'
# izračun in izpis v datoteko
datoteka = open(imeDatoteke, 'w')
x = obmocjeMin
while (x < obmocjeMax ):
    y = x ** 2
    print(pikaVVejico(str(x)), '\t', end ='', file=datoteka)
    print(pikaVVejico(str(y)), file=datoteka)
    x = x + korak
datoteka.close()
```

  - Zmaga je zagotovljena …

Datoteke

- Primer: graf funkcije $y = x^2$ na območju [-2, 2]
  - … BRAVO !!!



	A	B
1	-2	4
2	-1,9	3,61
3	-1,8	3,24
4	-1,7	2,89
5	-1,6	2,56
6	-1,5	2,25
7	-1,4	1,96
8	-1,3	1,69
9	-1,2	1,44
10	-1,1	1,21
11	-1	1
12	-0,9	0,81
13	-0,8	0,64
14	-0,7	0,49
15	-0,6	0,36
16	-0,5	0,25
17	-0,4	0,16
18	-0,3	0,09

Še nekaj za konec

- Kot smo videli, datoteko zapremo z ukazom `datoteka.close()`
- Tak klic je možen zato, ker je `datoteka` posebna spremenljivka, ki poleg vrednosti vključuje tudi funkcije.
- No, pravzaprav je `datoteka` objekt.
- Objekti so tudi vse ostale spremenljivke, saj je Python objektni programski jezik.
- To pomembno lastnost sem pred vami skrival do konca, saj bi nam objektni pogled na svet pri prvih korakih v programiranje stvari preveč zavozlal.
- Tisti, ki vas to zanima pa seveda lepo povabljeni v nadaljnje raziskovanje Pythona.
- Če se boste lotili grafike, naprednega dela z nizi, in še česa, bodo objekti vaši zvesti spremljevalci