7. Textové súbory

video prezentácia

textové súbory

So súbormi súvisia znakové reťazce ale aj príkazy `input()` a `print()` na vstup a výstup:

- znakové reťazce sú postupnosti znakov (v kódovaní **Unicode**), ktoré môžu obsahovať aj znaky konca riadka `'\n'`
- funkcia `input()` prečíta zo štandardného vstupu znakový reťazec, t.j. **číta** z klávesnice
- funkcia `print()` **zapíše** reťazec na štandardný výstup, t.j. **vypisuje** do textového konzolového okna

Textový súbor

Je postupnosť znakov, ktorá môže obsahovať aj znaky konca riadka. Väčšinou je táto postupnosť uložená na disku v súbore.

Na textový súbor sa môžeme pozerať ako na **postupnosť riadkov** (môže to byť aj prázdna postupnosť), pričom každý riadok je postupnosťou znakov (znakový reťazec), ktorá je ukončená špeciálnym znakom `'\n'`.

So súbormi sa vo všeobecnosti pracuje takto:

- najprv musíme vytvoriť spojenie medzi našim programom a súborom, ktorý je veľmi často v nejakej externej pamäti - tomuto hovoríme **otvoriť súbor**
- teraz sa dá so súborom pracovať, t.j. môžeme z neho čítať, alebo do neho zapisovať
- keď so súborom skončíme prácu, musíme zrušiť spojenie, hovoríme tomu **zatvoriť súbor**

Čítanie zo súboru

Najprv sa naučíme čítať zo súboru, čo označuje, že náš program sa postupne dozvedá, aký je obsah súboru.

Otvorenie súboru na čítanie

Súbor otvárame volaním štandardnej funkcie `open()`, ktorej oznámime meno súboru, ktorý chceme čítať. Táto funkcia vráti **referenciu** na súborový objekt (hovoríme tomu aj **dátový prúd**, t.j. **stream**):

```
premenná = open('meno_súboru', 'r')
```

Do **súborovej premennej** sa priradí **spojenie** s uvedeným súborom. Najčastejšie je tento súbor umiestnený v tom istom priečinku, v ktorom sa nachádza samotný Python skript. Meno

súboru môže obsahovať aj celú cestu k súboru, prípadne môže byť relatívna k umiestneniu skriptu.

Meno súborovej premennej je vhodné voliť tak, aby nejako zodpovedalo vzťahu k súboru, napríklad:

```
subor = open('pribeh.txt', 'r')
kniha = open('c:/dokumenty/python.txt', 'r')
subor_s_cislami = open('../texty/cisla.txt', 'r')
```

V týchto príkladoch otvárania súborov vidíte, že meno súboru môže byť kompletná cesta `'c:/dokumenty/python.txt'` alebo relatívna k pozícii skriptu `'../texty/cisla.txt'`.

Čítanie zo súboru

Najčastejšie sa informácie zo súboru čítajú po celých riadkoch. Možností, ako takto čítať je viac. Základný spôsob je:

```
riadok = subor.readline()
```

Funkcia `readline()` je **metódou** súborovej premennej, preto za meno súborovej premennej píšeme bodku a meno funkcie. Funkcia vráti znakový reťazec - prečítaný riadok aj s koncovým znakom `'\n'`. Súborová premenná si zároveň zapamätá, kde v súbore sa práve nachádza toto čítanie, aby každé ďalšie zavolanie `readline()` čítalo ďalšie a ďalšie riadky.

Funkcia vráti prázdny reťazec `''`, ak sa už prečítali všetky riadky a teda pozícia čítania v súbore je na konci súboru.

Zrejme prečítaný riadok nemusíme priradiť do premennej, ale môžeme ho spracovať aj inak, napríklad:

```
subor = open('subor.txt', 'r')
print(subor.readline())
print('dlzka =', len(subor.readline()))
print(subor.readline()[::-1])
```

V tomto programe sa najprv vypíše obsah prvého riadka, potom dĺžka druhého riadka (aj s koncovým znakom `'\n'`) a na záver sa vypíše tretí riadok ale otočený (aj s koncovým znakom `'\n'`, ktorý sa vypíše ako prvý).

Zatvorenie súboru

Keď skončíme prácu so súborom, **uzavrieme** otvorené spojenie volaním metódy:

```
subor.close()
```

Tým sa uvoľnia všetky systémové zdroje (resources), ktoré boli potrebné pre otvorený súbor. Zrejme so zatvoreným súborom sa už nedá ďalej pracovať a napríklad čítanie by vyvolalo chybovú správu.

Ďalej ukážeme, ako môžeme pracovať s textovým súborom. Predpokladajme, že máme pripravený nejaký textový súbor, napríklad `'subor.txt'`:

```
Od ucenia este
nikto nezomrel,
  ale naco riskovat.

Albert Einstein
```

Tento súbor má 5 riadkov (štvrtý je prázdny) a preto ho môžeme celý prečítať a vypísať takto:

```
t = open('subor.txt', 'r')

for i in range(5):
    riadok = t.readline()
    print(riadok)

t.close()
```

program vypíše:

```
Od ucenia este

nikto nezomrel,

  ale naco riskovat.


Albert Einstein
```

Keďže metóda `readline()` prečíta zo súboru celý riadok aj s koncovým `'\n'`, príkaz `print()` k tomu pridáva ešte jeden svoj `'\n'` a preto je za každým vypísaným riadkom ešte jeden prázdny. Buď príkazu `print()` povieme, aby na koniec riadka nevkladal prechod na nový riadok (napríklad `print(riadok, end='')`), alebo pred samotným výpisom z reťazca `riadok` vyhodíme posledný znak, napríklad:

```
t = open('subor.txt', 'r')
for i in range(5):
    riadok = t.readline()
    print(riadok[:-1])
t.close()
```

Takéto vyhadzovanie posledného znaku z reťazca môže nefungovať celkom správne pre posledný riadok súboru, ktorý nemusí byť ukončený znakom `'\n'`.

Zistenie konca súboru

Najväčším nedostatkom predchádzajúceho programu je to, že predpokladá veľkosť vstupného súboru presne 5 riadkov. Ak by sme tento počet dopredu nepoznali, musíme použiť nejaký iný spôsob. Keďže metóda `readline()` vráti na konci súboru **prázdny reťazec** `''` (pozor, nie jednoznakový reťazec `'\n'`), môžeme práve túto podmienku využiť na testovanie konca súboru:

```
t = open('subor.txt', 'r')
riadok = t.readline()
while riadok != '':
    print(riadok, end='')
    riadok = t.readline()
t.close()
```

Tento program už správne vypíše všetky riadky súboru, hoci nevidíme, či je štvrtý riadok prázdny alebo obsahuje aj nejaké medzery:

```
Od ucenia este
nikto nezomrel,
  ale naco riskovat.

Albert Einstein
```

šablóna čítania s while-cyklom

Naše programy, ktoré čítajú po riadkoch, budú často vyzerať takto:

```
subor = open('meno súboru', 'r')

riadok = subor.readline()
while riadok != '':
    # ... spracuj riadok
    riadok = subor.readline()

subor.close()
```

Môžeme to prepísať aj s použitím `break`:

```
t = open('subor.txt', 'r')
while True:
    riadok = t.readline()
    if riadok == '':
        break
    print(riadok, end='')
t.close()
```

Niekedy sa nám môže hodiť taký výpis prečítaného reťazca, ktorý napríklad zobrazí nielen medzery na konci reťazca, ale aj ukončovací znak `'\n'`. Využijeme na to štandardnú funkciu `repr()`.

funkcia `repr()`

Volanie štandardnej funkcie:

```
repr(reťazec)
```

vráti takú reťazcovú reprezentáciu daného parametra, aby sme po jeho vypísaní (napríklad funkciou `print()`) dostali presne taký tvar, aký očakáva Python pri zadávaní konštanty, teda aj s apostrofmi, prípadne aj so znakom `'\'` pri špeciálnych znakoch.

Môže sa použiť aj pri ladení a testovaní, lebo máme lepší prehľad o skutočnom obsahu reťazca. Napríklad:

```
>>> a = 'ahoj   \naj "apostrof" \' v texte  \n'
>>> print(a)
    ahoj
    aj "apostrof" ' v texte

>>> print(repr(a))
    'ahoj   \naj "apostrof" \' v texte  \n'
```

Doplníme do while-cyklu o volanie funkcie `repr()`:

```
t = open('subor.txt', 'r')
riadok = t.readline()
while riadok != '':
    print(repr(riadok))
    riadok = t.readline()
t.close()
```

Po spustení vidíme, že sa vypíše:

```
'Od ucenia este\n'
'nikto nezomrel,   \n'
'  ale naco riskovat.\n'
'\n'
'Albert Einstein\n'
```

Namiesto `while riadok != '':` môžeme zapísať `while riadok:`.

Vidíme, že druhý riadok obsahuje medzery aj na konci riadka. Ak by sme pri čítaní súboru nepotrebovali informácie o medzerách na začiatku a konci riadkov, môžeme využiť reťazcovú metódu `strip()`:

```
t = open('subor.txt', 'r')
riadok = t.readline()
while riadok:
    print(repr(riadok.strip()))
    riadok = t.readline()
t.close()
```

vypíše:

```
'Od ucenia este'
'nikto nezomrel,'
'ale naco riskovat.'
''
'Albert Einstein'
```

Všimnite si, že takto sme sa zbavili aj záverečného znaku `'\n'`. Ak by sme namiesto `riadok.strip()` použili `riadok.rstrip()`, vyhodia sa medzerové znaky len od konca reťazca (sprava) a na začiatku riadkov medzery ostávajú.

Použitie for-cyklu pre čítanie zo súboru

Python umožňuje použiť for-cyklus, aj pre súbory, o ktorých dopredu nevieme, koľko majú riadkov. For-cyklus má vtedy tvar:

```
for riadok in súborová_premenná:
    prikazy
```

kde `riadok` je ľubovoľná premenná cyklu, do ktorej sa budú postupne priraďovať všetky prečítané riadky - POZOR! aj s koncovým `'\n'`, **súborová_premenná** musí byť otvoreným súborom na čítanie.

Program sa teraz výrazne zjednoduší:

```
t = open('subor.txt', 'r')
for riadok in t:
    print(repr(riadok))
t.close()
```

šablóna čítania s for-cyklom

S použitím for-cyklu môžu byť naše programy, ktoré čítajú súbor po riadkoch, ešte čitateľnejšie:

```
subor = open('meno súboru', 'r')

for riadok in subor:
    # ... spracuj riadok

subor.close()
```

Python v tomto prípade považuje otvorený súbor za **postupnosť** riadkov (aj s koncovým `'\n'`) a for-cyklus dokáže prechádzať prvky ľubovoľných postupností.

Takýto for-cyklus bude fungovať aj vtedy, keď sme už zo súboru niečo čítali a potrebujeme spracovať už len zvyšok súboru, napríklad:

```
t = open('subor.txt', 'r')
riadok = t.readline()
print('najprv som precital:', repr(riadok.rstrip()))
print('v subore ostali este tieto riadky:')
for riadok in t:
    print(repr(riadok.rstrip()))
t.close()
```

Teraz sa vypíše:

```
najprv som precital: 'Od ucenia este'
v subore ostali este tieto riadky:
```

```
'nikto nezomrel,'
'ale naco riskovat.'
''
'Albert Einstein'
```

Pozor!

Vo for-cykle, ktorý prechádza riadky súboru, sa nezvykne volať `readline()`, nakoľko takto sa po každom prečítaní riadku pomocou `for`, prečíta ešte jeden, ktorý sa často už nespracuje. Napríklad:

```
t = open('subor.txt', 'r')
for riadok in t:
    print(repr(riadok))
    t.readline()
t.close()
```

Tento program vypíše len každý druhý riadok súboru.

Prečítanie celého súboru do jedného reťazca

Zapíšme riešenie takejto úlohy: do jednej reťazcovej premennej prečítame všetky riadky súboru, pričom im ponecháme koncové `'\n'`. Zrejme, ak by sme takýto reťazec naraz celý vypísali (pomocou `print()`), dostali by sme kompletný výpis. Zapíšme riešenie pomocou **pripočítavacej šablóny**:

```
t = open('subor.txt', 'r')
cely_subor = ''
for riadok in t:
    cely_subor = cely_subor + riadok
t.close()
print(cely_subor, end='')
```

Všimnite si, že riadok programu `cely_subor = cely_subor + riadok` by sme mohli zapísať aj takto `cely_subor += riadok`

To, čo sme prácne skladali cyklom, za nás urobí metóda `read()`, teda

```
t = open('subor.txt', 'r')
cely_subor = t.read()
t.close()
print(cely_subor, end='')
```

Samozrejme, takéto skladanie súboru do jednej reťazcovej premennej môžeme urobiť len vtedy, ak spracovávaný súbor nie je väčší ako kapacita pamäte pre Python (závisí to od vášho počítača, ale je to od niekoľkých 100MB po GB).

metóda `súbor.read()`

`súbor.read()`
`súbor.read(`*`počet_znakov`*`)`

Parametre

počet_znakov – určuje počet znakov, ktoré budem chcieť zo súboru prečítať

Metóda prečíta zo súboru a vráti ako výsledok funkcie znakový reťazec dĺžky `počet_znakov`. Ak už v súbore toľko znakov nie je, alebo sme nezadali počet znakov, vráti reťazec maximálne možnej dĺžky (alebo aj prázdny reťazec). Napríklad:

```
>>> t = open('subor.txt')
>>> print('prvý znak =', repr(t.read(1)))
>>> print('ďalších 10 znakov =', repr(t.read(10)))
>>> print('zvyšok súboru =', repr(t.read()))
```

Slovenčina v súbore

Hoci znakové reťazce sa v Pythone uchovávajú v kódovaní **Unicode**, pri práci so súbormi, ktoré obsahujú znaky s diakritikou, musíme upresniť aj kódovanie v súbore. Samotné súbory môžu mať pri uložení na disku rôzne kódovania (závisí to aj od vášho operačného systému). Takými kódovaniami môžu byť, napríklad `'cp1250'`, `'iso88591'`, `'utf-8'`, … a pri ich otváraní treba toto kódovanie uvádzať ako parameter funkcie `open()`. Súbory, ktoré dostanete na čítanie v tomto kurze, budú mať väčšinou kódovanie `'utf-8'`, ale vaše vlastné súbory môžu mať aj iné kódovanie.

Preto súbor s diakritikou najčastejšie otvárame takto:

```
subor = open(meno_suboru, 'r', encoding='utf-8')
```

Zápis do súboru

Doteraz sme čítali už existujúci súbor. Teraz sa naučíme textový súbor aj vytvárať. Bude to veľmi podobné ako pri čítaní súboru.

Otvorenie súboru

do **súborovej premennej** sa priradí **spojenie** so súborom:

```
subor = open('meno_súboru', 'w')
```

Súbor bude umiestnený v tom istom priečinku, kde sa nachádza samotný Python skript (resp. treba uviesť cestu). Ak tento súbor ešte neexistoval, tento príkaz ho **vytvorí** (vytvorí sa prázdny súbor). Ak takýto súbor už existoval, tento príkaz ho **vyprázdni**. Treba si dávať pozor, lebo omylom môžeme prísť o dôležitý súbor.

Možností, ako zapisovať riadky do súboru je viac. My si postupne ukážeme dva z nich: zápis pomocou základnej metódy pre zápis `write()` a pomocou nám známej štandardnej funkcie `print()`. Najprv metóda `write()`:

Zápis do súboru

Zápis nejakého reťazca do súboru urobíme pomocou volania:

```
subor.write(reťazec)
```

Táto metóda zapíše zadaný reťazec na momentálny koniec súboru. Ak chceme, aby sa v súbore objavili aj koncové znaky `'\n'`, musíme ich pridať do reťazca.

Niekoľko za sebou idúcich zápisov do súboru môžeme spojiť do jedného, napríklad:

```
subor.write('Py')
subor.write('thon\nje')
subor.write(' najlepsi\n')
```

môžeme zapísať jediným volaním metódy `write()`:

```
subor.write('Python\nje najlepsi\n')
```

Zatvorenie súboru

Tak, ako sme pri čítaní súboru museli na záver súbor zatvárať, musíme zatvárať súbor aj pri vytváraní:

```
subor.close()
```

Metóda `close()` skončí prácu so súborom, t.j. zruší **spojenie** s fyzickým súborom na disku. Bez volania tejto metódy nemáme zaručené, že Python naozaj fyzicky stihol zapísať všetky reťazce z volania `write()` na disk. Tiež operačný systém by mohol mať problém so znovu otvorením ešte nezatvoreného súboru.

Zápis do súboru ukážeme na príklade, v ktorom vytvoríme niekoľko riadkový súbor `'subor1.txt'`:

```
subor = open('subor1.txt', 'w')
subor.write('zoznam prvocisel:\n')
for ix in 2, 3, 5, 7, 11, 13:
    subor.write(f'cislo {ix} je prvocislo\n')
subor.close()
```

Program najprv do súboru zapísal jeden riadok `'zoznam prvocisel:'` a za ním ďalších 6 riadkov:

```
zoznam prvocisel:
cislo 2 je prvocislo
cislo 3 je prvocislo
cislo 5 je prvocislo
cislo 7 je prvocislo
cislo 11 je prvocislo
cislo 13 je prvocislo
```

Zápis do súboru pomocou print()

Doteraz sme štandardný príkaz `print()` používali na výpis do textovej plochy Shellu (do konzoly). Výstup z už odladeného programu môžeme veľmi jednoducho presmerovať do súboru.

Program vytvorí súbor `'nahodne_cisla.txt'`, do ktorého zapíše pod seba 100 náhodných čísel:

```
import random

subor = open('nahodne_cisla.txt', 'w')
for i in range(100):
    print(random.randint(1, 100), file=subor)
subor.close()
```

Všimnite si nový parameter pri volaní funkcie `print()`, pomocou ktorého presmerujeme výstup do nášho súboru (tu musíme uviesť súborovú premennú už otvoreného súboru na zápis).

Ak by sme chceli, aby boli čísla v súbore nie v jednom stĺpci ale v jednom riadku oddelené medzerou, zapísali by sme:

```
import random

subor = open('nahodne_cisla.txt', 'w')
for i in range(100):
    print(random.randint(1, 100), end=' ', file=subor)
print(file=subor)
subor.close()
```

Kopírovanie súboru

Ak potrebujeme obsah jedného súboru prekopírovať do druhého (pritom možno niečo spraviť s každým riadkom), môžeme použiť dve súborové premenné, napríklad:

```
odkial = open('subor.txt', 'r')
kam = open('subor2.txt', 'w')
for riadok in odkial:
    riadok = riadok.strip()
    if riadok != '':
        kam.write(riadok + '\n')
odkial.close()
kam.close()
```

Program postupne prečíta všetky riadky, vyhodí medzery zo začiatku a z konca každého riadka, a ak je takýto riadok neprázdny, zapíše ho do druhého súboru (keďže `strip()` vyhodil z riadka aj koncové `'\n'`, museli sme ho tam vo volaní metódy `write()` pridať).

Táto istá úloha by sa dala riešiť aj pomocou jednej súborovej premennej - najprv súbor čítame a do jednej reťazcovej premennej pripravujeme obsah nového súboru, nakoniec ho celý zapíšeme:

```
t = open('subor.txt', 'r')
cely = ''
for riadok in t:
    riadok = riadok.strip()
    if riadok != '':
        cely += riadok + '\n'
t.close()

t = open('subor2.txt', 'w')
t.write(cely)
t.close()
```

Ak by sme pri kopírovaní riadkov nepotrebovali meniť nič, môžeme použiť metódu `read()`, napríklad:

```
t = open('subor.txt', 'r')
cely = t.read()
t.close()

t = open('subor2.txt', 'w')
t.write(cely)
t.close()
```

Na prácu so súbormi môžeme využiť špeciálnu programovú konštrukciu `with`, pomocou ktorej si Python *domyslí*, že sme už so súborom skončili pracovať a teda ho automaticky zatvorí. Samotný príkaz má aj iné využitie ako pre prácu so súbormi, ale v tomto kurze sa s tým nestretneme.

Konštrukcia with

Všeobecný tvar príkazu je:

```
with open(...) as premenna:
    prikaz
    prikaz
    ...
```

Touto príkazovou konštrukciou sa otvorí požadovaný súbor a referencia na súbor sa priradí do premennej uvedenej za `as`. Ďalej sa vykonajú všetky príkazy v bloku a po ich skončení sa súbor **automaticky** zatvorí. Urobí sa skoro to isté, ako

```
premenna = open(...)
prikaz
prikaz
...
premenna.close()
```

Odporúčame pri práci so súbormi používať čo najviac práve túto konštrukciu, čo oceníte, napríklad aj pri práci so súbormi vo funkciách, v ktorých príkaz `return`, ak sa použije vo vnútri bloku `with`, automaticky zatvorí otvorené súbory.

Ukážme niekoľko príkladov zápisu pomocou `with`:

1. Prečítaj a vypíš obsah celého súboru:

```
2. with open('subor.txt', 'r') as subor:
3.     print(subor.read())
```

4. Vytvor súbor s tromi riadkami:

```
5. with open('subor.txt', 'w') as file:
6.     print('prvy\ndruhy\ntreti\n', file=file)
```

   Všimnite si tu použitie mena súborovej premennej: nazvali sme ju `file` rovnako ako meno parametra vo funkcii `print()`, preto musíme presmerovanie do súboru zapísať ako `print(..., file=file)`: formálnemu parametru `file` priradíme hodnotu skutočného parametra `file`.

7. Vytvor súbor 100 náhodných čísel:

```
8. import random
9.
10. with open('cisla.txt', 'w') as file:
11.     for i in range(100):
12.         file.write(f'{random.randint(0, 1000)} ')
```

Automatické zatváranie súboru

Python sa v jednoduchých prípadoch sám z vlastnej iniciatívy snaží korektne zatvoriť otvorené súbory, keď už sme s nimi skončili pracovať a pritom sme nepoužili metódu `close()`. V serióznych aplikáciách toto nebudeme používať, ale pri jednoduchých testoch a ukážkach sa to objaviť môže.

V nasledovných príkladoch využívame to, že funkcia `open()` vracia ako výsledok súborovú premennú, t.j. spojenie na súbor. Ak toto spojenie potrebujeme použiť len jednorázovo, nemusíme to priradiť do premennej, ale použijeme ho priamo napríklad s volaním nejakej metódy.

Ak do súboru zapisujeme len jedenkrát a hneď ho zatvárame, nemusíme na to vytvárať súborovú premennú, ale priamo pri otvorení urobíme jeden zápis. Vtedy sa súbor automaticky zatvorí. Napríklad:

```
>>> open('subor2.txt', 'w').write('first line\nsecond line\nend of file\n')
```

Týmto jedným príkazom sme vytvorili nový súbor `'subor3.txt'`, zapísali sme do neho 3 riadky a automaticky sa na záver zatvoril (dúfajme…). Korektný zápis. ktorý by sme použili v programe:

```
with open('subor2.txt', 'w') as f:
    f.write('first line\nsecond line\nend of file\n')
```

Podobne by sme to zapísali aj pomocou príkazu `print()`:

```
>>> print('first line\nsecond line\nend of file', file=open('subor3.txt', 'w'))
```

alebo radšej korektne:

```
with open('subor3.txt', 'w') as f:
    print('first line\nsecond line\nend of file', file=f)
```

Nezabudnite, že ak súbor `'subor3.txt'` niečo pred tým obsahoval, týmto príkazom sa celý prepíše.

Vyššie uvedený príklad, ktorý kopíroval kompletný súbor:

```
t = open('subor.txt', 'r')
cely = t.read()
t.close()
t = open('subor2.txt', 'w')
t.write(cely)
t.close()
```

by sa dal úsporne zapísať takto:

```
>>> open('subor2.txt', 'w').write(open('subor.txt', 'r').read())
```

čo je zrejme veľmi ťažko čitateľné, a my to určite budeme zapisovať radšej takto korektne:

```
with open('subor.txt', 'r') as r:
    with open('subor2.txt', 'w') as w:
        w.write(r.read())
```

Niekedy to môžete vidieť aj v takomto tvare:

```
with open('subor.txt', 'r') as r, open('subor2.txt', 'w') as w:
    w.write(r.read())
```

To znamená, že do príkazu `with` môžeme zadať naraz viac otvorených súborov (oddelených čiarkou). Po skončení bloku príkazov sa všetky takto otvorené súbory automaticky zatvoria.

Hoci teraz už vieme zapísať príkaz, ktorý na konzolu vypíše obsah nejakého textového súboru takto:

```
>>> print(open('readme.txt').read())
```

budeme to zapisovať korektnejšie:

```
>>> with open('readme.txt') as t:
        print(t.read())
```

alebo

```
>>> with open('readme.txt') as t: print(t.read())
```

Všimnite si, že sme pri `open()` nepoužili parameter `'r'` pre označenie otvorenia na čítanie. Keď totiž pri otváraní nezapíšeme `'r'`, Python si domyslí práve otváranie súboru na čítanie.

Ak očakávame, že otvorený súbor je príliš veľký a my ho naozaj nepotrebujeme vypísať celý, zapíšeme:

```
>>> with open('readme.txt') as t: print(t.read()[:1000])
```

Pridávanie riadkov do súboru

Videli sme dva rôzne typy otvárania textového súboru:

- `t = open('subor.txt', 'r')` - súbor sa otvoril na len čítanie, ak ešte neexistoval, program spadne
- `t = open('subor.txt', 'w')` - súbor sa otvoril na len zapisovanie, ak ešte neexistoval, tak sa vytvorí prázdny, inak sa zruší doterajší obsah (zapíše sa prázdny obsah)

Zoznámime sa s ešte jednou voľbou, pri ktorej sa súbor otvorí na zápis, ale nezruší sa jeho pôvodný obsah. Namiesto toho sa nové riadky budú pridávať na koniec súboru. Napríklad, ak máme súbor `'subor3.txt'` s tromi riadkami:

```
first line
second line
end of file
```

môžeme do neho pripísať ďalšie riadky, napríklad takto: namiesto `'r'` a `'w'` pri otváraní súboru použijeme `'a'`, ktoré označuje anglické slovo **append**:

```
t = open('subor3.txt', 'a')
t.write('pridany riadok na koniec\na este jeden\n')
t.close()
```

v súbore je teraz:

```
first line
second line
end of file
pridany riadok na koniec
a este jeden
```

Zrejme by sme to zvládli naprogramovať aj bez tejto voľby, len pomocou pôvodného čítania a zápisu, ale bolo by to časovo náročnejšie riešenie, napríklad takto:

```
with open('subor3.txt', 'r') as t:
    cely = t.read()                         # zapamätá si pôvodný obsah
with open('subor3.txt', 'w') as t:          # vymaže všetko
    t.write(cely)                           # vráti tam pôvodný obsah
```

```
    t.write('pridany riadok na koniec\na este jeden\n')
```

Zistite čo urobí:

```
for i in range(100):
    with open('subor4.txt', 'a') as file:
        print('vypisujem', i, 'riadok', file=file)
```

Uvedomte si, že takéto riešenie je veľmi neefektívne.

Príklad s grafikou

Máme pripravený textový súbor, v každom riadku ktorého sú dve celé čísla - súradnice nejakých bodov v grafickej ploche. Napríklad:

```
100 100
150 200
200 150
150 150
```

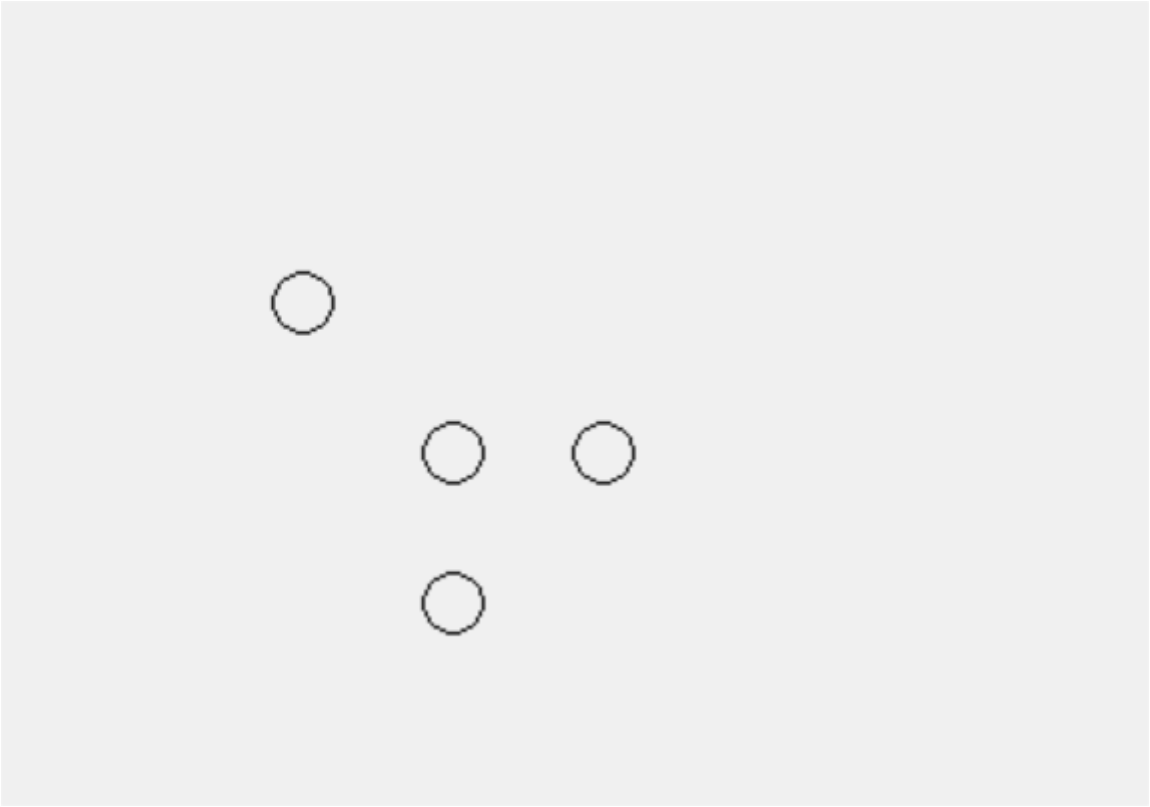
Napíšme program, ktorý prečíta súradnice týchto bodov a do grafickej plochy na príslušné miesta nakreslí malé krúžky:

```
import tkinter

canvas = tkinter.Canvas()
canvas.pack()

with open('body.txt') as subor:
    for riadok in subor:
        i = riadok.find(' ')
        x, y = int(riadok[:i]), int(riadok[i:])
        canvas.create_oval(x-10, y-10, x+10, y+10)

tkiner.mainloop()
```



Na cvičeniach budeme ďalej pracovať s touto ideou. Napríklad budeme tieto body spájať úsečkami, alebo budeme generovať vlastné textové súbory s nejakými kresbami (postupnosťami bodov).

Cvičenia

L.I.S.T.

- riešenia **aspoň 10 úloh** odovzdaj na úlohový server https://list.fmph.uniba.sk/
- používaj len konštrukcie z doterajších prednášok
- pozri si Riešenie úloh 7. cvičenia

V úlohách cvičenia môžeš využiť tieto textové súbory:

- súbor `'text1.txt'`:

```
Pocitac ma oproti ludskemu mozgu
len jednu prednost
- ze ho pouzivame.

Gabriel Laub
```

- súbor `'text2.txt'`:

```
Tri zakladne zakony robotiky.
```

```
- Po prve, robot nesmie zranit ludsku bytost alebo dovolit, aby sa jej v dosl
  edku necinnosti ublizilo.
- Po druhe, robot musi posluchat prikazy ludskych bytosti, okrem pripadov, ke
  d by taketo prikazy boli v rozpore s prvym zakonom.
- Po tretie, robot musi chranit svoju vlastnu existenciu, pokial takato ochra
  na nie je v rozpore s prvym a druhym zakonom
-
- Isaac Asimov
```

- súbor `'text3.txt'`:

```
- Ján Botto
- Žltá ľalija
-
- Stojí stojí mohyla
- Na mohyle zlá chvíľa
- na mohyle tŕnie chrastie
- a v tom tŕní chrastí rastie
- rastie kvety rozvíja
- jedna žltá ľalija
- Tá ľalija smutno vzdychá
- Hlávku moju tŕnie pichá
- a nožičky oheň páli
- pomôžte mi v mojom žiali
```

---

1. Napíš program, ktorý si vypýta meno súboru a z tohto súboru vypíše druhý riadok. Napríklad:

```
2. zadaj meno súboru: text1.txt
3. druhý riadok súboru: 'len jednu prednost'
```

2. Napíš program, ktorý si vypýta meno súboru a potom vypíše každý druhý znak z prvého riadka tohto súboru. Napríklad:

```
3. zadaj meno súboru: text1.txt
4. každý druhý znak: 'oia aort useumzu'

5. zadaj meno súboru: text2.txt
6. každý druhý znak: 'r aldezkn ooiy'
```

3. Napíš program, ktorý si vypýta meno súboru a potom vypíše počet riadkov a dĺžku najdlhšieho riadka (aj s koncovým `'\n'`). Napríklad:

```
4. meno súboru: text2.txt
5. počet riadkov súboru: 6
6. najdlhší riadok má 127 znakov
```

```
meno súboru: text3.txt
počet riadkov súboru: 13
najdlhší riadok má 28 znakov
```

Zapíš tri rôzne riešenia tejto úlohy:

a. čítaním pomocou `readline()` a while-cyklu
b. čítaním riadkov pomocou for-cyklu
c. prečítaním naraz celého súboru pomocou `read()` a potom kontrolovaním pozícií znaku `'\n'`

4. Napíš program, ktorý si vypýta meno súboru a potom vypíše všetky tie riadky súboru, ktoré obsahujú práve tri slová. V každom riadku je niekoľko slov, ktoré sú oddelené jednou medzerou. Napríklad:

```
meno súboru: text3.txt
riadky s tromi slovami:
Stojí stojí mohyla
rastie kvety rozvíja
jedna žltá ľalija
```

Uvedom si, že stačí v každom riadku počítať počet medzier.

5. Napíš program, ktorý si vypýta meno súboru a potom z každého riadku, ktorý obsahuje aspoň tri slová, vypíše druhé slovo. Napríklad:

```
meno súboru: text2.txt
druhé slová:
zakladne
prve,
druhe,
tretie,
```

6. Napíš funkciu `priemer(meno_suboru)`, ktorá otvorí a číta zadaný súbor. V každom riadku tohto súboru je jedno celé číslo. Funkcia zistí priemer týchto hodnôt. Napríklad, pre takýto súbor `'cisla.txt'`:

```
554
-8
27
4448
7
92
144
```

dostaneš:

```
>>> print('priemer =', priemer('cisla.txt'))
priemer = 752.0
```

7. Napíš funkciu `riadky_s_textom(meno_suboru, text)`, ktorá otvorí zadaný súbor (môže byť aj súbor s pythonovským programom) a vypíše z neho len tie riadky, ktoré obsahujú zadaný text. Napríklad:

```
>>> riadky_s_textom('riesenie.py', 'if ')
if a != b:
    serif = 1517
elif b < 7:
    if x:

>>> riadky_s_textom('text3.txt', 'ali')
Žltá ľalija
jedna žltá ľalija
Tá ľalija smutno vzdychá
pomôžte mi v mojom žiali
```

8. Napíš a otestuj tieto funkcie. Ich parametrom je meno nejakého textového súboru a všetky vrátia (`return`) nejaký jeden riadok súboru:
   - funkcia `posledny_riadok(meno_suboru)`
   - funkcia `predposledny_riadok(meno_suboru)`
   - funkcia `najdlhsi_riadok(meno_suboru)`

   Napríklad:

```
>>> posledny_riadok('text3.txt')
    'pomôžte mi v mojom žiali'
>>> predposledny_riadok('text3.txt')
    'a nožičky oheň páli'
>>> najdlhsi_riadok('text3.txt')
    'a v tom tŕní chrastí rastie'
```

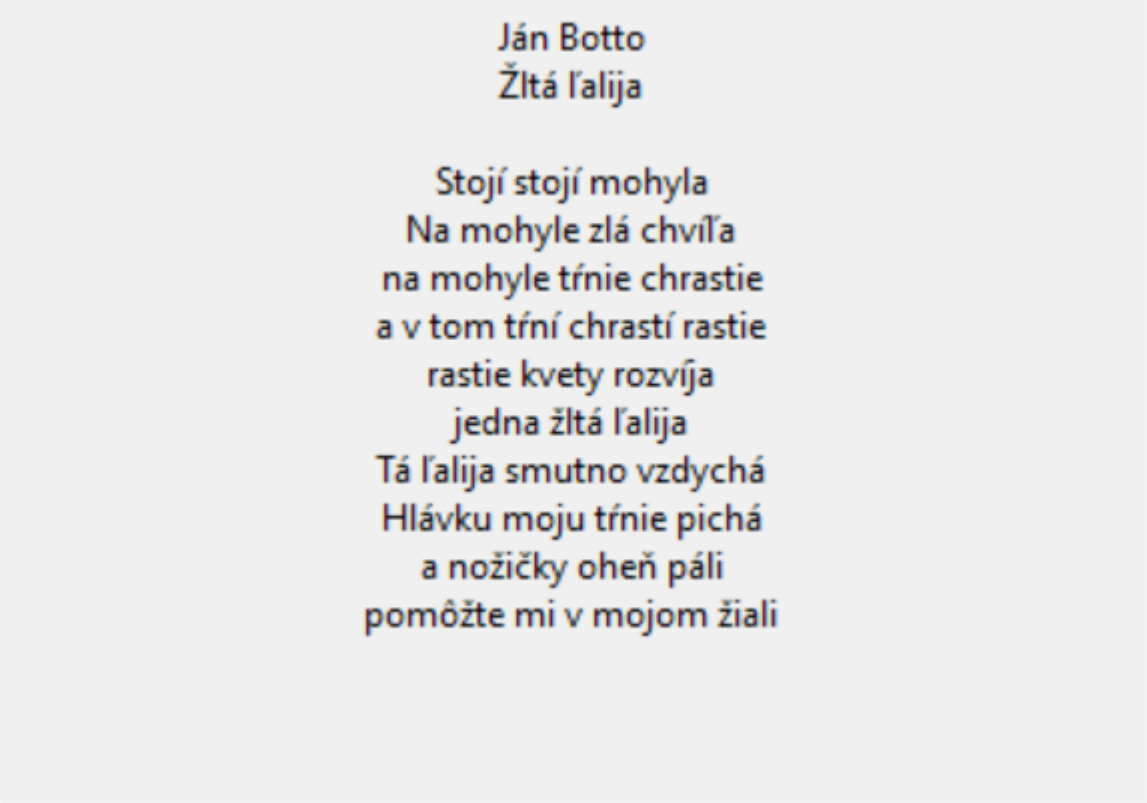
9. Napíš funkciu `ity(meno_suboru, index)`, ktorá z daného súboru vráti (`return`) riadok s daným poradovým číslom (`index`). Riadky číslujeme od `0`. Napríklad:

```
>>> ity('text1.txt', 2)
    '- ze ho pouzivame.'
>>> ity('text2.txt', 0)
    'Tri zakladne zakony robotiky.'
>>> ity('text3.txt', 3)
    'Stojí stojí mohyla'
```

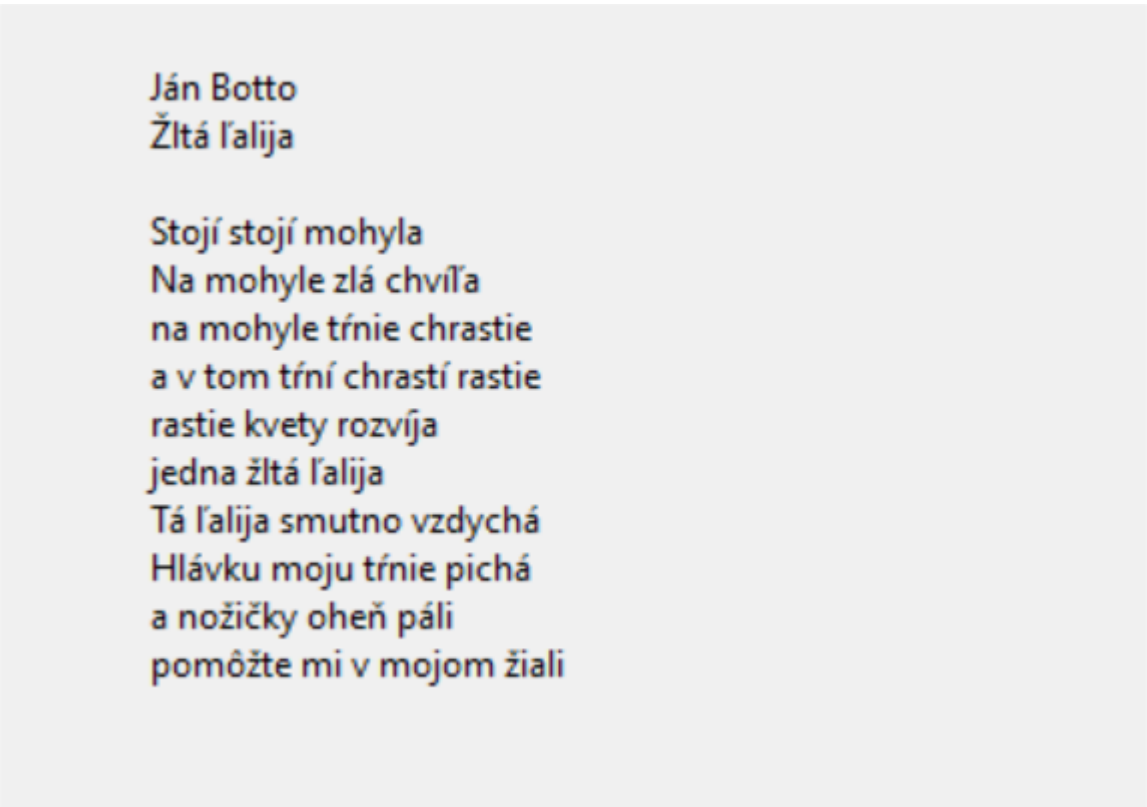
10. Vypíš obsah textového súboru do grafickej plochy. Súbor obsahuje niekoľko riadkov a funkcia `vypis_subor(meno_suboru)` tieto riadky vypíše pod sebou nejakým fontom. V globálnej premennej `canvas` je referencia na grafickú plochu. Napríklad:

```
>>> vypis_subor('text3.txt')
```

vypíše do grafickej plochy:



Pre zarovnanie vypisovaného textu pomocou `create_text` nie na stred ale na ľavý okraj môžeš použiť ďalší pomenovaný parameter `anchor='nw'`, potom by si mal dostať takýto výpis:



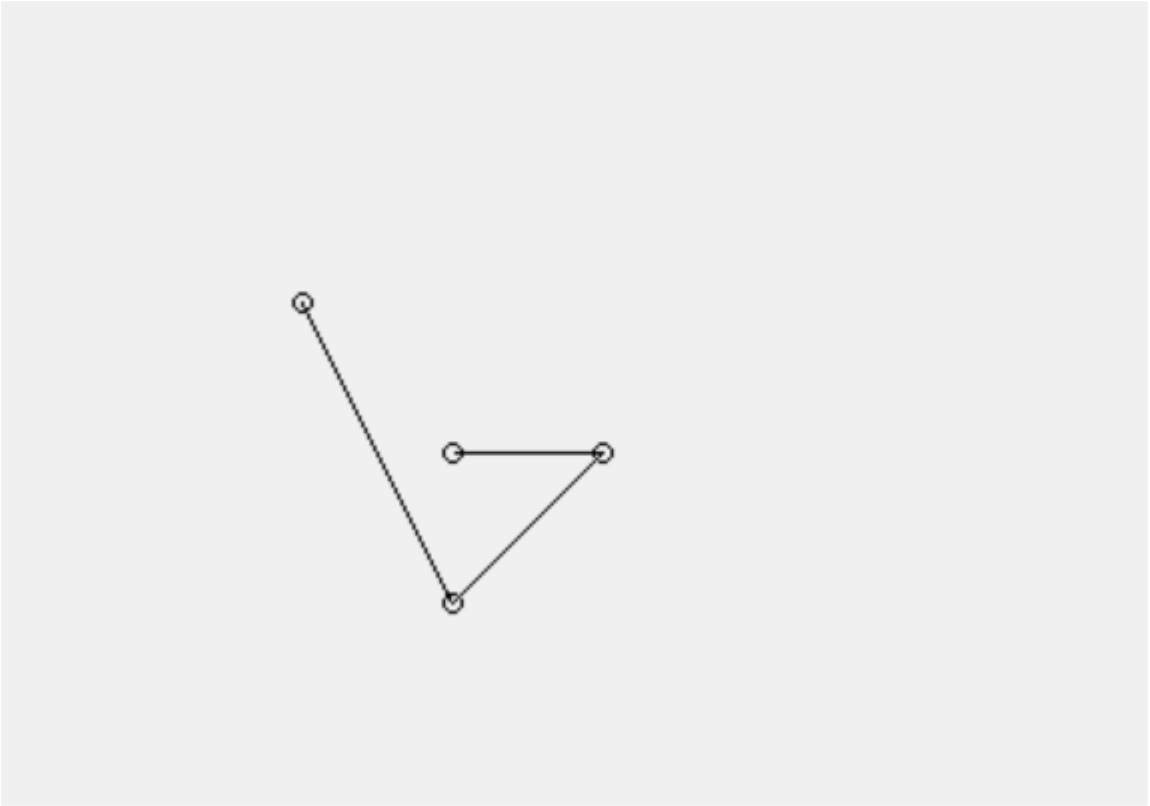
11. Na prednáške bol program, ktorý z textového súboru čítal súradnice bodov `x`, `y` a do grafickej plochy na príslušných miestach kreslil malé krúžky. Zapíš to ako funkciu `kresli(meno_suboru)`, ktorá namiesto kreslenia krúžkov, bude postupne spájať body zo súboru, t.j. najprv sa nakreslí úsečka z prvého bodu do druhého, potom z druhého do tretieho, … Napríklad, pre súbor `'body.txt'` z prednášky:

```
100 100
150 200
200 150
150 150
```

volanie:

```
>>> kresli('body.txt')
```

nakreslí tieto tri úsečky:



12. Funkciu `kresli()` z predchádzajúceho príkladu vylepši takto: ak sa vo vstupnom súbore nachádza prázdny riadok, tento označuje, že za ním nasleduje ďalšia skupina bodov, ktorá ale nie je s predchádzajúcimi bodmi spojená. Napríklad, pre súbor `'body1.txt'`:

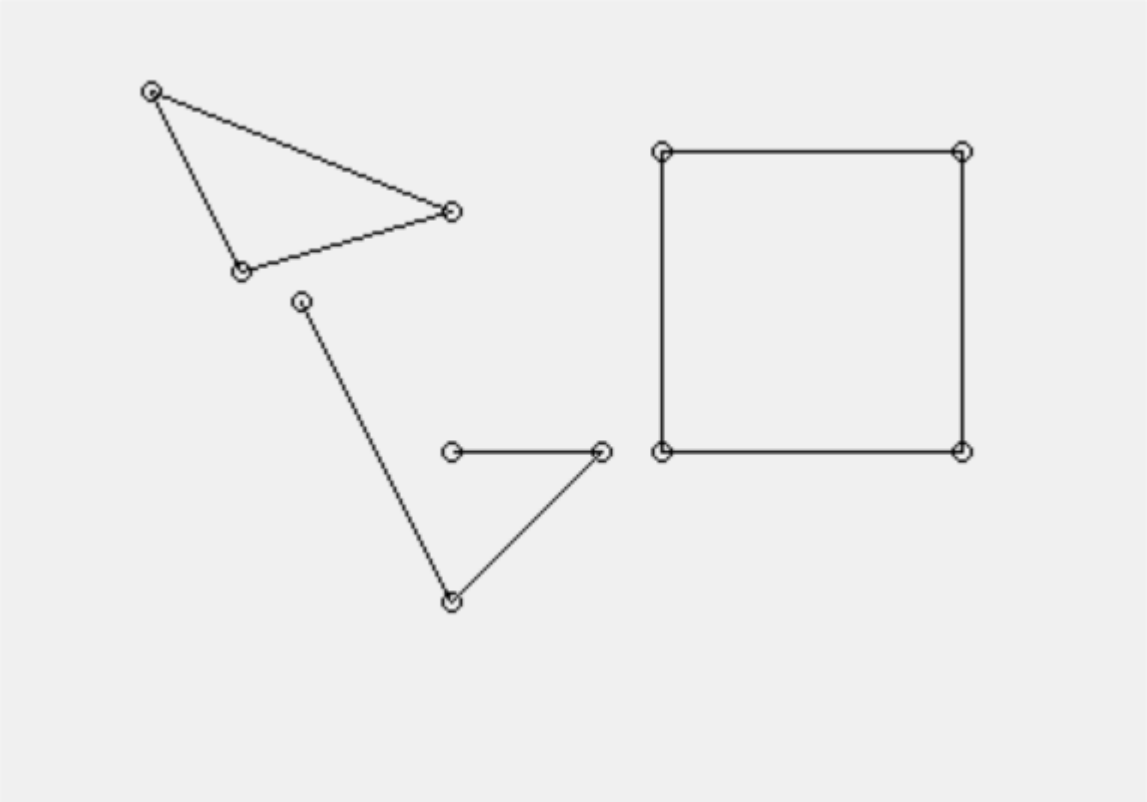
```
100 100
150 200
200 150
150 150

```

```
220 50
320 50
320 150
220 150
220 50

50 30
150 70
80 90
50 30
```

nakreslí:



13. Textový súbor obsahuje v každom riadku jedno meno farby. Napíš funkciu `do_riadkov(meno_suboru, sirka)`, ktorá vypíše rad štvorčekov (veľkosti 30x30). Každý z týchto štvorčekov zafarbí príslušnou farbou zo súboru. Po vykreslení `sirka` štvorčekov, pokračuje pod týmito v ďalšom rade štvorčekov s ďalšími farbami zo súboru. Takto pokračuje, kým sa neminú všetky farby zo súboru. Napríklad, pre súbor `'farby.txt'`:

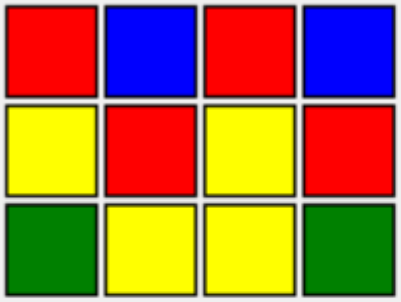
```
red
blue
red
blue
yellow
red
yellow
red
```

```
green
yellow
yellow
green
```

volanie:

```
import tkinter
canvas = ...
do_riadkov('farby.txt', 4)
```

nakreslí:



Ak by sme tento súbor vypisovali do 5 stĺpcov, dostali by sme:

14. Napíš funkciu `nahodne_cisla(meno_suboru, pocet)`. Funkcia vytvorí textový súbor s trojcifernými náhodnými číslami (zrejme z intervalu `<100, 999>`). V každom riadku bude jedno číslo, riadkov bude zadaný počet. Napríklad volanie:

```
>>> nahodne_cisla('cisla1.txt', 5)
```

Vytvorí päťriadkový súbor `'cisla1.txt'`, ktorý môže obsahovať:

```
272
598
822
927
233
```

Otestuj takto vytvorený súbor s tvojou funkciu, ktorá počíta priemer, napríklad:

```
>>> nahodne_cisla('cisla2.txt', 100)
>>> print('priemer =', priemer('cisla2.txt'))
priemer = 540.58
```

15. Napíš funkciu `vyrob(meno_suboru, pocet, text)`, ktorá vytvorí textový súbor s daným menom. Tento súbor bude pythonovským skriptom, ktorý príslušný `pocet` krát vypíše (pomocou `print()`) zadaný text. Tento skript by mal na opakovanie výpisu využiť while-cyklus (nie for-cyklus). Napríklad:

```
>>> vyrob('skript.py', 20, 'Programujem v Pythone')
```

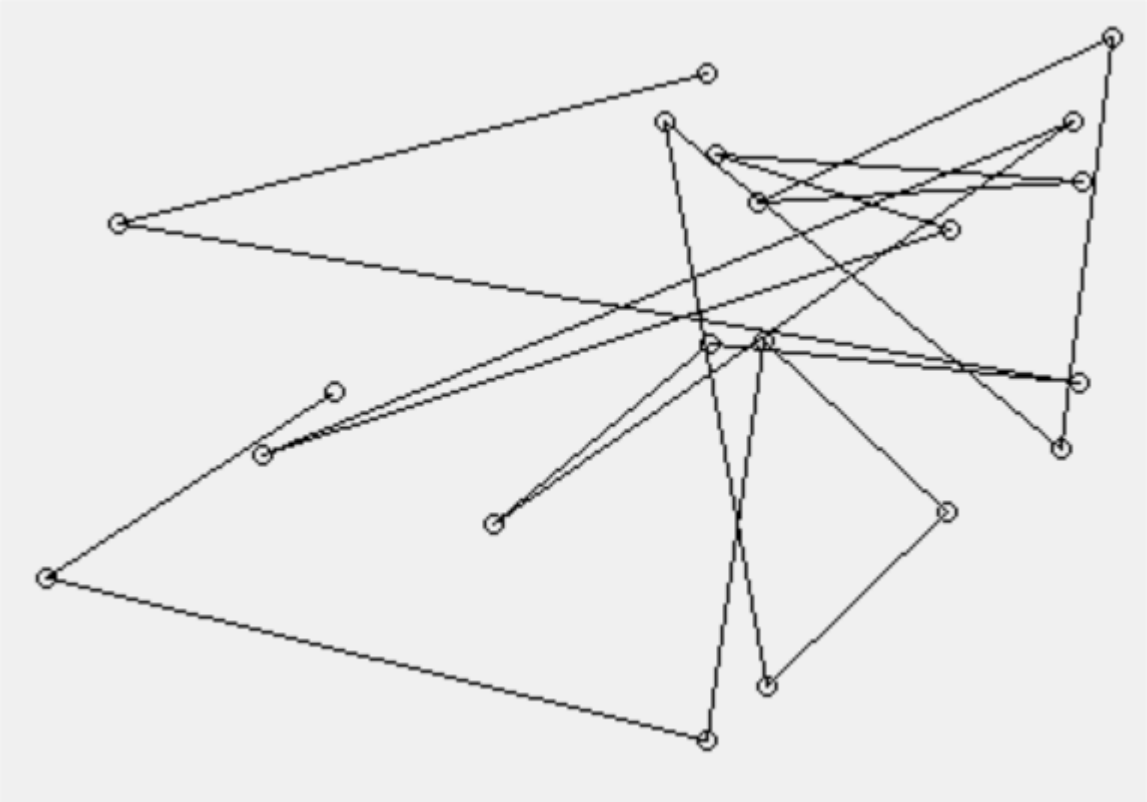
vytvorí nový program `'skript.py'` a ten po spustení, výpíše 20-krát pod seba:

```
Programujem v Pythone
Programujem v Pythone
...
```

16. Napíš funkciu `nahodne_body(meno_suboru, pocet)`, ktorá vygeneruje súbor s daným menom. Tento bude obsahovať zadaný `pocet` riadkov, pričom v každom bude náhodná dvojica celých čísel oddelená medzerou. Prvé číslo nech je z intervalu `<10, 370>` a druhé z intervalu `<10, 250>`. Takto vytvorený súbor by sa mohol využiť vo funkcii `kresli` z (11) úlohy, napríklad:

```
>>> nahodne_body('body3.txt', 20)
>>> kresli('body3.txt')
```

nakreslí a spojí nejakých 20 náhodných bodov, dostaneš niečo podobné:

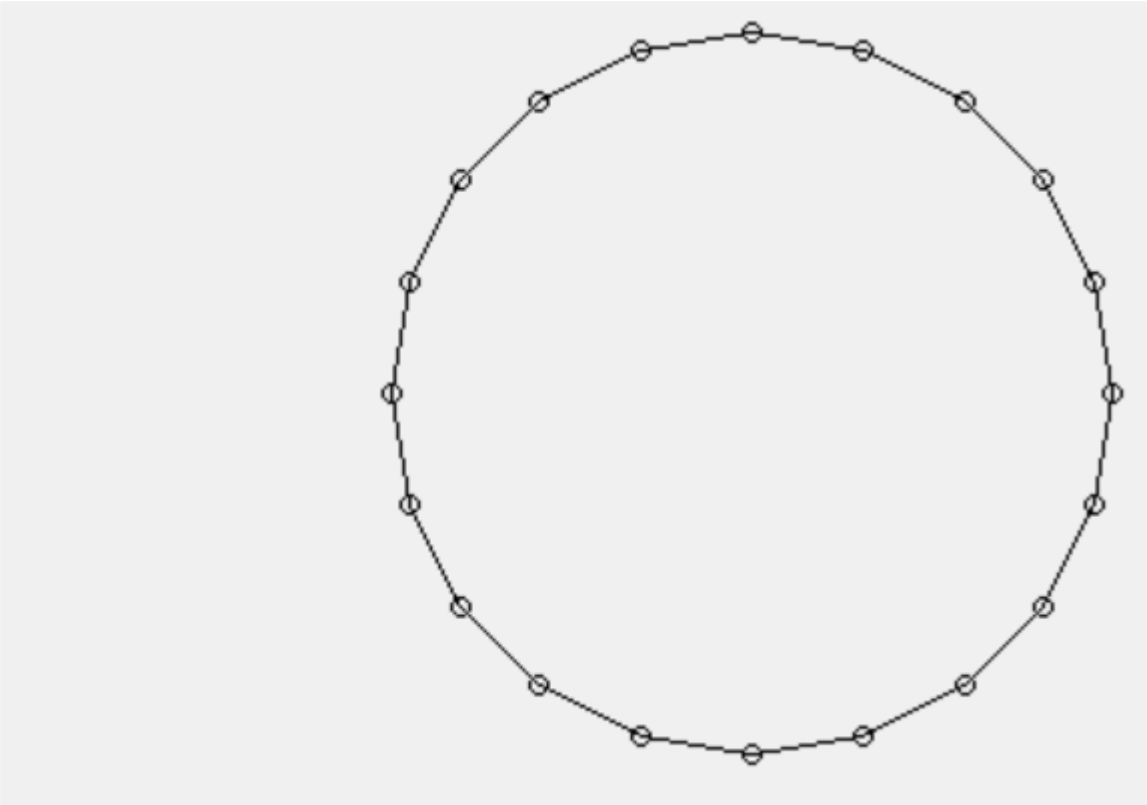


17. Napíš funkciu `body_na_kruznici(meno_suboru, n, r, x, y)`, ktorá vygeneruje súbor s daným menom. Tento bude obsahovať `n+1` riadkov, pričom v každom budú súradnice nejakého bodu ako dvojica **celých čísel** oddelená medzerou. Súbor bude obsahovať body pravidelného `n`-uholníka, ktoré sú rozložené na kružnici s polomerom `r` a so stredom v `(x, y)`. Zrejme prvý a posledný (`n+1`) bod budú rovnaké, aby sa pri kreslení

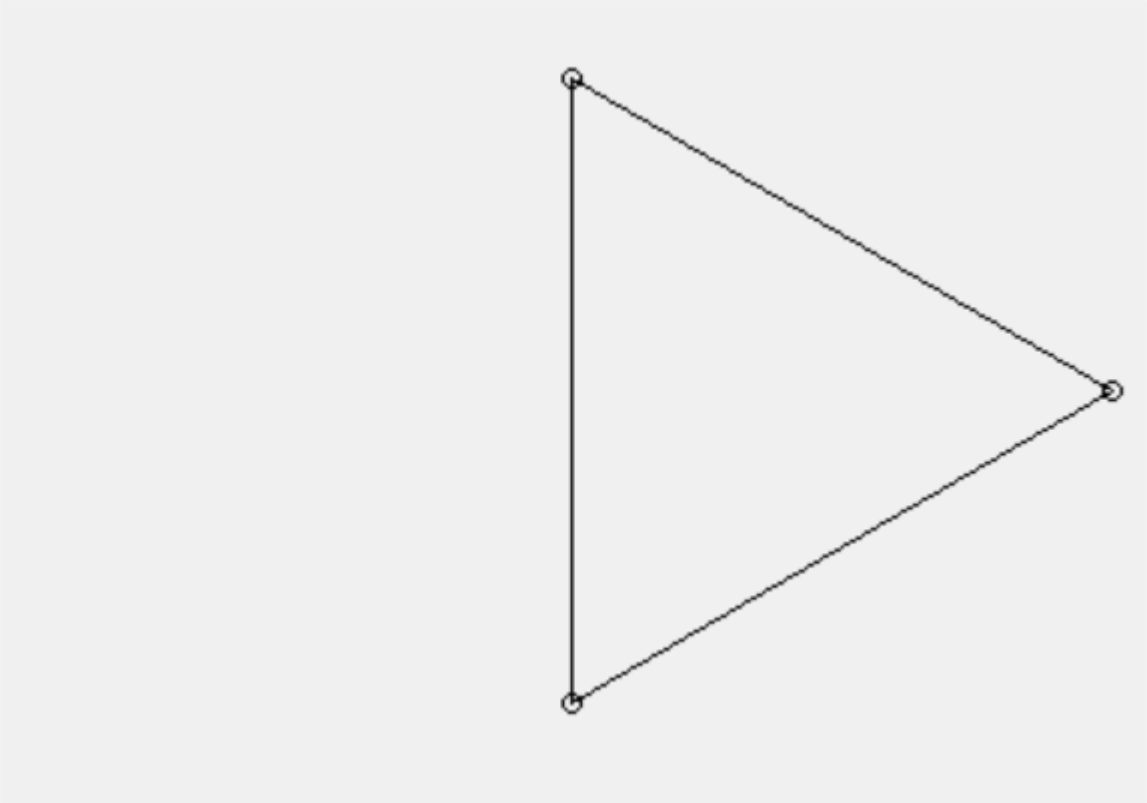
tento `n`-uholník uzavrel. Takto vytvorený súbor by sa mohol využiť vo funkcii `kresli` z (10) úlohy, napríklad:

```
18. >>> body_na_kruznici('body4.txt', 20, 120, 250, 130)
19. >>> kresli('body4.txt')
```

nakreslí takýto 20-uholník:



to isté pre trojuholník:



18. Napíš funkciu `pridaj(meno_suboru, text)`, ktorá do textového súboru **pridá na koniec** nový riadok so zadaným textom. Napríklad, ak súbor obsahoval riadky:

```
prvý riadok
druhý riadok
```

potom volania:

```
>>> pridaj('subor.txt', 'predposledný')
>>> pridaj('subor.txt', 'posledný riadok')
```

zmenia tento súbor:

```
prvý riadok
druhý riadok
predposledný
posledný riadok
```

19. Napíš funkciu `vyhod_riadok(meno_suboru, index)`, ktorá z textového súboru odstráni zadaný riadok (`index` je číslo riadka od 0). Ak sa `index` rovná **-1**, funkcia vyhodí posledný riadok. Ak riadok so zadaným indexom neexistuje, funkcia nerobí nič. Napríklad, pre 4-riadkový `'subor.txt'` z predchádzajúceho príkladu, volanie:

```
>>> vyhod_riadok('subor.txt', 1)
```

odstráni riadok s indexom 1, teda druhý v poradí:

```
prvý riadok
predposledný
posledný riadok
```