

# Programujeme v Pythone

učebnica informatiky pre stredné školy

Peter Kučera, Jaroslav Výbošťok

2

The collage consists of several windows from Python applications:

- A terminal window showing a grid of numbers.
- A Tkinter window titled "tk" containing a circle with a triangle inscribed in it, labeled with  $r$ ,  $\sin(u)*r$ , and  $\cos(u)*r$ . To its right is a green bell-shaped curve.
- A bar chart titled "Hlasovalo: 41948 ľudí." showing the distribution of votes across different categories.
- A dice rolling simulation showing three green dice and one red die.
- A weather forecast map of Slovakia with icons and temperature ranges like 24/28, 26/30, and 30/34.
- A Tkinter window titled "score.txt - Poznámkový blok" displaying a high-score table:

High score	=====
Alena	120
Bob	115
Charles	80
Dana	75
Edo	70

- A Tkinter window titled "tk" showing a 3D wireframe tetrahedron.
- A terminal window showing a list of numbers.
- A row of ten cartoon frogs numbered 1 through 10.
- A large watermark at the bottom right: [www.programujemevpython.sk](http://www.programujemevpython.sk)



## **Programujeme v Pythone 2**

učebnica informatiky pre stredné školy

Autori © Mgr. Peter Kučera, Mgr. Jaroslav Výbošťok

Design © Mgr. Peter Kučera

Jazyková korektúra: Mgr. Katarína Kučerová

Prvé vydanie, 2017

Verzia číslo: 20170927

Vydavateľ: Mgr. Peter Kučera

### **Údaje o nadobúdateľovi licencie:**

Názov/meno: SRRZ-Rodičovské združenie pri GYMNÁZIU a SOU strojárskom

E-mail: sobekjozef@gmail.com

Mesto/obec: Slovenská 5, 085 01 Bardejov

Licencia číslo: 7011710017 - licencia pre jedného používateľa

**Upozorňujeme, že elektronická kniha je dielom chráneným podľa autorského zákona a je určená len pre osobnú potrebu kupujúceho. Kniha ako celok ani žiadna jej časť nesmie byť voľne šírená na internete, ani nijako ďalej zverejňovaná. V prípade ďalšieho šírenia neoprávnene zasiahnete do autorského práva s dôsledkami podľa platného autorského zákona a trestného zákonníka.**

**Velmi si vážime, že e-knihu ďalej nešírite. Len vďaka vašim nákupom dostanú autori, vydavatelia a kníhkupci odmenu za svoju prácu. Ďakujeme, že tak prispievate k vzniku ďalších skvelých kníh.**

učebnicu a ďalšie materiály si môžete zakúpiť aj priamo na stránkach autora:  
<http://www.programujemevpython.sk/> a <https://www.facebook.com/programujemevpython>

ISBN 978-80-972779-1-8 (pdf)

ISBN 978-80-972779-2-5 (epub)

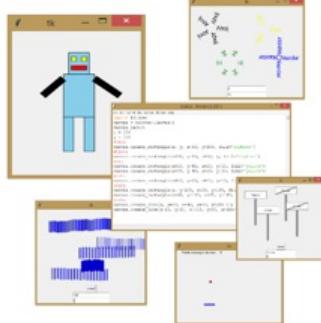
ISBN 978-80-972779-3-2 (mobi)

## Z NAŠEJ PONUKY

### Programujeme v Pythone

učebnica informatiky pre stredné školy

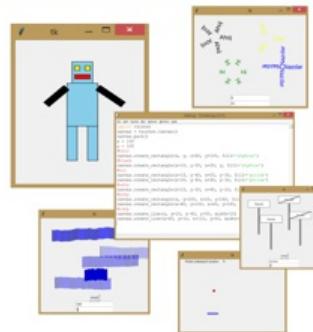
Peter Kučera



### Príručka pre učiteľa

k učebnici Programujeme v Pythone

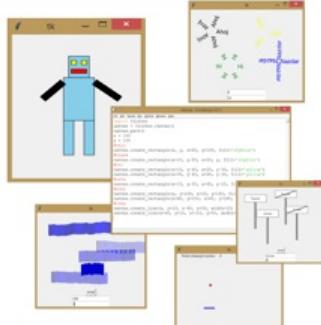
Peter Kučera



### Testy k učebnici

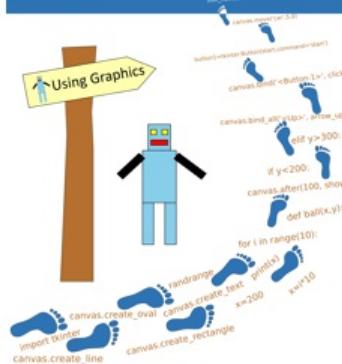
Programujeme v Pythone

Peter Kučera



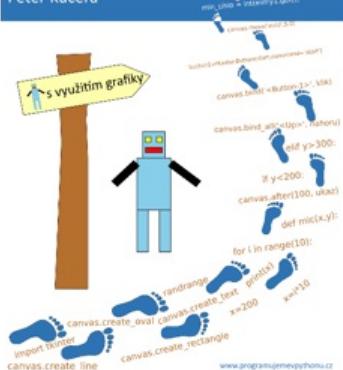
### Creating with Python

Peter Kučera



### Programujeme v Pythonu

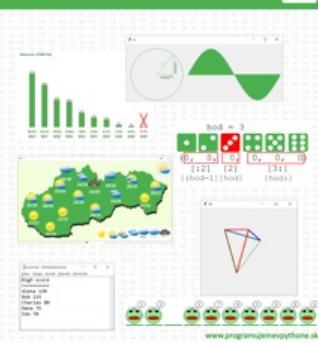
Peter Kučera



### Programujeme v Pythone

učebnica informatiky pre stredné školy

2



[www.programujemevpythonu.cz](http://www.programujemevpythonu.cz)

[www.facebook.com/programujemevpython](https://www.facebook.com/programujemevpython)

# Napísali o našich e-knihách...

Učebnica Programujeme v Pythone (učebnica pre stredné školy) si rýchlo vydobyla miesto vo vyučovaní programovania naprieč celým Slovenskom vďaka jednoduchému prístupu k celej škále programovacích prostriedkov zavedených prostredníctvom grafického prostredia tkinter. Tento prístup podporuje vizualizáciu výsledkov v interaktívnom alebo programovom režime pochopenie programovacích techník, priam vyzýva študentov na experimentovanie pri modifikácii množstva gradovaných nematematických úloh a umožňuje individuálne tempo v štúdiu rovnako dievčatám aj chlapcom, poskytuje študentom radosť z nadobúdania vedomostí. Vrelo odporúčam všetkým záujemcom o vniknutie do tajov programovania vo veku od 9 do 99 rokov, nielen stredoškolákom. Prvýkrát vo svojej dlhorocnej praxi učiteľa informatiky som sa stretla s metodickou príručkou k učebnici – pomôže nielen učiteľom, ale aj samoukom.

RNDr. Eva Hanulová,  
Gymnázium Jura Hronca, Bratislava

Tešíme sa z každej novej učebnice, a tých vhodných na vyučovanie programovania na strednej škole je ako Šafránu. Preto som sa jej veľmi potešila, no nielen ja, ale aj moji študenti. Veľmi oceňujem aj príručku pre učiteľa a testy, ktoré sú úplne perfektné - sú to ďalšie úlohy a námety na vyučovanie. Na druhý diel sa tešíme odvtedy, odkedy som sa dozvedela, že ho autori pišu.

RNDr. Eva Stanková,  
Gymnázium Ivana Horvátha, Bratislava

Veľmi oceňujem učebnicu Programujeme v Pythone. Ponúka študentom prehľadným spôsobom dostať sa do programovania. Na hodinách môžu študenti s učebnicou pracovať aj samostatne vlastným tempom. Veľkým prínosom je aj príručka pre učiteľa, kde je prehľadne spracovaná celá metodika cez vzdelávacie plány až po prehľadne spracované učivo s riešenými úlohami a praktickými metodickými poznámkami ku každej úlohe. Hodnotný je aj súbor testov.

RNDr. Ol'ga Poliaková, SPŠ, Bardejov

Informatika je v porovnaní s inými predmetmi predmetom, ktorého veľká časť učiva sa z roka na rok mení dosť výrazným tempom, čo vyvíja na učiteľa tlak venovať veľkú časť prípravy samoštúdiu. Učiteľ informatiky na základe "podpory" štátu (chýbajúce, resp. neaktuálne učebnice, starý hardvér, ...) v takejto situácii zistuje, že informatika v tejto spoločnosti nepatrí medzi klúčové predmety. Učebnica mi ušetrila množstvo času a energie, ktorú som mohol venovať iným aktivitám v škole. Oceňujem príručku pre učiteľa s vyriešenými zadaniami z učebnice. Veľakrát som našiel v príručke iné riešenia ako tie moje, mohol som ich predstaviť žiakom, a tým ich myslenie, uvažovanie posunúť do iného, ďalšieho levelu. Myslím, že vďaka učebnici sa skvalitnila výučba informatiky na našej škole.

Mgr. Peter Nemeč,  
Spojená škola sv. Vincenta de Paul, Bratislava

Učebnica Programujeme v Pythone je výbornou pomôckou pre učiteľov informatiky pri výučbe programovania. Metodika použitia grafických úloh v jazyku Python je vhodná na názorné osvojenie základov programovania. Prínosom učebnice je metodická príručka pre učiteľa a súbor testov. Učebnicu použijeme pri príprave budúcich učiteľov informatiky, ktorí ju môžu overiť v rámci svojej pedagogickej praxe na cvičných školách.

Ing. Janka Majherová, PhD., Katedra informatiky,  
Pedagogická fakulta, Katolícka univerzita v Ružomberku

# Obsah

## Obsah

### Úvod

#### 1 Textové reťazce

- 1.1 Čísla vs. textové reťazce, načítanie vstupu
- 1.2 Prechádzanie znakmi reťazca, konštruovanie nového reťazca
- 1.3 Podreťazce, rezy
- 1.4 Znaky a ich kódy
- 1.5 Pracujeme s textovými reťazcami
  - Cézarova šifra
  - Náhodné zamiešanie znakov v textovom reťazci
- 1.6 Logické operácie a textové reťazce
  - Operácia in a pravdivostné hodnoty (boolean)
- 1.7 Ďalšie užitočné funkcie na prácu s textovými reťazcami
  - Niekteré metódy textových reťazcov
- 1.8 Formátovanie reťazcov

#### 2 N-tice (tuple)

- 2.1 N-tice textových reťazcov - farby a slová
- 2.2 Body v rovine
- 2.3 N-tice ako parameter
- 2.4 Viacnásobné priradenie

#### 3 Textové súbory

- 3.1 Zápis do textového súboru
- 3.2 Pridávanie riadkov do textového súboru
- 3.3 Čítanie z textového súboru
- 3.4 Iné spôsoby čítania textového súboru
  - Cyklus s podmienkou (while cyklus)
  - Konštrukcia with
- 3.5 Práca s viacerými textovými súbormi

#### 4 Funkcie s návratovou hodnotou

#### 5 Práca s viacerými údajmi (zoznam)

- 5.1 Hádzanie hráčimi kockami
  - Spoločné vlastnosti textových reťazcov, n-tíc a zoznamu
- 5.2 Užitočné funkcie a metódy na prácu so zoznamom
  - Vizualizácia údajov programu
- 5.3 Rozdelenie textového reťazca do zoznamu
- 5.4 Využívame zoznam v programoch
  - Need for Speed
  - Vyhľadávanie prvku s požadovanými vlastnosťami
  - SMS hlasovanie, metódā sort
  - Žaby, vymieňanie prvkov v zozname

#### 6 Obrázky

- 6.1 Načítavanie a kreslenie obrázkov gif a png
- 6.2 Zoznam obrázkov
- 6.3 Vizualizácia predpovede počasia
  - Lambda funkcie

#### 7 Matematické výpočty a geometria

#### 8 Asociatívne polia (slovník - dictionary)

- 8.1 Vytvorenie slovníka, metódy na prácu so slovníkom
  - Šifrovanie náhodnou substitúciou
- 8.2 Frekvencia výskytov
  - Frekvencia znakov
  - Frekvencia slov v slovných hodnoteniach
- 8.3 Zoznam asociatívnych polí

#### 9 Vlastnosti útvarov nakreslených v canvase

9.1 Zisťovanie a zmena nastavení útvarov

9.2 Značky útvarov a ich využitie na pamätanie si informácie

## 10 Upravujeme vzhľad aplikácií

Metódy zobrazenia widgetov a ich umiestnenie

Listbox

Posúvač

Textová plocha

Radiobutton

Checkbutton

10.1 Menu

10.2 Dialógové okná

## 11 Úlohy na opakovanie

Príkazovník

Použitá literatúra

# Úvod

Sme radi, že elektronická učebnica informatiky pre stredné školy - Programujeme v Pythone si našla svojich priaznivcov. Prvý diel bol základným kurzom programovania, určeným všetkým študentom strednej školy, a napíňal Štátny vzdelávací program. Druhý diel je plynulým pokračovaním nielen pre študentov, ktorí sa chcú venovať programovaniu hlbšie alebo sa pripravujú na maturitu z informatiky, ale aj pre ďalších záujemcov či samoukov.

Učebnica spĺňa požiadavky na maturitu z informatiky podľa Cieľových požiadaviek na vedomosti a zručnosti maturantov z informatiky pre 1. oblasť s názvom Algoritmické riešenie problémov, ktoré sú platné od školského roka 2018/2019, ale aj v súčasnosti. Okrem tém (kapitol), ktoré sú priamo potrebné na maturitu, nájdete v učebnici aj témy, ktoré tento obsah dopĺňajú a pomôžu vám porozumieť dôležitým súvislostiam v Pythone, primerane na maturitnej úrovni (resp. na úrovni pre mierne pokročilých). Tieto témy vám pomôžu tvoriť komplexnejšie a zaujímavejšie programy.

V učebnici nájdete množstvo praktických a riešených príkladov, úlohy na precvičenie, ale aj otázky, ktoré vás majú nabádať na premýšľanie, objavovanie súvislostí, diskusiu v skupine, experimentovanie, ale aj hľadanie chýb a intuitívne hľadanie optimálneho riešenia.

Desiata kapitola je zameraná na upravovanie vzhľadu aplikácií - na ovládacie prvky z knižnice tkinter. Obsahuje praktické ukážky na použitie widgetov (ovládacích prvkov), nie je však nutné ovládať všetky tieto poznatky, skôr je to rýchla pomôcka na ich používanie. Aj preto táto kapitola neobsahuje úlohy a otázky.

Po preštudovaní si tejto učebnice a hodinách strávených praktickým programovaním by ste mali zvládnuť riešiť aj stredne náročné úlohy s použitím zložených dátových údajov a tiež by ste mali zvládnuť naprogramovať komplexnejší program väčšieho rozsahu, napríklad seminárnu prácu.

Aj keď v skutočnosti je táto e-kniha len postupnosťou núl a jednotiek, veríme, že v tejto postupnosti prevážia tie zaujímavé veci a učebnica sa stane vašou jednotkou pri štúdiu programovania.

Prajeme vám príjemné chvíle pri programovaní :)  
autori

Textové súbory a obrázky k úlohám si môžete stiahnuť z  
<http://www.programujemevpython.sk/download/pvp2-subory.zip>

# 1 Textové reťazce

## 1.1 Čísla vs. textové reťazce, načítanie vstupu

V minulosti sme už pracovali so znakovými reťazcami a tvorili sme náhodné vety. V tejto kapitole sa pozrieme na textové reťazce podrobnejšie. Zatiaľ vieme:

- spájať reťazce operáciou `+`

```
slovo = 'Py'+'thon'  
print(slovo)
```

- používať funkciu `int()`, ktorá z reťazca vytvorí číslo (ak obsahuje len znaky, ktoré môžu obsahovať číslo). Používali sme ju pri zadávaní vstupu z `entry`, napríklad `od = int(entry1.get())`.

```
a = '1250'  
b = int(a)  
b = b+10  
print(b)
```

Ak sme chceli vypísať `canvas.create_text(100, 200, text = 'Počet bodov: '+body)`, museli sme `body` (číslo) premeniť na text. Zatiaľ sme to vedeli vyriešiť tým, že sme zvlášť jedným príkazom vypísali text a ďalším príkazom číselnú hodnotu.

V takýchto prípadoch môžeme použiť funkciu `str()`, ktorá vytvorí z čísla textový reťazec.

Čiže `canvas.create_text(100, 200, text = 'Počet bodov: '+str(body))`.

```
body = 12  
oznam = 'Tvoj počet bodov: '+str(body)  
print(oznam)
```

### Otázky:

1. Podľa čoho spoznáme, či je v premennej číslo alebo znakový reťazec?
2. Čo sa stane, ak použijeme operáciu `+` na text a číslo zároveň?
3. Čo sa stane, ak použijeme operáciu `*` na text a číslo zároveň? (Napríklad: `s = 'abc'*4`)

Ak chceme presne zistiť, či si v premennej pamäťame číslo alebo znakový reťazec, použijeme funkciu `type(meno_premennej)`. Funkcia zistí typ informácie, ktorú si pamäťame v premennej, čiže dátový typ.

```
>>> a = 12  
>>> type(a)  
<class 'int'>  
>>>
```

Vidíme, že číslo je dátového typu `int`, čo je skratka anglického `integer`, čiže celé číslo.

```
>>> s = 'Python'  
>>> type(s)  
<class 'str'>  
>>>
```

Premenná `s` je dátového typu `str`, čo je skratka anglického `string`, čiže reťazec.

### Otázky:

4. Do premennej oznam sme priradili: oznam = ''. Akého dátového typu bude premenná oznam?
5. Vykonali sme tieto príkazy: pocet = '12' vysledok = pocet\*5. Čo bude v premennej vysledok a akého dátového typu bude premenná vysledok?
6. Vykonali sme tieto príkazy: pocet = '12' vysledok = pocet\*5. Čo sa zmení, ak by sme pocet\*5 nahradili napr. pocet\*-1 alebo pocet\*-5?

Doposiaľ sme väčšinu úloh riešili v grafickom prostredí knižnice **tkinter**. Vstup sme načítali pomocou súčiastky (widgetu) **entry**. Mnohé programy ale nepotrebujú okno s grafickým rozhraním, postačuje im textový režim (okno shell). S takým programom sme sa už stretli v minulosti pri tvorení náhodných viet. V textovom režime môžeme programu zadávať vstup (od používateľa programu) funkciou **input**. Funkcia **input** má jeden parameter - v ňom je napísaný text, ktorý sa zobrazí používateľovi, a program čaká na zadanie vstupu, zadávanie ukončíme stlačením klávesu enter.

```
#program rozhovor
from random import *
meno = input('Ako sa voláš?')
print('Ahoj '+meno+' rád Ťa spoznávam :))')
roknarodenia = input(meno+', v ktorom roku si sa narodil?')
meno2 = choice(['Alena', 'Barbora', 'Eva', 'Sofia'])
print('Á spomínam si, v roku '+roknarodenia+' sa narodila aj '+meno2)
```

### Otázky:

7. Čo bude robiť predchádzajúci program?
8. Akého dátového typu sú premenné: meno, roknarodenia a meno2?

### Úlohy:

- 1 Upravte program **rozhovor** tak, aby nám počítač vypočítal a napísal náš vek (pričízne, keďže nezadávame presný dátum narodenia).
- 2 Doplňte do programu **rozhovor** ďalšie otázky a odpovede.
- 3 Vytvorte program, ktorý vyzve používateľa na zadanie svojho mena a veku. Program následne používateľa pozdraví, napr. Ahoj Filip a vypíše, koľko rokov mu chýba do dovršenia 100 rokov.
- 4 Vytvorte program, v ktorom používateľ zadá svoje meno. Program potom vypíše „štítok“ v tvare:

```
*****
*      *
*  Andrea  *
*      *
*****
```

## 1.2 Prechádzanie znakmi reťazca, konštruovanie nového reťazca

Vypíšme jednotlivé znaky reťazca pod seba, každý znak na samostatný riadok. Na postupné prechádzanie reťazcom po jednotlivých znakoch môžeme použiť for cyklus:

```
for znak in 'Python':  
    print(znak)
```

alebo:

```
retazec = 'Python'  
for znak in retazec:  
    print(znak)
```

### Otázky:

9. Čo bude robiť nasledujúci program? Čo vypíše na obrazovku?

```
retazec = 'Python'  
poradie = 0  
for znak in retazec:  
    poradie += 1      #je to isté ako zápis poradie = poradie+1  
    print(znak)  
    print(poradie)
```

Funkcia `len(retazec)` zistí dĺžku reťazca.

```
veta = input('Napiš nejakú vetu:')  
dlzka = len(veta)  
print('Počet znakov v tvojej vete je:', dlzka)
```

K jednotlivým znakom reťazca `retazec` vieme pristupovať pomocou zápisu `retazec[číslo_znaku]`. Znaky sú číslované (indexované) postupne od `0` po `len(s)-1`. Číslu, ktorým pristupujeme k znaku reťazca, hovoríme aj `index`. Napríklad:

```
retazec = 'Python'  
print(retazec[0]) # vypíše 'P'  
print(retazec[1]) # vypíše 'y'
```

Celý reťazec môžeme vypísať aj takto:

```
retazec = 'Python'  
for i in range(len(retazec)):  
    print(retazec[i])  
    print(i)
```

V shelli uvidíme:

```
===== RESTART: vypis.py =====  
P  
0  
Y  
1  
t  
2  
h  
3  
o  
4  
n  
5  
>>>
```

Prvý znak reťazca má index `0` a posledný `len(s)-1`. Často potrebujeme pristupovať k poslednému znaku. Na pristupovanie k znakom z opačnej strany reťazca (od konca) môžeme používať na indexovanie aj záporné čísla. Zápornými indexmi označujeme znaky od konca reťazca (viď obrázok).

P	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

### Otzky:

10. Prečo posledný znak z reťazca `r` nezistíme zápisom `r[len(r)]`, ale zápisom `r[len(r)-1]`?
11. Čo sa stane, ak použijeme index, ktorý je mimo dĺžky reťazca? Napríklad: `retazec = 'Python'` `retazec[10]` alebo `retazec[-10]`.
12. Vieme zistiť, vypísat konkrétny znak reťazca. Čo sa stane, ak sa ho pokúsime zmeniť? Napríklad zápisom `retazec[2] = 'i'`.

Reťazec (`string`) je v Pythone nemenný typ (`immutable`), to znamená, že sa nám nepodarí zmeniť znak, napr. `retazec[1] = '-'` (Python vtedy ohlásí chybu).

```
>>> retazec = 'Python'
>>> retazec[1] = '-'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    retazec[1] = '-'
TypeError: 'str' object does not support item assignment
>>>
```

Namiesto zmeny reťazca musíme vždy konštruovať nový reťazec (môže mať aj rovnaké meno).

```
>>> meno = 'Hana'
>>> meno = 'J'+meno[1]+meno[2]+meno[3]
>>> meno
'Jana'
>>>
```

V nasledujúcom príklade vytvoríme nový reťazec, v ktorom všetky znaky `'o'` zameníme za hviezdičku `'*'`.

```
veta1 = 'Programujeme v Pythone'
veta2 = ''
for i in range(len(veta1)):
    if veta1[i] == 'o':
        veta2 = veta2+'*'
    else:
        veta2 = veta2+veta1[i]
print(veta1)
print(veta2)
```

Alebo môžeme niektoré znaky aj vynechať (napríklad medzery).

```
veta1 = 'Programujeme v Pythone'
veta2 = ''
for i in range(len(veta1)):
    if veta1[i] != ' ':
        veta2 = veta2+veta1[i]
print(veta1)
print(veta2)
```

### Úlohy:

5

Vytvorte program, ktorému na vstupe zadáme dva rovnako dlhé reťazce a program z nich vytvorí jeden spojený reťazec, v ktorom sa striedajú znaky z prvého a druhého reťazca. Napríklad zadáme

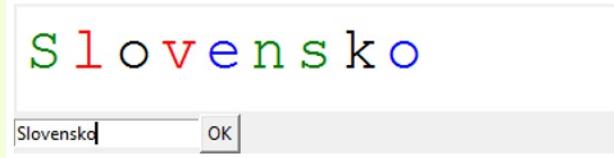
'Andrea' a 'Michal' a vznikne: 'AMnidcrheaal'.

- 6 Vytvorte program, ktorému na vstupe zadáme reťazec, ktorý vznikol prekladaním písmen z dvoch rôznych reťazcov (viď predchádzajúcu úlohu), a program nám vypíše dva pôvodné reťazce. Napríklad zadáme 'AMnidcrheaal' a vznikne 'Andrea' a 'Michal'.

- 7 Vytvorte program, ktorému na vstupe zadáme dva rôzne dlhé reťazce a program z nich vytvorí jeden spojený reťazec, v ktorom sa striedajú znaky z prvého a druhého reťazca. Kedže jeden z nich je kratší, na konci sú už len prilepené zostávajúce znaky z dlhšieho reťazca. Napríklad zadáme 'Diana' a 'Ferdinand' a vznikne: 'DFiearndainand'.

- 8 Vytvorte program, v ktorom používateľ zadá vetu v slovenčine ukončenú interpunkčným znamienkom (bodkou, otáznikom alebo výkričníkom). Program na základe ukončovacieho znamienka vypíše, či ide o vetu optytovaciu, rozkazovaciu alebo oznamovaciu.

- 9 Vytvorte program, v ktorom používateľ zadá slovo. Toto slovo sa do grafického plátna vypíše v podobe „reklamného nápisu“, kde každý znak zadaného slova bude vypísaný náhodne vybranou farbou (napr. z konkrétnie určených farieb).

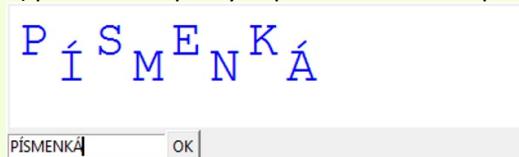


Slovensko

Slovensko

OK

- 10 Vytvorte program, v ktorom používateľ zadá slovo. Toto slovo sa do grafického plátna vypíše tak, že:
- a) písmená na párnych pozíciách budú napísané nižšie ako na nepárných pozíciách,

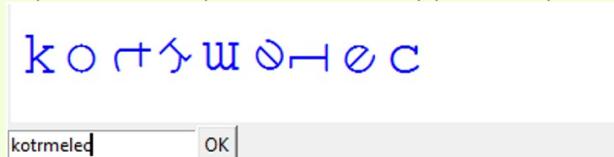


PÍSMEŇKÁ

PÍSMENKÁ

OK

- b) písmená budú pootáčané tak, aby posledné písmeno bolo opäť natočené „normálne“,

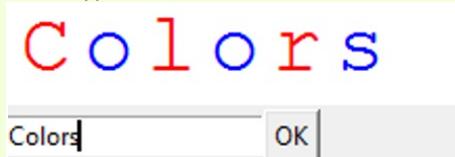


kotmeled

kotmeled

OK

- c) vo vypísanom slove sa budú striedať dve farby,

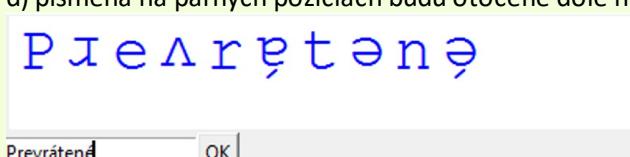


Colors

Colors

OK

- d) písmená na párnych pozíciách budú otočené dole hlavou.



Prevrátené

Prevrátené

OK

- 11 Vytvorte program, v ktorom používateľ zadá slovo.
- a) Toto slovo sa vypíše postupne po jednotlivých znakoch tak, že po každej sekunde sa objaví ďalšie písmeno.

Informatika

OK

Informatika

OK

Informatika

OK

- b) Jednotlivé písmená sa postupne po sekundách pridávajú k výslednému slovu tak, že prichádzajú sprava.  
c) Keď už bude vypísané celé slovo, po chvíli nápis „zhasne“ a začne sa vypisovať odnova.

12

Napíšte program, v ktorom používateľ zadá vete. Program vypíše:

- a) počet slov vo vete,  
b) dĺžku a poradové číslo najdlhšieho slova vo vete – ak ich je viac (s rovnakou dĺžkou), stačí vypísať jedno,  
c) najdlhšie slovo vo vete – ak ich je viac, vypíšte všetky.

13

Napíšte program, v ktorom používateľ zadá slovo. Program ho potom vypíše v takomto tvaru:

A  
H  
O  
J

#### Otázky:

13. V jednej z úloh sme navzájom spájali (prekladali znaky z jedného reťazca a druhého reťazca) dva nerovnako dlhé reťazce. Vedeli by sme výsledný reťazec rozdeliť na pôvodné reťazce? Svoju odpoveď zdôvodnite. Ako by sme mohli zlepšiť spôsob prekladania znakov tak, aby sme vedeli jednoducho rozdeliť aj reťazec, ktorý vznikol z dvoch rôzne dlhých reťazcov?

## 1.3 Podreťazce, rezy

Nové reťazce môžeme vytvárať aj pomocou rezov - indexovaním znakov s viacerými indexmi - **retazec [odkial : pokial]**. Znak s indexom **pokial** sa už nebude nachádzať vo výsledku. Vždy sem musíme písať index o jeden viac (podobne ako sme dávali hranice vo for cykle pre **range**).

```
>>> s = 'Programujeme v Pythone'  
>>> s[3:7]  
'gram'  
>>>
```

P	r	o	g	r	a	m	u	j	e	m	e		v		P	y	t	h	o	n	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> s = 'Programujeme v Pythone'  
>>> s[15:21]  
'Python'  
>>> s[15:22]  
'Pythone'  
>>>
```

Rezy môžeme použiť aj pri zmene formátu dátumu. Napríklad dátum 31. 12. 2017 chceme prepísať do tvaru rrrr-mm-dd, teda 2017-12-31:

```
>>> datum = '31-12-2017'  
>>> datum_en = datum[6:10] + '-' + datum[3:5] + '-' + datum[0:2]
```

```
>>> datum_en  
'2017-12-31'  
>>>
```

### Otázky:

14. Ktorý formát dátumu je vhodnejší pre informatiku? Prečo?
15. V programoch sa často používajú aj takéto zápisy. Vyskúšajte ich a zistite, čo znamenajú.
  - a)  $s[:5] \ s[:2]$
  - b)  $s[2:] \ s[3:]$
  - c)  $s[:]$
16. Ktoré z týchto zápisov budú fungovať a ktoré vrátia neprázdný reťazec?  
 $s[-5:-2]$   $s[5:2]$   $s[-2:-5]$   $s[-2:]$   $s[:-2]$

V reze môžeme určiť aj krok indexov (podobne ako v príkaze `range()`, ktorý sme používali vo `for` cykle). Krok určuje, o koľko sa budeme v indexoch posúvať od prvého uvedeného indexu. Ak krok neuvedieme, štandardne je krok 1.

```
>>> s = 'Programujeme v Pythone'  
>>> s[0:10:2]  
'Pormj'  
>>> s[0:22:2]  
'Pormjm yhn'  
>>> s[-1:-10:-1]  
'enohtyP v'  
>>> s[::-1]  
'enohtyP v emejumargorP'  
>>> s[::-1]  
'enohtyP v emejumargorP'  
>>>
```

Rez okrem konkrétnych čísel môže obsahovať aj premenné alebo nejaký výraz, ktorého výsledkom je číslo použiteľné v reze. Môžeme ich vytvárať aj takto:

```
s = 'Python'  
for i in range(len(s)):  
    print(s[:i+1])
```

```
P  
Py  
Pyt  
Pyth  
Pytho  
Python
```

```
s = 'Python'  
for i in range(len(s)):  
    ns = '-'*i+s[:i+1]  
    print(ns)
```

```
P  
-Py  
--Pyt  
---Pyth  
----Pytho  
-----Python
```

Pomocou rezov vytvoríme nový reťazec, v ktorom môžeme nahradieť niektorý znak alebo viacero znakov. Napríklad:

```
>>>s = 'Python'  
>>>s = s[:2] + 'T' + s[3:]  
>>>  
'PyThon'  
>>>
```

### Úlohy:

**14** Na vstupe zadáme reťazec, ktorý vznikol prekladaním písmen z dvoch rôznych reťazcov s rovnakou dĺžkou. Napríklad zadáme 'AMnidcrheaal'. Pomocou rezov (bez použitia for cyklu) vytvorte zo vstupu prvé a druhé slovo, teda: 'Andrea' a 'Michal'.

**15** Vytvorte program, ktorý zo vstupného reťazca nakreslí (ukážka je pre vstupný reťazec 'Python'):

```
.....P  
....PY  
....Pyt  
...Pyth  
..Pytho  
.Python
```

**16** Vytvorte program, ktorý zo vstupného reťazca nakreslí (ukážka je pre vstupný reťazec 'Python'):

```
.....PP.....  
.....yPPy.....  
....tyPPyt....  
...htyPPyth...  
.ohtyPPytho..  
.nohtyPPython.
```

**17** Napíšte program, v ktorom používateľ zadá platnú e-mailovú adresu. Program následne vypíše:  
a) doménu najvyššej úrovne (TLD) aj s bodkou,  
b) adresu mailového servera,  
c) meno používateľa,  
d) zoznam všetkých domén v poradí od domény prvej úrovne po doménu najnižšej úrovne.

```
Zadajte email: michal.velky@mail.pythonsoftware.net  
TLD: .net  
Server: mail.pythonsoftware.net  
User: michal.velky  
Domény:  
Doména 1. úrovne je: net  
Doména 2. úrovne je: pythonsoftware  
Doména 3. úrovne je: mail
```

**18** Napíšte program, v ktorom používateľ zadá platné rodné číslo z dvadsiateho storočia. Program následne vypíše:  
a) dátum narodenia v tvare deň.mesiac.rok,  
b) pohlavie.

Ukážka:

```
Rodné číslo: 806202/5675  
Dátum narodenia: 2.12.1980  
Pohlavie: Žena
```

(Pozn. Ženy majú v rodnom čísle k číslu mesiaca prirátané číslo 50)

19

Napište program, v ktorom používateľ zadá platnú URL adresu. Program následne vypíše:

- a) protokol použitej služby,
  - b) doménu najvyšszej úrovne,
  - c) doménovú adresu seryvera.

### Príklad:

<https://shop.pythonsoftware.com/download/examples>

## Výstup:

- a) https
  - b) com
  - c) shop.pythontutorialsoftware.com

## 1.4 Znaky a ich kódy

Už v minulosti sme sa pri reagovaní na udalosť stlačenia klávesu na klávesnici naučili zistiť tri rôzne informácie o klávese.

```
import tkinter  
canvas = tkinter.Canvas()  
canvas.pack()  
  
def klaves(event):  
    print(event.keysym)  
    print(event.keycode)  
    print(event.char)  
canvas.bind_all('<Key>', klaves)
```

Už teda vieme, že znaky majú aj svoj číselný kód. Počítač si všetko pamäta ako čísla, čiže aj znaky si pamäta ako číselné kódy. Kódy všetkých znakov si pamäta v tabuľke kódovania znakov. Existuje niekoľko tabuľiek kódovania znakov, ktoré sú navzájom rôzne, ale často majú kódovanie väčšiny znakov rovnaké (najčastejšie sa líšia kódy znakov národných abecied) (pozri napr.

[https://sk.wikipedia.org/wiki/K%C3%A1dovanie\\_\(informatika\)#K.C3.B3dovanie\\_znakov](https://sk.wikipedia.org/wiki/K%C3%A1dovanie_(informatika)#K.C3.B3dovanie_znakov).

Pomocou funkcie `ord('a')` zistíme kód znaku `'a'`. Naopak funkciou `chr(97)` zistíme, ktorý znak má číslo 97 v tabuľke kódovania znakov.

```
>>> ord('a')  
97  
>>> ord('b')  
98  
>>> chr(97)  
'a'  
>>> chr(98)  
'b'
```

Znaky majú čísla pridelené rozumne, tak, aby boli prirodzene za sebou zoradené. Napríklad za sebou idú znaky veľkej abecedy, za sebou idú znaky malej abecedy, za sebou idú číslice 0, 1 až 9. Znak 'a' má kód 97, za znakom 'z' nasleduje '{' (v tabuľke kódovania znakov) a má kód 123. Znaky malej abecedy si môžeme vypísať aj pomocou for cyklu:

```
for i in range(97, 123):  
    print(chr(i))
```

Ak si nepamäťame, kde začína malá abeceda v tabuľke kódovania znakov, môžeme to urobiť aj takto:

```
for i in range(ord('a'), ord('z')+1):
    print(chr(i))
```

alebo si môžeme vyrobiť textový reťazec, v ktorom budú iba znaky malej abecedy:

```
s=''
for i in range(97, 123):
    s = s+chr(i)
print(s)
```

### Otázky:

17. O koľko znakov je v tabuľke kódovania znakov vzdialenosť znaku 'A' a 'a'?
18. O koľko znakov je v tabuľke kódovania znakov vzdialenosť znaku 'B' a 'b'?
19. O koľko znakov je v tabuľke kódovania znakov vzdialenosť znaku 'Z' a 'z'?
20. Ako by sme mohli vyrobiť z veľkého písmena malé písmeno?
21. Ako by sme mohli vyrobiť z malého písmena veľké písmeno?
22. Ako by sme mohli vygenerovať náhodné malé písmeno (bez použitia funkcie choice)?

Znaky môžeme aj porovnať. Počítač porovná ich číselné hodnoty podľa tabuľky kódovania znakov. Napríklad pomocou podmienky v `if`-e môžeme zistiť, či je nejaký znak medzi 'a' a 'z'.

```
meno = input('Zadaj svoje meno:')
if 'a' <= meno[0] <= 'z':
    pismeno = chr(ord(meno[0])-32)
    nove_meno = pismeno+meno[1:]
    print(nove_meno+', mená pišeme s veľkým písmenom... ')
else:
    print(meno+', vidím, že meno pišeš správne :) ')
```

### Úlohy:

**20** Vytvorte program, ktorý vygeneruje náhodné 8-miestne heslo, ktoré obsahuje iba malé písmená.

**21** Vytvorte program, ktorý vygeneruje náhodné 8-miestne heslo, najprv tri veľké písmená, potom dve čísla a potom tri malé písmená.

**22** Vytvorte program, ktorý vygeneruje náhodné aspoň 8-miestne heslo tak, že najprv bude obsahovať iba malé písmená a následne niektoré z písmen (aspoň 1) zmení na veľké písmeno.

**23** Programom vytvorte reťazec, ktorý bude obsahovať postupne všetky písmená malej abecedy oddelené čiarkou.  
'a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z'

**24** Aspoň dvomi spôsobmi (pomocou cyklu for) vytvorte reťazec, ktorý obsahuje postupnosť všetkých cifier od 0 do 9 oddelených čiarkou.

'0, 1, 2, 3, 4, 5, 6, 7, 8, 9'

# 1.5 Pracujeme s textovými reťazcami

## Cézarova šifra

Cézarova šifra je najznámejšia a najstaršia šifra. Jej vznik siaha do čias vlády rímskeho cisára Caesara. Teraz si vytvoríme program, ktorým zašifrujeme zadaný text Cézarovou šifrou. Táto šifra je založená na tom, že každé písmeno správy je nahradené písmenom, ktoré je v abecede posunuté o zadaný posun. Napríklad správa '**abeceda**' by po zašifrovaní Cézarovou šifrou s posunom 1 bola '**bcdfeb**'. Pri šifrovaní s posunom 1 sa písmeno '**a**' zmení na '**b**', '**b**' na '**c**' atď. Posledné písmeno '**z**' v abecede sa zmení na '**a**'.

```
abcdefghijklmnopqrstuvwxyz (vstupná abeceda)  
bcdfghijklmnopqrstuvwxyz (prislúchajúce písmeno s posunom 1)
```

```
abcdefghijklmnopqrstuvwxyz (vstupná abeceda)  
cdefghijklmnopqrstuvwxyzab (prislúchajúce písmeno s posunom 2)
```

```
#Cézarova_šifra  
vstup = input('Zadaj text: ')  
sifra = ''  
for znak in vstup:  
    novyznak = znak  
    if 'a' <= znak <= 'y':  
        novyznak = chr(ord(znak)+1)  
    if znak == 'z':  
        novyznak = 'a'  
    sifra = sifra+novyznak  
print(sifra)
```

## Úlohy:

**25** Upravte program Cézarova\_šifra tak, aby šifroval vstup s posunom 2, neskôr 3.

**26** Upravte program Cézarova\_šifra tak, aby dešifroval šifru, ktorú sme zašifrovali s posunom 1.

**27** Upravte program Cézarova\_šifra tak, aby dešifroval šifru, ktorú sme zašifrovali s posunom 2, neskôr 3.

**28** Upravte program Cézarova\_šifra tak, aby sme mu okrem vstupu zadali aj posun a program zašifroval vstup so zadaným posunom.

## Otázky:

23. Aký je vzťah medzi posunom pri šifrovaní a dešifrovaní správy Cézarovou šifrou?
24. Je možné vytvoriť program na šifrovanie Cézarovou šifrou, ktorý pri istých posunoch bude dešifrovať už zašifrovanú správu?

## Náhodné zamiešanie znakov v textovom reťazci

## Otázky:

25. Ako by sme mohli vytvoriť nový reťazec, v ktorom budú znaky v náhodne zamiešanom poradí? Napríklad zo slova 'python' vznikne 'typonh'.
26. Vieme vytvoriť nový reťazec, v ktorom chýba jeden náhodný znak?

Najprv vytvorime program, ktorý odstráni z reťazca jeden zadaný znak:

```
s = 'Python'
kto = int(input('Zadaj index znaku, ktorý mám odstrániť:'))
s = s[:kto]+s[kto+1:]
print(s)
```

Teraz upravme program, aby sám náhodne vybral index znaku, ktorý odstráni, a odstránený znak vypíše:

```
from random import *
s = 'Python'
kto = randrange(len(s))
print('Odstraňujem znak', s[kto])
s = s[:kto]+s[kto+1:]
print(s)
```

Teraz nám už stačí postupne (náhodne) odstraňovať všetky znaky a odstránené pridávať do nového reťazca.

```
from random import *
s = 'Python'
novy = ''
for i in range(len(s)):
    kto = randrange(len(s))
    print('Odstraňujem znak', s[kto])
    novy = novy+s[kto]
    print('Zatiaľ som vytvoril:', novy)
    s = s[:kto]+s[kto+1:]
    print('Ešte zostali tieto znaky:', s)
print('Zamiešané slovo:', novy)
```

A takto môže vyzeráť priebeh programu:

```
Odstraňujem znak o
Zatiaľ som vytvoril: 
Ešte zostali tieto znaky: Pythn
Odstraňujem znak y
Zatiaľ som vytvoril: oy
Ešte zostali tieto znaky: Pthn
Odstraňujem znak t
Zatiaľ som vytvoril: oyt
Ešte zostali tieto znaky: Phn
Odstraňujem znak P
Zatiaľ som vytvoril: oytP
Ešte zostali tieto znaky: hn
Odstraňujem znak n
Zatiaľ som vytvoril: oytPn
Ešte zostali tieto znaky: h
Odstraňujem znak h
Zatiaľ som vytvoril: oytPnh
Ešte zostali tieto znaky:
Zamiešané slovo: oytPnh
```

### Otázky:

27. Môže sa stať, že po zamiešaní budeme mať rovnaký text ako na začiatku? Ak áno, môžeme to tiež považovať za náhodné zamiešanie?
28. Môže sa stať, že by sa v zamiešanom texte niektoré písmeno opakovalo? Za akých okolností? Je teda program správny?

## Úlohy:

29 Upravte program tak, aby sme mu mohli na vstupe zadať akýkoľvek reťazec.

30 Upravte program tak, aby nám vypísal viacero náhodných zamiešaní reťazca.

## 1.6 Logické operácie a textové reťazce

### Operácia `in` a pravdivostné hodnoty (boolean)

Operáciou `in` môžeme zistiť, či sa nejaký podreťazec nachádza v reťazci. Túto operáciu môžeme podobne ako porovnania použiť napríklad na rozhodovanie v `if`-e.

```
s = 'Python'
if 'o' in s:
    print('je tam')
else:
    print('nie je tam')
```

Ak tento výraz (`'o' in 'Python'`) zapíšeme priamo do shellu, Python nám napiše `True` alebo `False`, teda či je to pravda alebo nepravda.

```
>>> 'o' in 'Python'
True
>>> 'yt' in 'Python'
True
>>> 'ty' in 'Python'
False
>>>
```

Rovnako aj zapísanie iných výrazov do shellu, ktoré sme používali v `if`-e, Python hneď vyhodnotí a napíše nám, či je ich výsledkom pravda alebo nepravda.

```
>>> x = 10
>>> x > 7
True
>>> 9 < x < 12
True
>>> x == 7
False
>>>
```

Výsledok takého výrazu si tiež môžeme zapamätať do premennej. Takýto typ, ktorý obsahuje len hodnoty `True` alebo `False`, nazývame `boolean` (pravdivostné hodnoty). Môžeme to overiť funkciou `type()`.

```
>>> jetam = 'o' in 'Python'
>>> jetam
True
>>> type(jetam)
<class 'bool'>
>>> x = 5
>>> vysledok = x > 7
>>> vysledok
False
>>> type(vysledok)
<class 'bool'>
>>>
```

# 1.7 Ďalšie užitočné funkcie na prácu s textovými reťazcami

## Niekteré metódy textových reťazcov

Zatiaľ sme sa stretli s niektorými funkciemi na prácu s reťazcami. Niektoré z funkcií používajú špeciálny zápis v tvare `retazec.funkcia(parametre)`, sú už v Pythone definované presne na prácu s reťazcami. Takýmto funkciám sa hovorí metódy (viac si o tom povieme, keď sa budeme venovať objektovému programovaniu). Najčastejšie budeme používať tieto metódy:

`retazec.find(podretazec)` - vráti index prvého výskytu podreťazca v reťazci,  
`retazec.lower()` - vráti reťazec, v ktorom všetky veľké písmená zamení za malé,  
`retazec.upper()` - vráti reťazec, v ktorom všetky malé písmená zamení za veľké,  
`retazec.replace(podretazec1, podretazec2)` - vráti reťazec, v ktorom nahradí všetky výskytu `podretazec1` za reťazec `podretazec2`.

```
>>> s = 'Programujeme v Pythone'  
>>> s = s.lower()  
>>> print(s)  
programujeme v pythone  
>>>
```

```
>>> s = 'Programujeme v Pythone'  
>>> s = s.upper()  
>>> print(s)  
PROGRAMUJEME V PYTHONE  
>>>
```

Novú hodnotu nemusíme vždy priradiť premennej, niekedy ju stačí len vypísať:

```
>>> print('Programujeme v Pythone'.upper())  
PROGRAMUJEME V PYTHONE  
>>>
```

Metódou `replace` môžeme nahradiť rôzne veľké podreťazce rôzne veľkými podreťazcami:

```
>>> s = 'Okolo kolovrátka sedeli v kole všetci z okolia.'  
>>> s = s.replace('kol', '*')  
>>> print(s)  
O*o *ovrátka sedeli v *e všetci z o*ia.  
>>>
```

Môžeme ju využiť aj na odstránenie časti reťazca tak, že nahradíme podreťazec prázdnym reťazcom:

```
>>> s = 'Okolo kolovrátka sedeli v kole všetci z okolia.'  
>>> s = s.replace('kol', '')  
>>> print(s)  
Oo ovrátka sedeli v e všetci z oia.  
>>>
```

V nasledujúcej ukážke nahradíme všetky samohlásky pomlčkou:

```
samohlasky = 'áéíóúýý'  
s = 'Okolo kolovrátka sedeli v kole všetci z okolia.'  
for znak in samohlasky:  
    s = s.replace(znak, '-')  
    s = s.replace(znak.upper(), '-')  
print(s)
```

Program vypíše:

```
-k-l- k-l-vr-tk- s-d-1- v k-l- vš-tc- z -k-l--.
```

#### Otázky:

29. V reťazci s máme 'Okolo kolovrátku sedeli v kole všetci z okolia.' Ktorú hodnotu vráti metóda s.find(' '), ak je v reťazci s nejaká veta?
30. Čo bude výsledkom rezu s[:s.find(' ')] ?

#### Úlohy:

- 31** Vytvorte program, ktorý v zadanej vete nahradí všetky výskyty y, Y, i, a I pomlčkami, a našou úlohou je napísanie vety aj so správne doplnenými i a y. Program našu odpoveď skontroluje a nejako označí nesprávne doplnenia (ak sme v grafike, farebne ich zvýrazní, ak sme v textovom režime, za nesprávne zadaný znak vo vete napiše napríklad výkričník).
- 32** Pomocou programu vytvorte reťazec, ktorý obsahuje postupne všetky znaky malej abecedy od 'a' po 'z' a jednotlivé znaky sú oddelené čiarkou a medzerou. Za písmenom 'z' nie je ani medzera ani čiarka. Vymyslite viacero riešení tejto úlohy a diskutujte, ktoré z riešení je lepšie, krajsie, efektívnejšie.
- 33** Otočte reťazec zadaný na vstupe (napište odzadu), ale pri otočení nepoužite rezy.
- 34** Vytvorte modifikovanú hru obesenec. Program si vyberie jedno náhodné slovo (z vopred zadaných slov). Vytvorí rovnako dlhé slovo, ktoré obsahuje len pomlčky. Toto slovo bude animáciou postupne padať z horného okraja obrazovky k dolnému. Počas padania je našou úlohou uhádnuť slovo stláčaním písmen. Ak sa zadané písmeno v hľadanom slove nachádza, príslušné pomlčky tohto písmena sa nahradia uhádnutým písmenom. Po uhádnutí celého slova padá ďalšie slovo a získame bod. Ak slovo nestihнемe uhádnuť, hra končí.
- 35** Pri písaní textu sme namiesto klávesnice s rozložením znakov QWERTZ mali zapnutú klávesnicu s rozložením znakov QWERTY. Teda namiesto y sme písali z a namiesto z sme písali y. Vytvorte program, ktorému zadáme vety a program vypíše vety, kde budú z a y navzájom vymenené.

## 1.8 Formátovanie reťazcov

V úvode tejto kapitoly bol program rozhovor. Podľa odpovedí používateľa sme vytvárali nové otázky, v ktorých sme spájali odpoveď s iným textovým reťazcom.

```
#program rozhovor
from random import *
meno = input('Ako sa voláš?')
print('Ahoj '+meno+' rád Ťa spoznávam :)')
roknarodenia = input(meno+', v ktorom roku si sa narodil?')
meno2 = choice(('Alena', 'Barbora', 'Eva', 'Sofia'))
print('Á spomínam si, v roku '+roknarodenia+' sa narodila aj '+meno2)
```

Na úpravu, resp. na dopĺňanie reťazcov odpovedami môžeme použiť metódu **format**. Metóda **format** vloží na zadané miesto zadané parametre. Počet parametrov je rôzny. Metóda nahradí prvý výskyt {} prvým parametrom, druhý výskyt {} druhým parametrom, tretí výskyt {} tretím parametrom atď.. Napríklad:

```
>>> meno = 'Adam'
>>> 'Ahoj {}, rád Ťa spoznávam :).format(meno)
'Ahoj Adam, rád Ťa spoznávam :)'
>>>
```

Pozrite si ukážku programu s využitím metódy `format` s troma parametrami:

```
from random import *
meno = input('Ako sa voláš?')
rok = input('V ktorom roku si sa narodil?')
vek = 2017-int(roku)
meno2 = choice(('Alena', 'Barbora', 'Eva', 'Sofia'))
meno2 = meno2[:-1] + 'u'
s = 'Ahoj {}, rád Ťa spoznávam :). Poznáš {}? Aj ona má {} rokov.'.format(meno,
    meno2, vek)
print(s)
```

Vidíme, že parametrom môže byť aj číslo. Pre čísla sa dajú využiť aj ďalšie zápisy v `{}`. Tým môžeme zmeniť formát čísla podobne, ako sme zvyknutí v tabuľkovom kalkulátore. Napríklad zápis `{:b}` vypíše číslo v dvojkovej sústave, `{:x}` vypíše číslo v šestnástkovej sústave.

```
>>>'V mojom oblúbenom binárnom zápise máš {:b} rokov.'.format(vek)
'V mojom oblúbenom binárnom zápise máš 10000 rokov.'
>>>'V hexadecimálnom zápise máš {:x} rokov.'.format(vek)
'V hexadecimálnom zápise máš 10 rokov.'
>>>vek = 4
>>>'V hexadecimálnom zápise máš {:05x} rokov.'.format(vek)
'V hexadecimálnom zápise máš 00004 rokov.'
```

### Otzáky:

31. Zistite, akým spôsobom menia formát čísla zápisy: `{:05x}` `{:04x}` `{:02x}`.
32. Zistite, akým spôsobom menia formát čísla zápisy: `{:4}` `{:3}` `{:2}`.
33. Zistite, akým spôsobom menia formát čísla zápisy: `{:04}` `{:03}` `{:02}`.

Pomocou metódy `format` si z jednotlivých farebných zložiek (RGB) uvedených v desiatkovej sústave môžeme vytvoriť zápis farby v podobe textového reťazca v šestnástkovej sústave.

```
r = 100
g = 200
b = 100
farba = '#{0:02x}{1:02x}{2:02x}'.format(r, g, b)
print(farba)
```

Teraz môžeme generovať a kresliť rôzne farby aj systematicky for cyklom:

```
import tkinter
from random import *
canvas = tkinter.Canvas(width = 520, height = 520, bg = 'black')
canvas.pack()

for r in range(256):
    for g in range(256):
        f = '#{0:02x}{1:02x}{2:02x}'.format(r, g, 0)
        canvas.create_rectangle(r*2, g*2, r*2+2, g*2+2, fill = f,
                               outline = '')
```



Metódu **format** môžeme využiť aj pri formátovaní desatinných čísel (dátový typ **float**).

```
>>> cislo = 10/3
>>> cislo
3.333333333333335
>>> '{:5.2f}'.format(cislo)
' 3.33'
>>> '{:5.0f}'.format(cislo)
'   3'
>>> '{:.0f}'.format(cislo)
'3'
>>> '{:.4f}'.format(cislo)
'3.3333'
>>> type(cislo)
<class 'float'>
```

Zápisom **.2f** určíme, že desatinné číslo (**float**) sa zobrazí na dve desatinné miesta. Zápis **:7.3f** znamená, že sa desatinné číslo zobrazí na 3 desatinné miesta a celkovo bude na zobrazenie vyhradených 7 miest. Štandardne sa desatinné čísla zobrazujú na 6 miest **'{:f}'.format(cislo)**.

#### Úlohy:

**36**

Vykachličkujte celú plochu canvasu farebnými štvorčekmi. Každý štvorček bude mať náhodnú farbu „namiešanú“ z hodnôt RGB.

**37**

Vytvorte program, ktorý zobrazí pás so všetkými odtieňmi modrej. Pás bude tvorený z tenkých čiar. Modrá zložka farby čiary sa bude postupne zvyšovať.



## 2 N-tice (tuple)

### 2.1 N-tice textových reťazcov - farby a slová

Pri žrebovaní náhodnej farby funkciou `choice` sme farby zapisovali do akéhosi zoznamu farieb a všetky farby boli zapísané v zátvorkách. Rovnako pri vytváraní náhodných viet sme jednotlivé slová, z ktorých sme skladali vety, zapisovali do akéhosi zoznamu a z neho sme tiež funkciou `choice` vyberali jedno náhodné slovo.

```
farba = choice(('red', 'green', 'yellow', 'blue'))
```

Tieto informácie si tiež môžeme zapísat do premennej a pomocou funkcie `type()` už vieme zistiť typ takejto premennej. Takýto typ nazývame **n-tica** alebo **tuple**. Vyskúšajme to v shelli:

```
>>> from random import *
>>> farby = ('red', 'green', 'yellow', 'blue')
>>> type(farby)
<class 'tuple'>
>>> choice(farby)
'blue'
>>> choice(farby)
'yellow'
>>>
```

My už vieme pracovať s textovým reťazcom a môžeme povedať, že reťazec je postupnosťou znakov. N-tica je tiež postupnosťou, ale nie elementárnych znakov, ale rôznych informácií. Prvkom n-tice môže byť textový reťazec, ale aj číslo alebo niečo iné.

```
>>> informacie = ('red', 'Eva', 23, 3.14, 'abc')
>>> type(informacie)
<class 'tuple'>
>>>
```

Kedže aj textový reťazec, aj n-tica sú postupnosti, mohli by mať niektoré vlastnosti rovnaké a možno môžeme na prácu s nimi používať rovnaké funkcie.

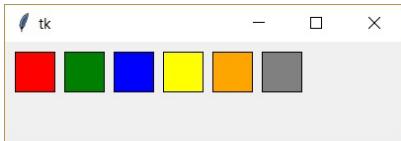
#### Otázky:

1. Ktoré z funkcií a operácií používaných na prácu s reťazcami môžeme použiť na prácu s n-ticami?
2. Vyskúšajte indexovanie prvkov. Môžeme ho použiť aj na n-tice? Je v niečom rozdiel?
3. Ako to bude s používaním rezov (slice)?
4. For cyklom sme mohli prechádzať prvkami reťazca. Funguje to aj pre n-tice?

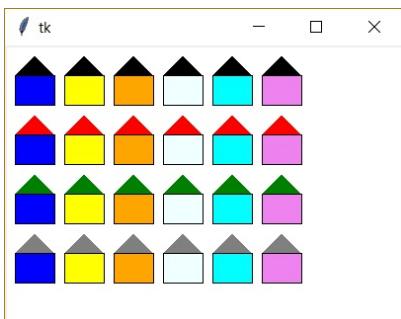
Nasledujúci program nakreslí štvorčeky postupne všetkými farbami z n-tice:

```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 100)
canvas.pack()

farby = ('red', 'green', 'blue', 'yellow', 'orange', 'grey')
x = 10
for farba in farby:
    canvas.create_rectangle(x, 10, x+40, 50, fill = farba)
    x += 50
```

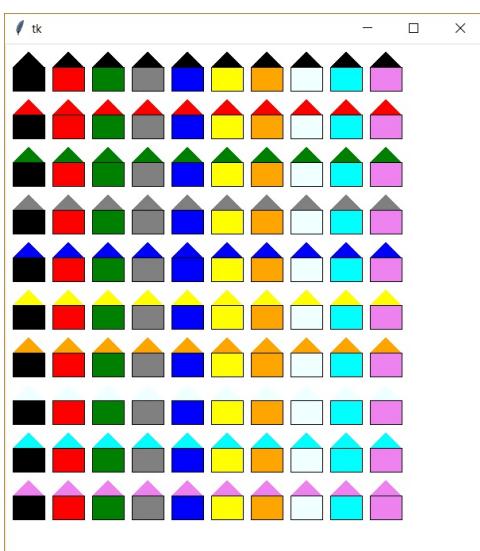


Stavebná firma ponúka zákazníkom rôzne kombinácie farby strech a fasády domu. Na farbu strech a fasády používajú úplne inú paletu farieb. Nasledujúci program im vykreslí všetky farebné kombinácie podľa ponúkaných farieb strechy a palety farieb pre fasádu:



```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 280, bg = 'white')
canvas.pack()
farby1 = ('black', 'red', 'green', 'grey')
farby2 = ('blue', 'yellow', 'orange', 'azure', 'cyan', 'violet')
x = 10
y = 30
for f1 in farby1:
    for f2 in farby2:
        canvas.create_polygon(x, y, x+20, y-20, x+40, y, fill = f1)
        canvas.create_rectangle(x, y, x+40, y+30, fill = f2)
        x += 50
    x = 10
    y += 60
```

Firma sa rozhodla, že zvýši ponuku farieb a každá z doposiaľ ponúkaných farieb sa bude dať použiť aj na farbu strechy aj fasády. V programe stačí z dvoch n-tíc vytvoriť jednu, ktorá vznikne spojením `farby1` a `farby2`:



```
import tkinter
canvas = tkinter.Canvas(width = 600, height = 650, bg = 'white')
canvas.pack()

farby1 = ('black', 'red', 'green', 'grey')
```

```

farby2 = ('blue', 'yellow', 'orange', 'azure', 'cyan', 'violet')
cela_ponuka = farby1+farby2
x = 10
y = 30
for f1 in cela_ponuka:
    for f2 in cela_ponuka:
        canvas.create_polygon(x, y, x+20, y-20, x+40, y, fill = f1)
        canvas.create_rectangle(x, y, x+40, y+30, fill = f2)
        x += 50
    x = 10
    y += 60

```

### Otázky:

5. Čo sa zmení v ponuke firmy, keď v programe hned' za riadok `cela_ponuka = farby1+farby2` napíšeme riadok: `cela_ponuka = cela_ponuka[::-2]`?

### Úlohy:

**1**

V programe vytvoríme tri n-tice s názvom podmet, prísudok a predmet.

```

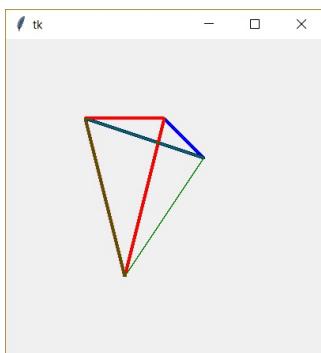
podmet = ('Kamarát', 'Spolužiak', 'Andrej', 'Roman')
prisudok = ('videl', 'prezradil', 'povedal', 'napísal', 'zistil',
            'nakreslil')
predmet = ('tajomstvo', 'prekvapenie', 'predsavzatie')

```

Vytvorte program, ktorý vypíše všetky možné vety so slovosledom: podmet, prísudok a predmet.

## 2.2 Body v rovine

Pomocou n-tíc môžeme definovať aj body v rovine. Súradnice bodov sú dvojice čísel (čiže n-tica s n=2). Vytvoríme body A, B, C a D so súradnicami A = (100, 100), B = (200, 100), C = (250, 150), D = (150, 300) a z nich nakreslíme trojuholníky ABC, ABD a ACD, každý inou farbou.



```

import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

A = (100, 100)
B = (200, 100)
C = (250, 150)
D = (150, 300)
canvas.create_line(A, B, C, A, fill = 'blue', width = 4)
canvas.create_line(D, B, A, D, fill = 'red', width = 4)
canvas.create_line(A, C, D, A, fill = 'green', width = 2)

```

Program doplníme. Pri pohybe myši zmeníme súradnice bodu D na aktuálne súradnice myši, pôvodné

trojuholníky (čiže štvorsten) zmažeme a nakreslíme ich znova s novými súradnicami bodu D. Teraz môžeme pohybom myši interaktívne meniť jeden z vrcholov štvorstena.

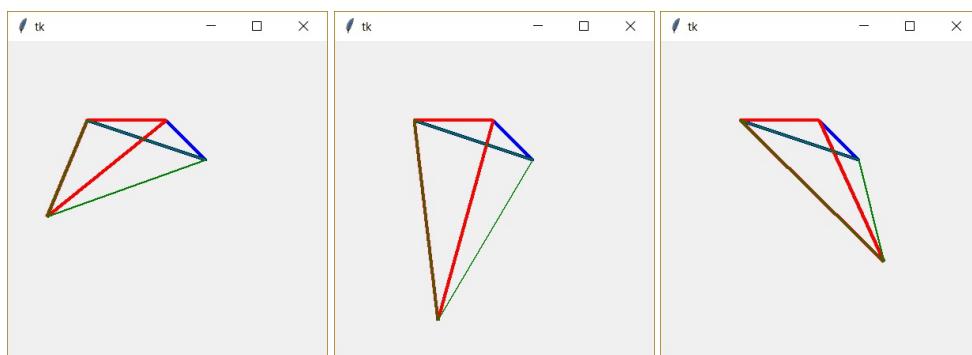
```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

A = (100, 100)
B = (200, 100)
C = (250, 150)
D = (150, 300)

def kresli():
    canvas.delete('all')
    canvas.create_line(A, B, C, A, fill = 'blue', width = 4)
    canvas.create_line(D, B, A, D, fill = 'red', width = 4)
    canvas.create_line(A, C, D, A, fill = 'green', width = 2)

def zmenD(suradnice):
    global D
    D = (suradnice.x, suradnice.y)
    kresli()

kresli()
canvas.bind('<Motion>', zmenD)
```



### Úlohy:

2

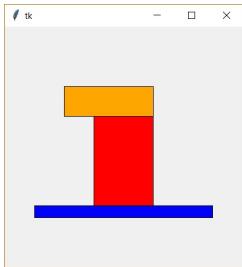
Vylepšite program štvorsten. Napríklad vymyslite spôsob ovládania, ktorým môžeme meniť polohu iných vrcholov.

N-tice môžeme použiť aj pri zadávaní súradníc v kreslení iných útvarov (oval, rectangle, polygon). Dajú sa použiť ako jednotlivé body, alebo celú postupnosť súradníc tvorí jedna n-tica. Napríklad:

```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

A = (100, 100)
B = (200, 100)
C = (250, 150)
D = (150, 300)

canvas.create_rectangle(A, C, fill = 'orange')
obdlznik1 = C+D
canvas.create_rectangle(obdlznik1, fill = 'red')
obdlznik2 = (50, 300, 350, 320)
canvas.create_rectangle(obdlznik2, fill = 'blue')
```



Teraz vytvoríme program na kreslenie čiary klikaním myši. Na začiatku budeme mať prázdnú n-ticu. N-tica bude postupnosť súradnic čiary. Pri každom kliknutí myši pridáme do tejto n-tice x-ovú a y-ovú súradnicu myši, čiže dve čísla. Takúto novú postupnosť bodov nakreslíme ako čiaru.

```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

ciara = ()

def kresli():
    canvas.delete('all')
    if len(ciara) >= 4:
        canvas.create_line(ciara, fill = 'blue', width = 4)

def pridajbod(suradnice):
    global ciara
    ciara = ciara + (suradnice.x, suradnice.y)
    kresli()

canvas.bind('<Button-1>', pridajbod)
```



### Otázky:

6. Prečo je vo funkčii kresli príkaz if? Čo by sa zmenilo vo funkčnosti programu, ak by sme ho vyniechali?

Nakreslili sme nejaký obrázok. Teraz budeme experimentovať v príkazovom riadku. Najprv si vypíšeme celú n-ticu **ciara**:

```
>>> ciara
(239, 77, 178, 61, 126, 91, 106, 104, 79, 154, 61, 197, 77, 260,
... 232, 248)
>>>
```

Pripravíme si prázdnú n-ticu **ciara\_zmensená**, do ktorej postupne vložíme polovičné hodnoty čísel z pôvodnej n-tice **ciara**. Pozor, jednoprvkovú n-ticu zapíšeme v zátvorke a za jedným prvkom musí byť čiarka. Napríklad: **(10,)** je jednoprvková n-tica. Všetko budeme písat v shelli, aby sme hneď videli, čo sa bude diať.

```
>>> ciara_zmensená = ()
>>> for i in ciara:
```

```

        ciara_zmensená += (i/2,)

>>>ciara_zmensená
(119.5, 38.5, 89.0, 30.5, 63.0, 45.5, 53.0, 52.0, 39.5, 77.0, 30.5, 98.5,
... 116.0, 124.0)

```

Vidíme, že v n-tici `ciara_zmensená` máme polovičné hodnoty pôvodných súradníc. Teraz zmažeme všetko nakreslené a nakreslíme novú čiaru, ale už podľa zmenšenej n-tice `ciara_zmensená`. Nakreslíme ju inou farbou a označujeme si ju, aby sme ju mohli posunúť, následne znova nakreslíme inou farbou čiaru podľa n-tice `ciara_zmensená`.

```

>>>canvas.delete('all')
>>>canvas.create_line(ciara_zmensená, fill = 'red', width = 4, tags = 'foto1')
57
>>>canvas.move('foto1', 200, 0)
>>>canvas.create_line(ciara_zmensená, fill = 'green', width = 4, tags = 'foto2')
58

```

Na obrázku vidíme výsledok:



#### Úlohy:

**3** Doplňte program tak, aby nám po stlačení tlačidla zmenšil obrázok.

**4** Zmeňte program tak, aby zmenšoval na polovicu len šírku alebo len výšku obrázku.

## 2.3 N-tice ako parameter

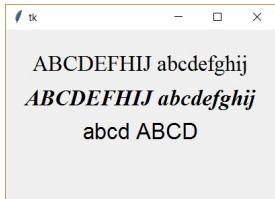
Pri nastavovaní typu písma v príkaze `canvas.create_text` sme doposiaľ vedeli zadávať len jednoslovny názov písma. Práve n-tice nám pomôžu zadávať aj viacslovný názov fontu. V skutočnosti v parametri font zadávame n-ticu. Jej prvým prvkom je názov fontu, druhý prvak je veľkosť fontu a ďalšími môžu byť nastavenia pre `bold`, `italic` a `underline`.

```

import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

canvas.create_text(200, 50, text = 'ABCDEFGHIJ abcdefghij',
                  font = ('Times New Roman', 20))
canvas.create_text(200, 100, text = 'ABCDEFGHIJ abcdefghij',
                  font = ('Times New Roman', 20, 'bold', 'italic'))
canvas.create_text(200, 150, text = 'abcd ABCD', font = ('', 20))

```



Pomocou spájania jednoprvkových n-tíc si vieme vytobiť aj ukážky rôznych veľkostí písma:

```
import tkinter
canvas = tkinter.Canvas(width = 600, height = 400)
canvas.pack()
y = 10
for i in range(10, 30, 2):
    canvas.create_text(300, y, text = 'ABCDEFGHIJ abcdefghij',
                       font = ('Times New Roman',) + (i,))
    y += i*2
```

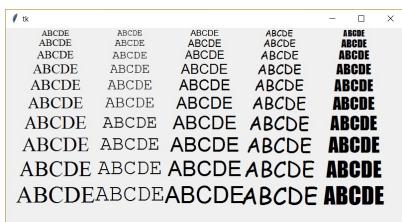


Aj z názvov typu písma môžeme vytobiť n-ticu. Prechádzaním prvkov tejto n-tice a vytváraním novej n-tice na nastavenie písma si vytvoríme vzorky písma:

```
import tkinter
canvas = tkinter.Canvas(width = 800, height = 400)
canvas.pack()

pisma = ('Times New Roman', 'Courier New', 'Brush Script', 'Comic Sans MS',
          'Impact')

x = 100
for j in range(len(pisma)):
    y = 10
    for i in range(10, 30, 2):
        canvas.create_text(x, y, text = 'ABCDE', font = (pisma[j],)+(i,))
        y += i*2
    x += 150
```



### Úlohy:

5

- Zadefinujte si vlastnú n-ticu so súradnicami bodov tvoriacich vhodnú lomenú čiaru – výškový profil turistickej trasy (x-ové súradnice vyjadrujú prejdenú vzdialenosť a y-ové súradnice nadmorskú výšku bodu). Vytvorte program, ktorý:
- zobrazí túto lomenú čiaru,
  - spočíta celkovú dĺžku klesania na trase (počet výškových metrov, ktoré turista dokopy stratí v klesajúcich úsekokach) a celkovú dĺžku stúpania,

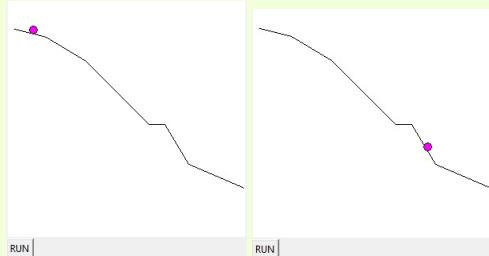
c) spočíta reálnu dĺžku trasy (čiary).



Stúpanie:180  
Klesanie:120  
Celkom:438

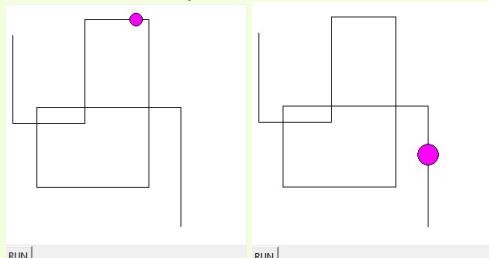
6

Vytvorte program, ktorý bude simulovať plynulý pohyb guličky spustenej dolu vyskladanou traťou. Trať si zadefinujte ako n-ticu bodov klesajúcej lomenej čiary. Po spustení simulácie sa gulička bude pohybovať smerom dolu po čiare. Na konci čiary zastane.



7

Vytvorte program, ktorý bude simulovať jazdu smetiarskeho auta po určenej trase v uliciach mesta. Ulice sú na seba kolmé, na každom rohu trasy sa nachádza smetný kôš, ktorý treba vysypať. (Koše zobrazovať nemusíte). Po každom vyprázdení koša („na každom rohu“) sa krúžok predstavujúci smetiarske vozidlo zväčší, čím znázorňuje zväčšujúci sa objem odpadu, ktorý vezie. Trasu smetiarskeho auta si zadefinujte ako vlastné n-ticu bodov tak, aby jednotlivé úseky trasy boli na seba kolmé.



## 2.4 Viacnásobné priradenie

Vyskúšajme v shelli tieto zápisby:

```
>>> a = 1, 2, 3, 4, 5
>>> a
(1, 2, 3, 4, 5)
>>> type(a)
<class 'tuple'>
>>>
```

```
>>> info = 'Python', 2, True, 0.3, -5
>>> info
('Python', 2, True, 0.3, -5)
>>> type(info)
<class 'tuple'>
>>>
```

V oboch ukážkach Python priradil do premennej n-ticu. Pre Python sú v zápise na identifikáciu n-tice dôležitejšie čiarky ako zátvorky.

**Otázky:**

7. Čo si myslíte, bude fungovať aj takýto zápis for cyklu? Svoju odpoveď zdôvodnite a následne overte.
- for i in 1, 2, 5, 8, 13:  
    print(i)
  - for i in 1, 'Python', False, 0.8, -13:  
    print(i)

Zapisovanie n-tíc bez zátvoriek sa využíva aj na viacnásobné priradenie. Jedným priradením priradíme viacerým premenným naraz hodnoty. Počet premenných na ľavej strane a počet prvkov postupnosti na pravej strane musí byť rovnaký.

```
>>> r, g, b = 0, 200, 50  
>>> r  
0  
>>> g  
200  
>>> b  
50
```

```
>>> meno, priezvisko, roknarodenia = 'George', 'Orwell', 1903  
>>> meno  
'George'  
>>> priezvisko  
'Orwell'  
>>> roknarodenia  
1903
```

Tento zápis sa často používa aj na vzájomnú výmenu hodôt premenných:

```
>>> a, b = 10, 20  
>>> a  
10  
>>> a, b = b, a  
>>> a  
20  
>>> b  
10
```

**Otázky:**

- Ako môžeme iným spôsobom vymeniť hodnoty dvoch premenných?
- Dá sa viacnásobné priradenie použiť aj na výmenu hodnôt viacerých premenných?
- Môžeme viacnásobné priradenie použiť aj na výmenu hodnôt premenných navzájom rôzneho typu?
- Čo bude výsledkom nasledujúcich priradení?
  - a = b = c = 0
  - a, b = c, d = 10, 20

# 3 Textové súbory

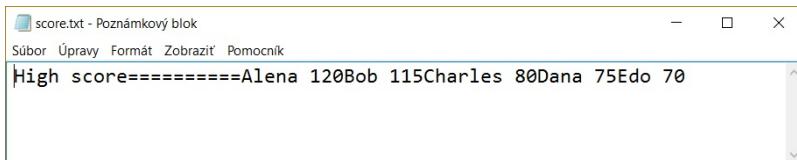
## 3.1 Zápis do textového súboru

Pomocou premenných si vieme pamätať informácie v spustenom programe. Ak si informácie uložíme do súboru, môžeme s nimi pracovať aj neskôr po opäťovnom spustení programu. Ukážeme si najjednoduchšie a najčastejšie používané súbory - textové súbory.

```
subor = open('score.txt', 'w')
subor.write('High score')
subor.write('='*10)
subor.write('Alena 120')
subor.write('Bob 115')
subor.write('Charles 80')
subor.write('Dana 75')
subor.write('Edo 70')
subor.close()
```

Zápisom `subor = open('score.txt', 'w')` otvoríme súbor `score.txt` na zapisovanie a do súborovej premennej `subor` (môžeme si ju pomenovať aj inak) sa priradí spojenie so súborom. Príkazom `subor.write(reťazec)` zapíšeme zadaný reťazec do súboru. Na správne ukončenie súboru musíme na konci súbor zatvoriť `subor.close()`. Zatvorením súboru sa zruší spojenie premennej `subor` so súborom. Keď na konci práce súbor nezatvoríme, nemusí Python všetky zápis fyzicky zapísť aj na disk. Pri otváraní súboru na zápis nemusí súbor s týmto menom ešte existovať. Python v takomto prípade vytvorí nový súbor. Ak už súbor s rovnakým menom existuje, Python vymaže jeho pôvodný obsah. A kde nájdeme tento súbor? Ak neurčíme celú cestu k súboru, nájdeme ho v rovnakom priečinku, kde je umiestnený program.

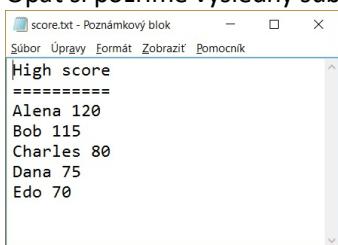
Pozrime sa na vytvorený súbor pomocou jednoduchého editora textových súborov - napríklad cez poznámkový blok:



Vidíme, že sme použili viackrát príkaz `write`, všetky zapísané reťazce sa nám uložili tesne za sebou do jedného riadku. Koniec riadku môžeme zapísť znakmi '`\n`'. Aj jedným príkazom `write` môžeme zapísť viac riadkov. Nás program môže vyzeráť aj takto:

```
subor = open('score.txt', 'w')
subor.write('High score\n')
subor.write('=='*10)
subor.write('\nAlena 120\nBob 115\nCharles 80\n')
subor.write('Dana 75\n')
subor.write('Edo 70\n')
subor.close()
```

Opäť si pozrime výsledný súbor cez editor súborov (napríklad cez poznámkový blok):



Vidíme, že súbor obsahuje viac riadkov, koniec riadku sme určili znakmi '`\n`'.

### Otázky:

- Čo bude obsahovať textový súbor vytvorený týmto programom?

```
f = open('pokus.txt', 'w')
f.write('\n'*10)
f.write('=')
f.close()
```

- Chceme vytvoriť textový súbor, ktorý obsahuje očíslované riadky postupne číslami 1, 2 až 9. Ktorý z programov je správnym riešením? Nájdite chyby v ostatných programoch.

```
subor = open('pokus1.txt', 'w')
for i in range(1, 10):
    subor.write(i)
subor.close()

subor = open('pokus2.txt', 'w')
for i in range(1, 10):
    subor.write(i+'\n')
subor.close()

subor = open('pokus3.txt', 'w')
for i in range(1, 10):
    subor.write(str(i) + '\n')
subor.close()

subor = open('pokus4.txt', 'w')
for i in range(1, 10):
    subor.write(str(i))
    subor.write('\n')
subor.close()

for i in range(1, 10):
    subor = open('pokus5.txt', 'w')
    subor.write(str(i) + '\n')
    subor.close()
```

Do textového súboru (otvoreného na zápis) môžeme zapisovať aj príkazom `print`. Príkaz `print` má štandardný výstup do shellu, ale s parametrom `file` a súborovou premennou ho zmeníme do daného súboru.

```
subor = open('pokus.txt', 'w')
for i in range(1, 10):
    print(i, file = subor)
subor.close()
```

### Úlohy:

1

Vytvorte program, ktorý do textového súboru zapíše tabuľku kódovania znakov malej abecedy. Na samostatnom riadku bude kód písmena abecedy, medzera a znak.

2

Vytvorte program, ktorý do textového súboru vygeneruje zadaný počet náhodných hesiel. Heslo má práve 8 znakov a skladá sa len z písmen malej abecedy.

## 3.2 Pridávanie riadkov do textového súboru

Textový súbor môžeme otvoriť aj s parametrom `'a'`. Napríklad: `subor = open('mena.txt', 'a')`. Tým ho otvoríme na pridávanie riadkov (append), to znamená, že sa nevymaže pôvodný obsah súboru. Pri pridávaní

zapisujeme informácie rovnakým spôsobom ako pri štandardnom zápise. Rovnako nie je potrebné, aby pri otvorení na pridávanie alebo zápis súbor už existoval. Nasledujúca ukážka programu zistí naše meno a vek a tieto informácie pridá na koniec súboru (obe do jedného riadku). Po každom spustení programu súbor predĺžime o jeden riadok.

```
meno = input('Ako sa voláš?')
print('Vitaj ',meno,', som rád, že ťa vidím.')
vek = input('Kolko máš rokov?')
subor = open('mena.txt', 'a')
subor.write(meno+' ')
subor.write(vek+'\n')
print(meno, ' zapamätal som si informácie o Ťebe do súboru :)')
subor.close()
```

### Otzázky:

3. V predchádzajúcej ukážke programu je premenná `vek`. Prečo ju nemusíme premeniť na reťazec funkciou `str()`?
4. Odhadnite a vyskúšajte, ako sa bude správať predchádzajúci program, ak vynecháme zatvorenie súboru pri ďalších spusteniach.
5. Môžeme nejakým spôsobom zachovať rovnakú funkčnosť programu a nepoužiť príkaz `write?` Ktoré riešenie je v tejto situácii podľa vás vhodnejšie?

```
veta = input('Napiš svoj odkaz, ktorý uložím pre budúcnosť :) :')
if veta == 'delete':
    subor = open('odkazy.txt', 'w')
    subor.close()
else:
    subor = open('odkazy.txt', 'a')
    subor.write(veta+'\n')
    subor.close()
```

Tento program nám slúži na písanie odkazov do budúcnosti. V prípade, že zadáme ako text odkazu slovo `delete`, program vymaže doposiaľ zapísané odkazy. Už vieme, že otvorením súboru na zápis sa vymaže jeho doterajší obsah. Keďže súbor hneď zatvoríme, zostane súbor prázdny.

### Úlohy:

**3** Upravte program na písanie odkazov do budúcnosti tak, aby do súboru nezapisoval prázdne riadky v prípade, že používateľ nič nenapíše, iba stlačí enter.

**4** V obchodnom centre je pri východe dotykový displej, na ktorom môžeme vyjadriť svoju spokojnosť / nespokojnosť s poskytnutými službami. Vytvorte program, ktorý sa nás spýta na spokojnosť so službami a hlasovanie zákazníka pridá do textového súboru `spokojnost.txt`.

Boli ste spokojní  
s našimi službami ?

Áno :)

Nie :(

**5** Vytvorte program, v ktorom ľavým tlačidlom myši kreslíme do canvasu malé krúžky. Súradnice `x` a `y` stredu každého krúžku sa po nakreslení pridajú do textového súboru `kreslenie.txt`. Pričom súradnica `x` bude na samostatnom riadku a na ďalšom riadku bude súradnica `y`. V programe môžeme,

napríklad stlačením medzery, vymazať všetky nakreslené krúžky. Vymazaním nakreslených krúžkov sa vymazú aj doposiaľ uložené informácie v textovom súbore. Ukážka súboru pre tri nakreslené krúžky na súradničiach 100, 200 a 57, 84 a 251, 238:

```
100  
200  
57  
84  
251  
238
```

### 3.3 Čítanie z textového súboru

Rovnako ako pri zápisе do súboru aj pri čítaní musíme súbor najprv otvoriť. Parametrom `'r'` namiesto parametra `'w'` alebo `'a'` súbor otvoríme na čítanie `subor = open('meno.txt', 'r')`. Ak nezadáme žiadny parameter, súbor sa automaticky otvorí na čítanie `subor = open('meno.txt')`. Pozor, pri otváraní súboru na čítanie musí už súbor existovať. Inak nám Python ohlási chybu. Problémy môžu nastať pri rôznych kódovaniach znakov v súbore. Niekedy bude potrebné, aby sme upresnili použité kódovanie. Nastavíme ho parametrom `encoding - subor = open('meno.txt', 'r', encoding = 'kódovanie')`. Kódovania môžu byť napríklad: `'utf-8'`, `'cp1250'`, `'iso88591'`. Vytvorme si pomocou jednoduchého textového editora (alebo programom) textový súbor s týmto obsahom:

```
prvý riadok  
druhý riadok  
tretí riadok
```

a uložme ho s názvom `info1.txt`. Jeden celý riadok súboru prečítame príkazom `riadok = subor.readline()`. Po prečítaní riadku sa jeho obsah uloží do premennej `riadok`. Prečítaný riadok si nemusíme vždy zapamätať do premennej, môžeme ho použiť aj v nejakej funkcií, ktorá očakáva na vstupe reťazec. Napríklad: `print(subor.readline())`.

Pri prezeraní textového súboru v textovom editore nevidíme znaky konca riadkov `'\n'`, ale po prečítaní celého riadku do premennej sú na konci textu aj znaky `'\n'`. Prvý riadok súboru môžeme prečítať týmto programom:

```
subor = open('info1.txt', 'r')  
riadok = subor.readline()  
print(riadok)  
subor.close()
```

Aj pri čítaní súboru je vhodné po skončení čítania súbor zatvoriť. Po spustení programu nám Python do shellu vypíše text `'prvý riadok'` a jeden prázdny riadok. To spôsobil neviditeľný znak konca riadku (enter `'\n'`). Aby sme sa presvedčili, čo je v premennej `riadok`, vypísali sme ju priamo v shelli a vidíme, že tam je `'prvý riadok\n'`.

```
===== RESTART: reading.py =====  
prvý riadok  
  
>>> riadok  
'prvý riadok\n'  
>>>
```

Ked' chceme vidieť skutočný obsah reťazca - presnú reprodukciu reťazca aj so špeciálnymi znakmi (napr. medzerami na konci a tiež koncom riadku) tak, ako sa vypíše po zadaní mena premennej priamo v shelli, použijeme funkciu `repr(retazec)`.

```
subor = open('info1.txt', 'r')  
riadok = subor.readline()
```

```
print(repr(riadok))
riadok = subor.readline()
print(repr(riadok))
riadok = subor.readline()
print(repr(riadok))
subor.close()
```

Teraz vidíme vypísaný celý súbor aj so všetkými znakmi ako reťazce (v apostrofoch):

```
===== RESTART: reading2.py =====
'prvý riadok\n'
'druhý riadok\n'
'tretí riadok'
>>>
```

Upravíme náš textový súbor. Druhý riadok necháme úplne prázdný a pozrieme sa na výpis súboru programom.

```
prvý riadok
tretí riadok
```

Program v druhom riadku vypíše iba znak konca riadku:

```
===== RESTART: reading2.py =====
'prvý riadok\n'
'\n'
'tretí riadok'
>>>
```

### Otázky:

6. Zistite, čo by sa stalo, ak by sme čítali zo súboru štvrtý neexistujúci riadok. Vyhlási pri tomto pokuse Python chybu, alebo niečo vypíše?
7. Mohli by sme náš program na čítanie troch riadkov napísať aj efektívnejšie?

Ak poznáme presný počet riadkov súboru, môžeme na prečítanie použiť aj for cyklus. Na čítanie nášho trojriadkového súboru môžeme použiť aj tento program:

```
subor = open('info1.txt', 'r')
for i in range(3):
    riadok = subor.readline()
    print(repr(riadok))
subor.close()
```

Častejšie budeme pracovať so súbormi s vopred neznámym počtom riadkov. Preto nebudem môcť použiť for cyklus s konkrétnym rozsahom (počtom opakovania). For cyklus môžeme zapísať aj v podobe, kde riadiaca premenná bude postupne obsahovať jednotlivé riadky a nemusíme ich čítať funkciou `subor.readline()`. Zapíšeme to takto:

```
subor = open('info1.txt', 'r')
for riadok in subor:
    print(repr(riadok))
subor.close()
```

Pri čítaní riadku za koncom súboru Python nehlási chybu, ale do premennej uloží prázdný reťazec. Uvedomme si, že po prečítaní ktoréhokoľvek existujúceho riadku nebude v premennej prázdný reťazec, lebo v reťazci bude minimálne '`\n`' - ukončenie riadku. Mohli by sme teda riadky súboru čítať aj nejakým iným cyklom dovtedy, kým neprečítame prázdný reťazec. My sme zatiaľ používali len časovač (timer). Ukážme si, ako by sme textový súbor s vopred neznámym počtom riadkov pomocou časovača:

```

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

def citaj():
    riadok = subor.readline()
    if riadok == '':
        subor.close()
    else:
        print(repr(riadok))
        canvas.after(10, citaj)

subor = open('info1.txt', 'r')
citaj()

```

### Otázky:

8. Koľkokrát sa vykoná funkcia `citaj()` v predchádzajúcom programe, ak by bol súbor `info1.txt` prázdny?
9. Čo sa zmení vo fungovaní predchádzajúceho programu, ak otvorenie súboru dáme priamo do funkcie `citaj()`?
10. Čo sa stane po spustení predchádzajúceho programu, ak súbor `info1.txt` bude obsahovať aj prázdne riadky?

Pri čítaní súboru nevieme čítať len jeden zadaný riadok (napríklad nevieme prečítať len piaty riadok zo súboru), ale musíme čítať postupne všetky riadky súboru. Takémučto čítaniu hovoríme aj sekvenčný prístup - k potrebnej informácii nepristupujeme priamo, ale postupne. Nasledujúci program najprv prečítaním celého súboru zistí počet riadkov v súbore a zapamätá si ho do premennej `pocet_riadkov`. Potom sa nás spýta, ktorý riadok chceme zobraziť (k-ty riadok). Následne opäť otvorí súbor na čítanie, prečíta `k` riadkov a len posledný k-ty riadok vypíše.

```

subor = open('score.txt', 'r')
pocet_riadkov = 0
for riadok in subor:
    pocet_riadkov += 1
print('Počet riadkov v súbore je:' + str(pocet_riadkov))
k = int(input('Napiš číslo riadku, ktorý mám vypísať:'))

subor = open('text.txt', 'r')
for i in range(k):
    riadok = subor.readline()
print(repr(riadok))
subor.close()

```

Súbor nemusíme čítať viackrát, ak si ho celý zapamäťame do nejakej premennej. No nie vždy vieme, s akým veľkým súborom budeme pracovať. Pamäťanie si celého súboru v premennej (alebo nejakej štruktúre) znamená, že je jeho obsah uložený v operačnej pamäti. Veľkosť operačnej pamäte býva obmedzená a často výrazne menšia ako veľkosť disku, na ktorom je uložený súbor. Teda nevýhodou uloženia do operačnej pamäte je jej obmedzená veľkosť, no naopak výhodou je rýchlosť prístupu k nej. Ak veľký textový súbor viackrát čítame od začiatku, môže byť nás program pomalý, lebo čítanie z disku je výrazne pomalšie ako prístup do operačnej pamäte. Jeden spôsob riešenia môže byť náročný na pamäť - tomu hovoríme, že má nároky na priestor - má väčšiu priestorovú zložitosť. Druhý spôsob môže byť menej efektívny vzhľadom na čas, čiže môže mať väčšiu časovú zložitosť.

Nasledujúci program si zapamätá všetky riadky do n-tice s názvom `riadky (tuple)`. Po zadaní čísla riadku sa nám vypíše len tento riadok priamo z n-tice, čiže z operačnej pamäte, a už nemusíme znova čítať textový súbor.

```

subor = open('score.txt', 'r')
pocet_riadkov = 0

```

```

riadky = ()
for riadok in subor:
    pocet_riadkov += 1
    riadky = riadky + (riadok,)

subor.close()
print('Počet riadkov v súbore je:' + str(pocet_riadkov))
k = int(input('Napiš číslo riadku, ktorý mám vypísať:'))
riadok = riadky[k-1]
print(repr(riadok))

```

### Otázky:

11. Prečo je v programe `riadky[k-1]`, keď chceme vypísať riadok `k`?
12. Ako môžeme zmeniť tento program, ak nechceme použiť premennú na počítanie riadkov, ale chceme vypísať informáciu o počte riadkov?

Všimnime si, že v premennej riadky sú zapísané všetky riadky súboru.

```

>>> riadky
('High score\n', '=====\\n', 'Alena 120\\n', 'Bob 115\\n', 'Charles 80\\n',
'Dana 75\\n', 'Edo 70\\n')
>>>

```

Ked' ju vypíšeme v shelli, zistíme, že každý riadok má aj ukončovací znak '`\n`'. Niekedy sa potrebujeme týchto znakov zbaviť. Na to nám slúži metóda `strip()`. Okrem '`\n`' odstráni z reťazca aj medzery na začiatku a na konci reťazca.

```

>>> s = ' textový     reťazec      \\n'
>>> s.strip()
'textový     reťazec'
>>>

```

Pozrime si upravený program:

```

subor = open('score.txt', 'r')
riadky = ()
for riadok in subor:
    riadky = riadky + (riadok.strip(),)

subor.close()
print('Počet riadkov v súbore je:' + str(len(riadky)))
k = int(input('Napiš číslo riadku, ktorý mám vypísať:'))
print(repr(riadky[k-1]))

```

N-tica `riadky` teraz neobsahuje '`\n`:

```

>>> riadky
('High score', '=====', 'Alena 120', 'Bob 115', 'Charles 80', 'Dana 75',
'Edo 70')
>>>

```

### Úlohy:

- 6** V textovom súbore `prihlaseni.txt` máme zoznam osôb, ktoré sa prihlásili na koncoročný výlet. Informácie o jednej osobe sú uložené v dvoch riadkoch. Na prvom z dvojice riadkov je meno a na druhom je vek osoby. Ukážka súboru:

Martina  
16

Roman

14

Adam

15

Jana

14

Mária

15

Erik

15

Vytvorte program, ktorý:

- a) vypíše počet prihlásených osôb,
- b) vypíše počet osôb, ktoré majú menej ako 15 rokov,
- c) po zadaní ceny cestovného na jednu osobu vypočíta a vypíše cestovné pre všetky osoby spolu, pričom osobám mladším ako 15 rokov uplatní zľavu 50%,
- d) vypočíta a vypíše priemerný vek osôb.

7

V obchodnom centre je pri východe dotykový displej, na ktorom môžeme vyjadriť svoju spokojnosť / nespokojnosť s poskytnutými službami. V textovom súbore `spokojnosť.txt` sú uložené hlasovania zákazníkov o spokojnosti s poskytnutými službami. Na jednom riadku je hlasovanie práve jedného zákazníka. Bud' je tam áno alebo nie. Ukážka súboru:

áno  
áno  
nie  
áno

Vytvorte program, ktorý:

- a) spočíta a vypíše celkový počet hlasujúcich,
- b) graficky zobrazí výsledok hlasovania pomocou stĺpcov pre hodnoty áno a nie,
- c) vypočíta a zobrazí percento spokojných a percento nespokojných zákazníkov.

8

V textovom súbore `kreslenie.txt` sú uložené súradnice naklikaných krúžkov v grafickej ploche (súradnice ich stredu). Pričom súradnica x je na samostatnom riadku súboru a na ďalšom riadku je súradnica y. Ukážka súboru:

100  
200  
57  
84  
251  
238

Vytvorte program, ktorý postupne v pravidelných časových intervaloch nakreslí v grafickej ploche krúžky podľa súradníc v súbore.

9

Vytvorte program, ktorý číta a postupne zobrazuje v grafickom plátnе jednotlivé riadky textu z textového súboru (`text.txt`). Po každom zobrazení riadku je jednosekundová pauza.

10

Upravte program z predchádzajúcej úlohy tak, aby zvládal aj výpis dlhších textov. Keď sa textom zaplní grafická plocha (sami si stanovte maximálny počet vypísaných riadkov), po sekunde sa všetok text zmaže a výpis plynule pokračuje.

11

V textovom súbore `citaty.txt` sú na nepárných riadkoch napísané citáty známych osobností a na párnych riadkoch ich autori. Vytvorte program, ktorý zobrazí citát na grafickej ploche a pod ním sa inou farbou zobrazí meno autora citátu. Používateľ po stlačení jedného z klávesov Enter / Space / PageDown zobrazí ďalší citát zo súboru.

Za dvadsať rokov budeš viac sklamaný z vecí, ktoré si neurobil, než z tých, ktoré si urobil.

Mark Twain

## 3.4 Iné spôsoby čítania textového súboru

### Cyklus s podmienkou (while cyklus)

V kapitole 3.3 sme čítali textový súbor pomocou časovača. Keď sme prišli na koniec súboru, v premennej **riadok** sme po prečítaní riadku získali prázdný reťazec. A tak sme vytvorili podmienku na naplánovanie ďalšieho volania časovača iba v prípade, že prečítaný reťazec nie je prázdnec.

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

def citaj():
    riadok = subor.readline()
    if riadok == '':
        subor.close()
    else:
        print(repr(riadok))
        canvas.after(10, citaj)

subor = open('info1.txt', 'r')
citaj()
```

Niekedy nám môže byť užitočný cyklus **while**. Tento cyklus sa používa v prípade, že vopred nepoznáme počet opakovaní, ale vieme definovať podmienku pokračovania cyklu (v prípade, že je podmienka pravdivá, cyklus sa vykoná znova). Je to podobné ako s časovačom, zásadný rozdiel je v tom, že medzi jednotlivými opakovaniami cyklu nečakáme. Cyklus **while** zvykneme označovať aj ako cyklus s podmienkou na začiatku. Takto zapisujeme cyklus **while**:

```
while podmienky:
    príkaz1
    príkaz2
    .
    .
```

V tejto situácii nepoznáme počet opakovaní, ale vieme definovať podmienku pokračovania cyklu. Tu je ukážka programu, ktorý sme pôvodne riešili pomocou časovača, prepísaná na cyklus **while**.

```
subor = open('info1.txt', 'r')
riadok = subor.readline()
while riadok != '':
    print(repr(riadok))
    riadok = subor.readline()
subor.close()
```

#### Otázky:

13. Čo bude robiť nasledovný program? Koľkokrát sa zopakuje while cyklus? Môže sa stať, že sa nevykoná ani raz?

```
from random import *
sucet = 0
```

```

print('Súčet je: ', sucet)
while sucet < 30:
    cislo = randrange(10)
    print('Pričítal som číslo: ', cislo)
    sucet += cislo
    print('Súčet je: ', sucet)

```

14. Čo bude robiť nasledovný program? Koľkokrát sa vykoná while cyklus? Za akých okolností spadne program a vyhlási chybu?

```

sucet = 0
pocet = 0
znamka = int(input('Zadaj známku:'))
while 0 < znamka < 6:
    pocet += 1
    sucet += znamka
    znamka = int(input('Zadaj známku:'))
print('Počet zadaných známok: ', pocet)
print('Priemer známok:', sucet/pocet)

```

15. Čo bude robiť nasledovný program? Koľkokrát sa vykoná cyklus while?

```

from random import *
znamka = randrange(1, 6)
pocet = 0
sucet = 0
while znamka < 6:
    pocet += 1
    sucet += znamka
    print('Pripočítal som známku:', znamka)
    print('Aktuálny počet známok:', pocet)
    znamka = randrange(1, 6)
print('Počet zadaných známok: ', pocet)
print('Priemer známok:', sucet/pocet)

```

16. Za akých okolností cyklus while:

- a) sa nevykoná ani raz,
- b) nikdy neskončí opakovanie?

For cyklus, v ktorom využívame `range`, vieme veľmi ľahko prepísať na `while` cyklus. Napríklad:

```

for i in range(1, 11):
    print('*'*i)

i = 1
while i < 11:
    print('*'*i)
    i += 1

```

### Úlohy:

**12** Upravte riešenie niektoréj z úloh s for cyklom a nahradťte for cyklus cyklom while.

## Konštrukcia with

Pri práci so súbormi sa môžeme stretnúť aj s konštrukciou `with`. Pri takomto zápisе práce so súborom nie je potrebné súbor zatvoriť. Python ho zatvorí automaticky po skončení konštrukcie `with`. Všetky príkazy konštrukcie `with` sú odsunuté a podľa odsunutia vie Python určiť, že patria k tejto konštrukcii (podobne ako `for, def, if, ...`). Konštrukciu `with` zapíšeme takto:

```
with open(...) as premenná:
    príkaz1
    príkaz2
    .
    .
```

Nasledujúci program prečíta a vypíše celý súbor **score.txt**:

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

Nasledujúci program pridáva texty zadané v **entry** na koniec súboru. Kedže používa konštrukciu **with**, nie je potrebné zatvárať súbor.

```
import tkinter

def save():
    with open('text1.txt', 'a') as subor:
        subor.write(entry1.get()+'\n')

entry1 = tkinter.Entry()
entry1.pack()
button1 = tkinter.Button(text = 'zapiš', command = save)
button1.pack()
```

### Úlohy:

#### 13

V textovom súbore **dialog.txt** je pripravený dialóg maximálne piatich postáv v divadelnej hre. Na nepárnom riadku je číslo postavy, ktorá hovorí, a na párnom (nasledujúcom) riadku je replika, ktorú povie. Napíšte program, ktorý postavám pridelí vami určené farby a vypíše do grafického plátna obsah súboru s použitím farieb jednotlivých postáv.

Ahojte!

Nepôjdeme všetci traja zajtra do kina?

A čo dávajú?

Neviem, chcela som ísť na bicykel s Martinou.

Môžete ísť v nedeľu, má byť krajšie počasie.

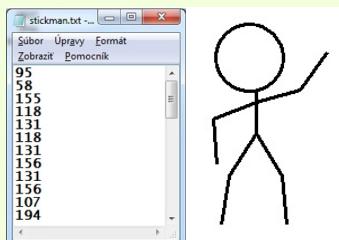
Ešte som nepozeral program.

Dávno som nevidel nejaké sci-fi.

Ale nech to nie je veľmi strašidelné.

#### 14

V pripravenom textovom súbore **stickman.txt** je na každom riadku jedno číslo. Štvorce riadkov tak popisujú jednotlivé časti kreslenej postavičky – Stickmana. Prvá štvorica čísel popisuje hlavu podobne ako pri metóde `create_oval` a každá ďalšia štvorica predstavuje jednu čiaru Stickmana (súradnice začiatočného a koncového bodu čiary). Napíšte program, ktorý vykreslí Stickmana.



#### 15

Vytvorte anketový program, ktorý zobrazuje výsledky ankety. V textovom súbore **anketa.txt** je na prvom riadku anketová otázka a na ďalších troch riadkoch čísla predstavujúce počty hlasov za „Áno“, „Nie“ a „Neviem“.

a) Program zobrazí otázku a stĺpcový graf predstavujúci počty odpovedí.

b) V programe budú tri tlačidlá umožňujúce hlasovať za jednu z možností tak, aby sa graf aktualizoval.

Nové počty hlasov budú zároveň uložené, aby sa mohli zobrazíť pri ďalšom spustení programu.

c) Možnosť s najväčším počtom hlasov bude v grafe zvýraznená inou farbou.

- d) V grafe bude informácia, koľko percent respondentov hlasovalo za danú možnosť.  
e) Trojica možných odpovedí nebude len Áno / Nie / Neviem. Názvy jednotlivých troch možností budú zapísané v súbore hneď pod anketovou otázkou.  
f) Hlasovať sa nebude dať len klikaním na tlačidlá, ale aj kliknutím priamo na možnosť.

Ktoré ovocie máte najradšej?



## 3.5 Práca s viacerými textovými súbormi

V textovom súbore sa nedá prepísať (zmeniť) nejaký riadok. Nemôžeme v ňom zároveň čítať a zapisovať. Môžeme ale jeden súbor otvoriť na čítanie, jeho obsah postupne čítať a zapisovať ho do nového súboru.

```
subor1 = open('score.txt', 'r')
subor2 = open('score2.txt', 'w')
for riadok in subor1:
    subor2.write(riadok)
subor1.close()
subor2.close()
```

### Otázky:

17. Doposiaľ sme pri zapisovaní riadkov do súboru pridávali na koniec aj '\n'. Prečo sme to v tomto prípade neurobili?

Asi je zbytočné vytvárať rovnakú kópiu súboru. Jednotlivé riadky môžeme aj zmeniť. Tento program vytvorí kópiu súboru tak, že riadky súboru budú očíslované.

```
subor1 = open('score.txt', 'r')
subor2 = open('score2.txt', 'w')
pocet = 0
for riadok in subor1:
    pocet += 1
    subor2.write(str(pocet) + '. ' + riadok)
subor1.close()
subor2.close()
```

### Ukážka súboru **score2.txt**:

```
1. High score
2. ======
3. Alena 120
4. Bob 115
5. Charles 80
6. Dana 75
7. Edo 70
```

### Úlohy:

- 16** Textový súbor obsahuje okrem textu aj prázdne riadky. Pomocou programu vytvorte kópiu tohto súboru, ktorá neobsahuje prázdne riadky, ale ani medzery na začiatku a na konci neprázdných riadkov.

17

V textovom súbore `prihlaseni.txt` máme zoznam osôb, ktorí sa prihlásili na koncoročný výlet. Informácie o jednej osobe sú uložené v dvoch riadkoch. Na prvom je meno a na druhom je vek osoby. Ukážka súboru:

```
Martina  
16  
Roman  
14
```

Vytvorte program, ktorý vytvorí textový súbor, kde budú iba mená prihlásených.

18

V textovom súbore `kreslenie.txt` sú uložené súradnice naklikaných krúžkov v grafickej ploche (súradnice ich stredu). Pričom súradnica x je na samostatnom riadku súboru a na ďalšom riadku je súradnica y. Ukážka súboru:

```
100  
200  
57  
84
```

Programom vytvorte takú kópiu súboru, kde x-ová a y-ová súradnica jedného krúžku bude na jednom riadku súboru a budú oddelené práve jednou medzerou.

Aj pri práci s viacerými súbormi môžeme použiť konštrukciu `with`.

```
with open('score.txt', 'r') as vstup:  
    with open('score3.txt', 'w') as vystup:  
        pocet = 0  
        for riadok in vstup:  
            pocet += 1  
            vystup.write(str(pocet) + '. ' + riadok)
```

V ďalšej ukážke z jedného textového súboru `správa.txt` vytvoríme dva nové tak, že na striedačku zapíšeme jeden riadok do súboru `riadky1.txt` a druhý riadok uložíme do súboru `riadky2.txt`. Ukážka správy:

```
3. máj 2017 o 13:53 sme.sk  
Urýchľovač sa znova rozbehol, hľadanie novej fyziky  
odštartoval aj Slovák. Po vianočnej prestávke spustili  
Veľký hadrónový urýchľovač. Detektor ATLAS strážil  
Tomáš Blažek.  
BRATISLAVA. Po vianočnej prestávke sa prvý raz rozbehol  
najväčší urýchľovač častic na svete. Do Veľkého hadrónového  
urýchľovača (LHC) sa koncom apríla podarilo vstreknúť  
prvé zväzky protónov, ktoré vďaka opatrnému ovládaniu  
postupne dosahovali jednotlivé sektory, až nakoniec  
úspešne prešli celým obvodom kruhového urýchľovača.
```

Program:

```
vstup = open('správa.txt', 'r')  
vystup1 = open('riadky1.txt', 'w')  
vystup2 = open('riadky2.txt', 'w')  
kam = 0  
riadok = vstup.readline()  
while riadok != '':  
    if kam == 0:  
        vystup1.write(riadok)  
    else:  
        vystup2.write(riadok)  
    kam = (kam + 1)%2 # % 2 vypočíta zvyšok po delení dvomi  
    riadok = vstup.readline()
```

```
vstup.close()  
vystup1.close()  
vystup2.close()
```

Ukážka **riadky1.txt**:

```
3. máj 2017 o 13:53 sme.sk  
odštartoval aj Slovák. Po vianočnej prestávke spustili  
Tomáš Blažek.  
najväčší urýchlovač častic na svete. Do Veľkého hadrónového  
prvé zvázky protónov, ktoré vďaka opatrnému ovládaniu  
úspešne prešli celým obvodom kruhového urýchlovača.
```

Ukážka **riadky2.txt**:

```
Urýchlovač sa znova rozbehol, hľadanie novej fyziky  
Veľký hadrónový urýchlovač. Detektor ATLAS strážil  
BRATISLAVA. Po vianočnej prestávke sa prvý raz rozbehol  
urýchlovača (LHC) sa koncom apríla podarilo vstreknúť  
postupne dosahovali jednotlivé sektory, až nakoniec
```

Všimnite si, že názov súboru je textový reťazec. Pri otváraní súboru môžeme použiť aj premennú. Napríklad používateľ nám zadá meno súboru a program otvorí zadaný súbor. Program na rozdeľovanie správy môže začínať aj takto:

```
nazov_suboru = input('Zadaj celý názov vstupného súboru: ')  
vstup = open(nazov_suboru, 'r')
```

#### Otázky:

18. Ako inak môžeme zapísť v programe riadok `kam = (kam + 1)%2`?
19. Ak porovnáme počet riadkov v súbore `riadky1.txt` a `riadky2.txt` pri akejkolvek dĺžke súboru `správa.txt`, čo bude vždy platit?
20. Čo bude robiť program, ak nebudeme mať súbor `správa.txt` a ak bude súbor prázdný?

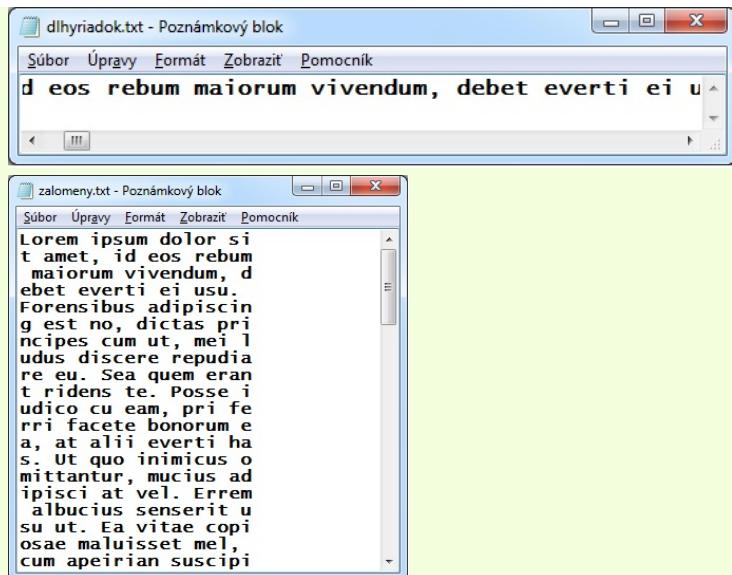
#### Úlohy:

**19** Vytvorte program, ktorý spojí textové súbory `riadky1.txt` a `riadky2.txt` do pôvodného súboru `správa.txt`.

**20** Alena si povedala, že rozdelenie riadkov správy do dvoch súborov správu veľmi neutají a lepšie bude, ak ju rozdelíme po znakoch. Vytvorte program, ktorý:  
a) zo súboru `správa.txt` vytvorí súbory `znaky1.txt` a `znaky2.txt` tak, že jednotlivé znaky v riadku bude striedavo ukladať do týchto súborov.  
b) zo súborov `znaky1.txt` a `znaky2.txt` vytvorí pôvodnú správu.

**21** Vytvorte program, ktorý zašifruje textový súbor Cézarovou šifrou do nového textového súboru. (Stačí, keď bude šifrovať len znaky bez diakritiky z malej abecedy).

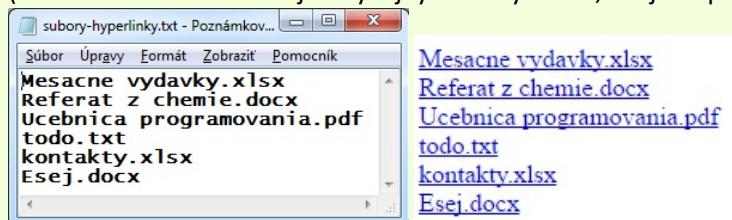
**22** V textovom súbore `dlhyriadok.txt` je zapísaný iba jeden veľmi dlhý riadok. Napíšte program, ktorý od používateľa načíta jedno kladné celé číslo – dĺžku riadku a vytvorí nový súbor, v ktorom bude pôvodný dlhý riadok pozalamovaný na danú dĺžku. Výstupný súbor teda bude obsahovať viacero riadkov, pričom každý (okrem posledného) bude mať dĺžku, ktorú zadal užívateľ. Výsledné riadky môžu byť zalomené aj v strede slov.



**23**

V textovom súbore dokumenty.txt je uvedený zoznam dokumentov v jednom priečinku (na každom riadku je jeden názov dokumentu aj s príponou). Vytvorte program, ktorý vygeneruje HTML súbor s hyperlinkami na jednotlivé dokumenty.

(Poznámka: HTML súbor je obyčajný textový súbor, iba jeho prípona nie je .txt, ale .html)



**24**

V pripravenom textovom súbore je clanok24.txt článok napísaný iba malými písmenami. Napíšte korektorský program, ktorý vytvorí nový súbor. V ňom budú aspoň začiatočné písmená viesť zmenené na veľké písmená. Vety v článku sú zakončené bodkou, výkričníkom alebo otáznikom (a za nimi nasleduje jedna medzera). V jednom riadku môže byť aj viac viet a nemusia končiť presne na konci riadku, ale môžu presahovať aj na ďalší riadok.

# 4 Funkcie s návratovou hodnotou

Vytvoriť funkciu, volať funkciu a používať parametre už vieme. Funkcie, ktoré sme používali, boli podprogramom, ktorý vykonal nejakú činnosť, ale zatiaľ nevrátil na výstupe funkcie žiadny výsledok. Vrátením výsledku nemyslíme to, keď vytvorený podprogram napríklad vypísal príkazom `print()` vypočítanú informáciu. S používaním funkcií s výsledkom sme sa už však stretli. Napríklad funkcie: `randrange()`, `randint()`, `ord()`, `str()`, `int()`, `len()`, `input()`, `choice()`, `type()` a pod..

## Otázky:

1. Ako používame spomenuté funkcie v programe? V čom sa líši ich spôsob použitia od funkcií, ktoré sme sami vytvárali?

Ked' sme sami vytvorili funkciu (bez návratovej hodnoty), napríklad funkciu `dom()`, ktorú sme použili na nakreslenie domu v canvase, zavolali sme ju v tvare `dom(parametre)`. Funkcia nevrátila žiadny výsledok, a teda sme nič nemuseli spracovať. Ale napríklad funkcia `randrange()` je funkcia s návratovou hodnotou a po zavolaní funkcie vráti funkcia náhodné číslo, ktoré treba nejako spracovať. Čiže výsledok si zapamätáme do nejakej premennej, alebo ho podhodíme ako vstup pre inú funkciu v podobe parametra. Funkciu `randrange(číslo)` sme používali takto:

- a) výsledok sme si zapamäタli do premennej: `a = randrange(10)`,
- b) výsledok sme použili ako vstup pre ďalšiu funkciu alebo výpočet: `print(randrange(10)),  
2*randrange(10), canvas.create_text(randrange(10), y, text = '...')`

Na určenie výstupu sa vo funkcií používa príkaz `return`. Príkaz `return` ukončí výpočet a uvedená hodnota sa stáva výsledkom funkcie.

Vytvorme funkciu, ktorá nám vráti náhodné písmeno malej abecedy s názvom `randchar()`.

```
from random import *

def randchar():
    cislo = randint(97, 122)
    znak = chr(cislo)
    return znak

print(randchar())
```

Volanie funkcie použijeme ako vstup pre funkciu `print()`, ktorá výsledok vypíše. Funkciu môžeme ešte upraviť, v príkaze `return` sa môže nachádzať priamo celý výpočet, a nie je potrebné použiť premenne `cislo` a `znak`. Funkciu použijeme na vytvorenie 10-znakového reťazca náhodných písmen malej abecedy.

```
from random import *

def randchar():
    return chr(randint(97, 122))

s = ''
for i in range(10):
    s = s + randchar()

print(s)
```

Môžeme vytvoriť funkciu `randretazec(počet_znakov)`, ktorá nám vráti reťazec z náhodných písmen malej abecedy. Dĺžku reťazca určíme vstupným parametrom.

```
from random import *

def randchar():
```

```

        return chr(randint(97, 122))

def randretazec(dlzka):
    s = ''
    for i in range(dlzka):
        s = s + randchar()
    return s

print(randretazec(5))
heslo = randretazec(8)
print(heslo)

```

### Otázky:

2. Ako bude fungovať program, ak by sme vo funkcií `randretazec()` umiestnili príkaz `return` ešte pred `for` cyklus, alebo by bol súčasťou `for` cyklu?

```

def randretazec(dlzka):
    s = ''
    return s
    for i in range(dlzka):
        s = s + randchar()

def randretazec(dlzka):
    s = ''
    for i in range(dlzka):
        s = s + randchar()
    return s

```

Nesmieme zabúdať, že príkaz `return` ukončí vykonávanie funkcie a vráti výsledok. To znamená, že nemá zmysel, aby za príkazom `return` boli ďalšie príkazy, lebo tie sa už nevykonajú. Ďalšie príkazy vo funkcií majú zmysel, iba ak sa príkaz `return` nevykoná vždy, ale len pri splnení nejakej podmienky. Vytvoríme vlastnú funkciu na výpočet absolútnej hodnoty zo zadaného čísla:

```

def absolutnahodnota(cislo):
    if cislo > 0:
        return cislo
    else:
        return -1*cislo

print(absolutnahodnota(-10))
print(absolutnahodnota(10))

```

V skutočnosti môžeme funkciu `absolutnahodnota()` zapísat aj jednoduchším spôsobom:

```

def absolutnahodnota(cislo):
    if cislo < 0:
        cislo = -1*cislo
    return cislo

```

### Otázky:

3. Čo sa stane, ak vstupom do funkcie `absolutnahodnota()` nebude číslo?

Takto môžeme skúsiť vyriešiť problém so zadáním iného vstupu ako vstupu typu `int`:

```

def absolutnahodnota(cislo):
    if type(cislo) == int:
        if cislo < 0:

```

```

        cislo = -1*cislo
        return cislo
    
```

### Otázky:

4. Je predchádzajúce riešenie funkcie `absolutnahodnota()` rozumné? Môže nám z nejakého záporného čísla vrátiť záporné číslo?
5. Čo nám vrátia nasledujúce volania?
  - a) `absolutnahodnota(-10/2)`
  - b) `absolutnahodnota(' -5 ')`
  - c) `absolutnahodnota(-10//2) # // je celočíselné delenie`
6. Ako môžeme funkciu `absolutnahodnota()` vylepšiť?

Vytvoríme funkciu `xnay(x, y)`, ktorá číslo `x` umocní na `y`. Dajme tomu, že nepoznáme operáciu `**`.

```

def xnay(x,y):
    vysledok = 1
    for i in range(y):
        vysledok = vysledok * x
    return vysledok

print(xnay(2, 4))
print(xnay(2, 1))
print(xnay(2, 0))
    
```

### Otázky:

7. Je táto funkcia `xnay()` dobre vytvorená? Pre ktoré vstupy bude počítať správne výsledky, pre ktoré nesprávne výsledky, za akých okolností bude hlásiť chybu?

Niekedy potrebujeme, aby funkcia vrátila viac ako jeden výsledok. Vtedy sa použije na výstupe n-tica. My už vieme, že na zapísanie n-tice nie sú vždy potrebné zátvorky, preto nám bude fungovať takýto zápis: `return hodnota1, hodnota2, hodnota3, ...`

Vytvoríme funkciu `rozdelmeno(vstup)`, ktorá rozdelí vstup - meno zadané v tvare `meno medzera priezvisko` na n-ticu (meno, priezvisko):

```

def rozdelmeno(vstup):
    pozicia = vstup.find(' ')
    meno = vstup[:pozicia]
    priezvisko = vstup[pozicia+1:]
    return meno, priezvisko

meno, priezvisko = rozdelmeno('Jana Šikovná')
print(meno)
print(priezvisko)
    
```

alebo celú n-ticu priradíme len do jednej premennej:

```

info = rozdelmeno('Jana Šikovná')
print(info)
    
```

Vypíše sa `('Jana', 'Šikovná')`.

Doposiaľ sme na výpočty používali bežne operácie `+`, `-`, `*`, `/`. Niekedy nám budú užitočné aj ďalšie operácie: `//`, `%` a `**`. Operácia `//` je celočíselné delenie (v niektorých jazykoch označovaná aj ako div). Napríklad `7 // 3 = 2`. Operácia `%` vypočítava zvyšok po delení, napríklad `7 % 3 = 1` (zvyškom po delení sedem troma je jeden). Operácia `**` sa používa na umocnenie. Napríklad `2 ** 3 = 8` (dva na tretiu).

## Úlohy:

- 1** Upravte funkciu `rozdeleno` tak, aby fungovala aj v prípade, že celé meno obsahuje viac krstných mien oddelených medzerou a potom priezvisko. Napríklad rozdelí meno 'Adam Ján Múdry'.
- 2** Vytvorte funkciu, ktorá zo zadaného reťazca vyhodí samohlásky.
- 3** Vytvorte funkciu, ktorá na zadanom mieste nakreslí domček so strechou a na výstupe vráti n-ticu, ktorá bude obsahovať identifikátory nakreslených útvarov.
- 4** Vytvorte funkciu, ktorej na vstupe zadáme dva reťazce a výstupom bude vstupný reťazec, ktorý má väčšiu dĺžku.
- 5** Vytvorte funkciu `isEmail`, ktorá overí, či string zadaný vo vstupnom parametri má tvar emailovej adresy (teda `niečo@niečo.niečo`).
- 6** Vytvorte funkciu `delitele`, ktorá vráti n-ticu so všetkými deliteľmi čísla zadaného v parametri.

Pomocou operácie `//` a `%` môžeme získavať cifry z čísla bez toho, aby sme číslo premenili na `string`.

```
cislo = int(input('Zadaj číslo:'))

def posledna_cifra(cislo):
    cifra = cislo % 10
    return cifra

print(posledna_cifra(cislo))
```

Funkciu upravíme. Na výstupe nám vráti okrem poslednej cifry aj zvyšnú cifernú časť čísla, z ktorej ešte chceme postupne odtrhnúť posledné cifry.

```
cislo = int(input('Zadaj číslo:'))

def posledna_cifra(cislo):
    cifra = cislo % 10
    cislo = cislo // 10
    return cifra, cislo

cifra, cislo = posledna_cifra(cislo)
print(cifra)
print(cislo)
```

Teraz stačí v cykle viackrát odtrhnúť a vypísať poslednú cifru čísla (vždy nového) a môžeme vypísať jednotlivé cifry. Problémom však je, ako určíme počet opakovania cyklu. Napríklad, ak by sme vytvorili `for cyklus` s desiatimi opakovami, program bude dobre fungovať pre 10-ciferné čísla, ale nie pre akékoľvek. Áno, mohli by sme číslo premeniť na string a funkciou `len()` zistiť počet znakov, ale práve tejto konverzii sa chceme vyhnúť. Vedeli by sme to vyriešiť aj použitím `časovača` (timera), čiže cyklu, ktorý nepotrebuje vopred poznáť počet opakovania. No časovač môže spomaliti riešenie. Vhodnejšie je použiť `while cyklus`, ktorý už poznáme z kapítoly o textových súboroch.

A takto rozložíme číslo na jednotlivé cifry pomocou `while cyklu`:

```
cislo = int(input('Zadaj kladné celé číslo:'))

def posledna_cifra(cislo):
    cifra = cislo % 10
    cislo = cislo // 10
    return cifra, cislo
```

```

while cislo > 0:
    cifra, cislo = posledna_cifra(cislo)
    print(cifra)

```

```

Zadaj kladné celé číslo:1234
4
3
2
1
>>>

```

Študenti vytvorili funkciu **palindrom(vstup)**, ktorá zistí, či zadaný reťazec je palindrom. Vo funkcii sa nemohlo použiť otočenie reťazca. Výstupom bude hodnota **True** alebo **False**. Palindrom je slovo, veta, číslo (pre nás reťazec), ktorý aj po otočení sprava doľava je rovnaký. Napríklad známe palindromy sú: anna, kobylamamalybok, jelenovipivonelej, ŤAHAŤ (ďalšie palindromy, aj v iných jazykoch, môžete nájsť na internete). Riešenia študentov nájdete v nasledujúcich otázkach.

### Otázky:

8. Ako by ste vytvorili túto funkciu vy?
9. Ktoré z funkcií sú dobrým riešením? Pri ktorých vstupoch funkcie nebudú dávať správny výsledok? Pri ktorých vstupoch funkcie vyhlási Python chybu?  
a)

```

def jepalindrom(vstup):
    z = 0
    k = len(vstup)-1
    while z < k:
        if vstup[z] == vstup[k]:
            vysledok = True
        else:
            vysledok = False
        z += 1
        k -= 1
    return vysledok

```

b)

```

def jepalindrom(vstup):
    z = 0
    k = len(vstup)-1
    while z <= k:
        if vstup[z] == vstup[k]:
            vysledok = True
        else:
            vysledok = False
        z += 1
        k -= 1
    return vysledok

```

c)

```

def jepalindrom(vstup):
    z = 0
    k = len(vstup)-1
    vysledok = True
    while z < k and vysledok:
        if vstup[z] != vstup[k]:
            vysledok = False
        z += 1
        k -= 1

```

```

        return vysledok

d)

def jepalindrom(vstup):
    z = 0
    k = len(vstup)-1
    vysledok = False
    while z < k:
        if vstup[z] == vstup[k]:
            vysledok = True
        z += 1
        k -= 1
    return vysledok

e)

def jepalindrom(vstup):
    z = 0
    k = len(vstup)-1
    vysledok = False
    while z <= k:
        if vstup[z] == vstup[k]:
            vysledok = True
        z += 1
        k -= 1
    return vysledok

f)

def jepalindrom(vstup):
    vysledok = True
    for i in range(len(vstup)//2):
        if vstup[i] != vstup[-i-1]:
            vysledok = False
    return vysledok

g)

def jepalindrom(vstup):
    vysledok = True
    for i in range(len(vstup)):
        if vstup[i] != vstup[-i-1]:
            vysledok = False
    return vysledok

h)

def jepalindrom(vstup):
    for i in range(len(vstup)):
        if vstup[i] == vstup[-i-1]:
            vysledok = True
        else:
            vysledok = False
    return vysledok

i)

def jepalindrom(vstup):
    vysledok = False
    for i in range(len(vstup)):
        if vstup[i] == vstup[-i-1]:
            vysledok = True
    return vysledok

```

Ak ste odhadli, ktoré riešenia sú dobré; dobré, ale málo efektívne; zlé, skúste ich otestovať napríklad na týchto vstupoch:

```
print(jepalindrom('jelenovipivonelej'))  
print(jepalindrom('abccxa'))  
print(jepalindrom('a'))  
print(jepalindrom(''))
```

### Úlohy:

7

Vytvorte program, ktorý prečíta textový súbor `vstup.txt` a vytvorí textový súbor `vystup.txt`, v ktorom budú len tie riadky zo vstupného súboru, ktoré sú palindromy. Na testovanie palindromu vytvorte vlastnú funkciu `jepalindrom`, ktorá bude podľa vás najlepším riešením. O kvalite riešenia navzájom diskutujte.

8

Vytvorte funkciu `cifernysucet(cislo)`, ktorá vypočíta ciferný súčet zadaného čísla na vstupe.

9

Prerobte niektoré vaše staršie jednoduché programy, ktoré spracujú vstupné hodnoty a vypíšu jednu alebo niekoľko výstupných hodnôt tak, aby využívali funkciu s návratovou hodnotou.

10

Vyberte si niekoľko vašich funkcií, ktoré majú vstupný parameter typu `string`, s ktorým niečo urobia, a vytvorte si vlastný modul `StringFunctions` – zbierku takýchto „stringových“ funkcií, ktoré potom môžete využívať v iných vašich programoch.

11

Vytvorte funkciu `rgb`, ktorá zo vstupných hodnôt v desiatkovej sústave namieša hodnotu farby v hexadecimálnom zápise a na výstupe vráti reťazec s farbou.

# 5 Práca s viacerými údajmi (zoznam)

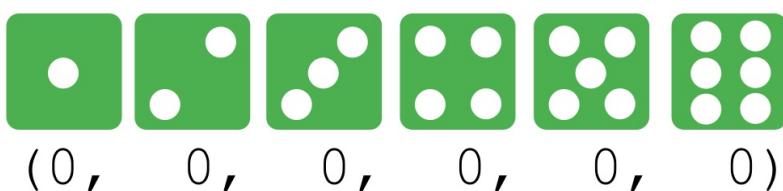
## 5.1 Hádzanie hracími kockami

Študenti dostali na matematike úlohu, aby zistili, ako často padajú jednotlivé čísla na hracej kocke. Kockou majú hodiť aspoň 1000-krát a potom nakresliť diagram. Soňa je šikovná programátorka, tak sa rozhodla, že vytvorí program, ktorý to bude simulovať. Najprv mala takýto nápad - v cykle 1000-krát vyžrebuje náhodné číslo z rozsahu 1 - 6 a potom si v premenných `počet1`, `počet2`, `počet3`, `počet4`, `počet5`, `počet6` bude pamätať počet padnutí jednotlivých čísel. Toto riešenie hned' zavrhla, lebo sa jej vôbec nepáčilo. Povedala si, že to musí vyriešiť krajšie.

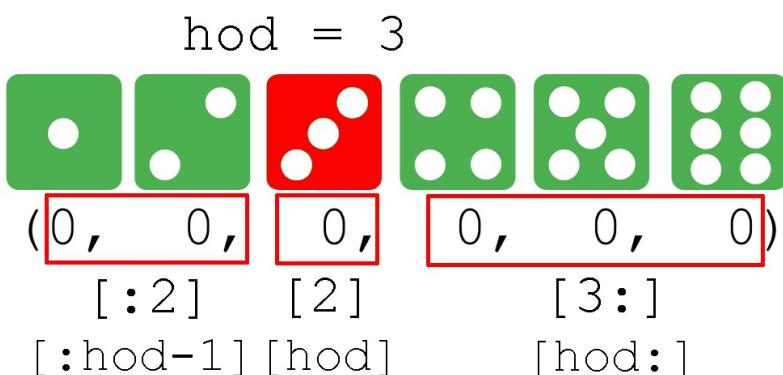
### Otázky:

1. Prečo jej prvy nápad nie je pekným riešením?
2. Ako by sme tento problém vyriešili krajšie len s použitím poznatkov, ktoré sme už získali?

Potom dostala ďalší nápad. Hodnoty premenných pre jednotlivé čísla na kocke spojí do n-tice a nemusí použiť komplikovaný `if`, ktorým zistovala, ktorú premennú treba zvýšiť.



Ked' počítač hodí **trojku**, na príslušnom mieste v n-tici zvýši číslo o jeden. Musí s použitím rezov vystrihnúť prvé dve čísla. Tretie číslo vystrihne a zvýši ho. Vystrihne zvyšok - od štvrtého čísla po koniec. Z týchto troch častí vytvorí novú n-ticu.



Nakoniec vytvorila takýto program:

```
pocty = (0,) * 6
for i in range(10):
    hod = randint(1, 6)
    pocet = pocety[hod-1]+1
    print('Už {}-krát padlo číslo {}'.format(pocet, hod))
    pocety = pocety[:hod-1]+(pocet,) + pocety[hod:]

for i in range(6):
    print('Číslo {} padlo {}-krát.'.format(i+1, pocety[i]))
```

Najprv testovala program s menším počtom opakovaní a až potom ich nastavila na 1000 pokusov. Pri 10 pokusoch videla takýto výpis:

```
Už 1-krát padlo číslo 4.  
Už 1-krát padlo číslo 6.  
Už 2-krát padlo číslo 4.  
Už 1-krát padlo číslo 2.  
Už 2-krát padlo číslo 6.  
Už 3-krát padlo číslo 4.  
Už 2-krát padlo číslo 2.  
Už 1-krát padlo číslo 5.  
Už 2-krát padlo číslo 5.  
Už 1-krát padlo číslo 3.  
Číslo 1 padlo 0-krát.  
Číslo 2 padlo 2-krát.  
Číslo 3 padlo 1-krát.  
Číslo 4 padlo 3-krát.  
Číslo 5 padlo 2-krát.  
Číslo 6 padlo 2-krát.
```

Soňa je s programom spokojná, až na jednu drobnosť. Vytváranie novej n-tice pomocou rezov sa jej zdalo komplikované. Chcela by jednoducho zmeniť hodnotu konkrétneho prvku. Zistila, že takúto vlastnosť má dátový typ **list**, niekde sa môžeme stretnúť aj s označením **zoznam** alebo **pole**.

Dátový typ **zoznam** je tiež postupnosť dát rôzneho typu (podobne ako n-tica). Veľkým rozdielom je, že **zoznam** je meniteľný typ na rozdiel od n-tice a textového reťazca. To znamená, že v ňom môžeme zmeniť hodnotu konkrétneho prvku len pomocou indexu, a nemusíme vytvárať nový **zoznam** pomocou rezov.

Zoznam (list, pole) vytvárame pomocou hranatých zátvoriek.

```
>>> pocty = [1, 2, 3, 0, 2, 4]  
>>> pocty  
[1, 2, 3, 0, 2, 4]  
>>> type(pocty)  
>class 'list'<  
>>> len(pocty)  
6  
>>> a = []  
>>> type(a)  
>class 'list'<  
>>> len(a)  
0
```

K jednotlivým prvkom zoznamu (listu) sa dá pristupovať priamym prístupom pomocou indexu a môžeme ním reprezentovať lineárnu dátovú štruktúru. To znamená, že má aj vlastnosti, ktoré mávajú v iných programovacích jazykoch dátové štruktúry označované ako pole (array). Z tohto dôvodu budeme zoznam (list) niekedy označovať aj názvom pole. Treba však upozorniť, že Python má aj dátovú štruktúru array, ktorá je akýmsi špeciálnym zoznamom, v ktorom prvkami môžu byť len reálne čísla.

### Otzáky:

3. Ktoré z funkcií a operácií používaných na prácu s reťazcami alebo n-ticami môžeme použiť na prácu so zoznamom (listom)?
4. Vyskúšajte indexovanie prvkov. Môžeme ho použiť aj na zoznam, je v niečom rozdiel?
5. Ako to bude s používaním rezov (slice)?
6. For cyklom sme mohli prechádzať prvkami reťazca. Funguje to aj pre zoznam?
7. Čo sa stane, ak budeme indexovať mimo zoznamu (na neexistujúci prvak)?

Takto môžeme upraviť program s kockami z n-tíc na zoznam:

```
pocety = [0]*6
for i in range(10):
    hod = randint(1, 6)
    pocet = pocety[hod-1]+1
    print('Už {}-krát padlo číslo {}'.format(pocet, hod))
    pocety[hod-1] += 1

for i in range(6):
    print('Číslo {} padlo {}-krát.'.format(i+1, pocety[i]))
```

Výpis bude rovnaký (len s inými náhodnými číslami). Nakoniec sme ešte vypísali celý zoznam **pocety**.

```
Už 1-krát padlo číslo 4.
Už 1-krát padlo číslo 3.
Už 1-krát padlo číslo 1.
Už 2-krát padlo číslo 1.
Už 2-krát padlo číslo 4.
Už 2-krát padlo číslo 3.
Už 1-krát padlo číslo 5.
Už 2-krát padlo číslo 5.
Už 1-krát padlo číslo 2.
Už 3-krát padlo číslo 4.
Číslo 1 padlo 2-krát.
Číslo 2 padlo 1-krát.
Číslo 3 padlo 2-krát.
Číslo 4 padlo 3-krát.
Číslo 5 padlo 2-krát.
Číslo 6 padlo 0-krát.
>>> pocety
[2, 1, 2, 3, 2, 0]
>>>
```

### Úlohy:

- 1** Upravte program tak, aby sme po spustní zadali počet simulovaných hodov a potom program simuluje taký počet hodov.
- 2** Upravte program tak, aby nám pri výpise hodu napísal aj číslo hodu.
- 3** Upravte program tak, aby postupnosť žrebovaných hodov uložil do súboru.

Vytvorme program, v ktorom budeme hádzať naraz troma hracími kockami. Spočítame súčet jedného hodu na všetkých troch kockách a zistíme, ako často padajú jednotlivé súčty.

### Otzázky:

8. Ktoré súčty môžeme získať pri hode troma kockami?
9. Čo si myslíte, aká bude frekvencia jednotlivých súčtov pri hode troma kockami v porovnaní s frekvenciou výskytu hodnôt pri hádzaní len jednou kockou? Svoju odpoveď zdôvodnite.
10. Akým spôsobom si môžeme pamätať frekvenciu jednotlivých súčtov?

Programu na vstupe zadáme počet simulovaných hodov a odštartujeme simuláciu. Program postupne simuluje hody a po každom hode vypíše histogram frekvencie výskytu jednotlivých súčtov.

```
import tkinter
from random import *
```

```

canvas = tkinter.Canvas(width = 600, height = 500, bg = 'white')
canvas.pack()

pocty = [0]*16

def kresli():
    canvas.delete('stlpce')
    for i in range(16):
        canvas.create_rectangle(50+i*30, 450, 50+i*30+20, 450-pocty[i],
                               fill = '#4cb050')

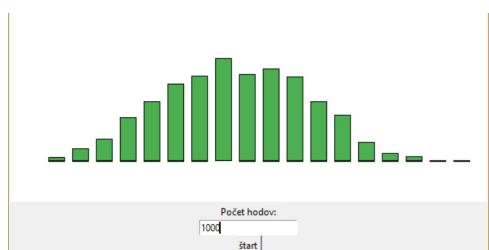
def simulacia():
    global pocet_hodov, pocty
    pocet_hodov -= 1
    sucet = 0
    for i in range(3):
        hod = randint(1, 6)
        sucet += hod
    pocty[sucet-3] += 1
    kresli()
    if pocet_hodov > 0:
        canvas.after(10, simulacia)

def start():
    global pocet_hodov
    pocet_hodov = int(entry1.get())
    simulacia()

label1 = tkinter.Label(text = 'Počet hodov:')
label1.pack()
entry1 = tkinter.Entry()
entry1.pack()
button1 = tkinter.Button(text = 'štart', command = start)
button1.pack()
pocet_hodov = 0

```

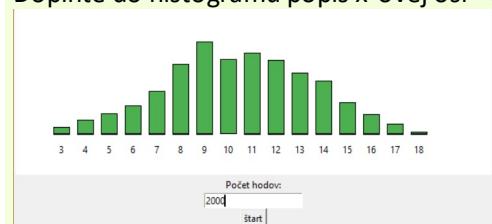
Takéto môže byť rozloženie frekvencie súčtov hodnôt pri hádzaní troma kockami po 1000 hodoch.



#### Úlohy:

4

Doplňte do histogramu popis x-ovej osi - označenie súčtov.



5

Doplňte do histogramu popis jednotlivých stĺpcov - nad stĺpcom zobrazte počet takých súčtov.



## 6

Priebeh simulácie uložte do textového súboru tak, že v jednom riadku budú medzerou oddelené hodnoty, ktoré padli pri jednom hode viacerými kockami.

```
5 4 3
2 4 2
1 2 6
4 2 2
```

Zatiaľ máme veľkosť canvasu vopred presne pripravenú. Program vylepšíme tak, že po zadaní počtu hodov sa zároveň prispôsobí výška canvasu. Predpokladáme (odhadujeme), že na najvyšší stĺpec budeme potrebovať približne jednu päťtinu z počtu hodov a tiež rezervu 50 pixelov na vypísanie označenia osi x a popis hodnôt stĺpcov. Vytvoríme globálnu premennú `maxy` a do funkcie `start()` doplníme výpočet maximálnej y-ovej súradnice `maxy = pocet_hodov//5+50` a zmeníme výšku canvasu `canvas['height'] = maxy`. Ihned po priradení sa veľkosť canvasu upraví.

**Poznámka:** Podobne vieme zmeniť aj šírku canvasu (`canvas['width'] = 400`), alebo farbu pozadia canvasu (`canvas['bg'] = 'red'`). Ak by sme, naopak, chceli zistiť aktuálnu veľkosť canvasu, vypíšeme ju napríklad cez `print(canvas['height'])`, alebo ju priradíme do premennej `vyska = canvas['height']`. Všimnime si, že výsledkom nie je číslo, ale textový reťazec, čiže budeme ho musieť funkciou `int()` transformovať na číslo.

Takto bude zapísaná upravená funkcia `start()`:

```
maxy = 500
def start():
    global pocet_hodov, maxy
    pocet_hodov = int(entry1.get())
    maxy = pocet_hodov//5+50
    canvas['height'] = maxy
    simulacia()
```

Nestačí upraviť iba túto funkciu, ale všade pri vykreslovaní stĺpcov a iných hodnôt musíme upraviť pôvodnú fixne zadanú y-ovú súradnicu tak, aby sa vypočítala podľa `maxy`.

```
canvas.create_rectangle(50+i*30, maxy-50, 50+i*30+20, maxy-50-pocty[i],
                      fill = '#4cb050')
```

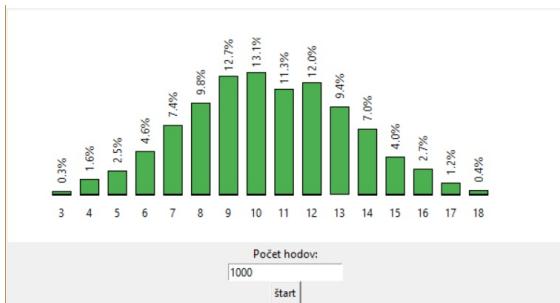
Popis jednotlivých stĺpcov môže obsahovať ich presnú hodnotu, ale môžeme ho vyjadriť aj v percentoch (zo všetkých hodov). K aktuálnemu percentuálnemu zastúpeniu jednotlivých súčtov potrebujeme poznáť aktuálny (už vykonaný) počet hodov. Zdá sa, že ten máme v premennej `pocet_hodov`, ale nie je to tak. V premennej `pocet_hodov` si pamäタame počet hodov, ktoré máme ešte vykonať, a nie už vykonaný počet hodov. Bud si vytvoríme novú premennú, v ktorej si budeme počítať počet už zrealizovaných hodov, alebo môžeme sčítať súčet všetkých prvkov zoznamu pomocou vstavanej funkcie `sum()`.

```
>>> pocty
[2, 7, 9, 26, 31, 40, 56, 72, 70, 49, 62, 36, 22, 11, 6, 1]
>>> sum(pocty)
500
>>>
```

Pre každý stĺpec vypočítame jeho pomer k celku a vynásobíme 100.

```
>>> pocety
[2, 7, 9, 26, 31, 40, 56, 72, 70, 49, 62, 36, 22, 11, 6, 1]
>>> percenta = [0]*16
>>> pocet_hodov_zrealizovany = sum(pocety)
>>> i = 0
>>> for hodnota in pocety:
    percenta[i] = hodnota / pocet_hodov_zrealizovany * 100
    i += 1
>>> percenta
[0.4, 1.4000000000000001, 1.7999999999999998, 5.2, 6.2, 8.0, 11.200000000000001,
14.399999999999999, 14.000000000000002, 9.8, 12.4, 7.199999999999999,
4.399999999999995, 2.199999999999997, 1.2, 0.2]
>>>
```

Pomocou formátovania reťazcov vieme desatinné číslo zobraziť napríklad na 1 desatinné miesto:  
`'{:.1f}'.format(cislo)`.



Výsledný program po úpravách:

```
import tkinter
from random import *
canvas = tkinter.Canvas(width = 600, height = 500, bg = 'white')
canvas.pack()

pocety = [0]*16
maxy = 500

def kresli():
    canvas.delete('stlpce')
    canvas.delete('popis')
    percenta = [0]*16
    pocet_hodov_zrealizovany = sum(pocety)
    i = 0
    for hodnota in pocety:
        percenta[i] = hodnota / pocet_hodov_zrealizovany * 100
        i += 1
    for i in range(16):
        canvas.create_rectangle(50+i*30, maxy-50, 50+i*30+20, maxy-50-pocety[i],
                               fill = '#4cb050')
        percento = '{:.1f}%'.format(percenta[i])
        canvas.create_text(50+i*30+10, maxy-50-pocety[i]-20, text = percento,
                           tags='popis', angle = 90)

def simulacia():
    global pocet_hodov, pocety
    pocet_hodov -= 1
    sucet = 0
    for i in range(3):
        hod = randint(1, 6)
        sucet += hod
```

```

pocety[sucet-3] += 1
kresli()
if pocet_hodov > 0:
    canvas.after(10, simulacia)

def start():
    global pocet_hodov, maxy
    pocet_hodov = int(entry1.get())
    maxy = pocet_hodov//5+50
    canvas['height'] = maxy
    for i in range(16):
        canvas.create_text(50+i*30+10, maxy-30, text = str(i+3))
    simulacia()

label1 = tkinter.Label(text = 'Počet hodov:')
label1.pack()
entry1 = tkinter.Entry()
entry1.pack()
button1 = tkinter.Button(text = 'štart', command = start)
button1.pack()
pocet_hodov = 0

```

### Otázky:

11. Je tento program efektívny? Môžeme v ňom niečo vylepšiť (iným spôsobom riešenia sa dopracovať k rovnakému efektu)?
12. Prečo je lepšie umiestniť vypisovanie popisu osi x vo funkcií `start`, a nie `kresli`?
13. Prečo bude program hlásiť chybu, ak by sme do funkcie `start` hneď pred volanie `simulacia` umiestnili volanie funkcie `kresli`? Akým spôsobom by sme mohli predísť tejto chybe?

### Úlohy:

**7** Doplňte do programu tlačidlo, ktoré bude prepínať, či nad stĺpcom bude zobrazený počet súčtov alebo vyjadrenie v percentoch (zo všetkých padnutých súčtov).

**8** Upravte program tak, že mu zadáme počet kociek, ktorými naraz hádzeme, a tiež počet simulovaných hodov. Program aj podľa tohto vstupu upraví (prispôsobí) priebežné zobrazenie histogramu. Šírku stĺpca prispôsobí veľkosťi canvasu a počtu stĺpcov (v závislosti od počtu použitých hracích kociek).

**9** Vytvorte program, ktorý vykreslí vedľa seba 20 štvorčekov náhodne vybraných farieb. Pri prechode myši ponad ne sa budú štvorčeky zväčšovať podľa ukážky.



**10** Vytvorte program, ktorý bude predstavovať animovaný „reklamný nápis“. Používateľ zadá slovo, ktoré sa následne vypíše do grafickej plochy, pričom každý znak zadaného slova bude vypísaný náhodne vybranou farbou.

- Farby písmen sa budú každú sekundu náhodne meniť.
- Farby písmen sa budú každú sekundu cyklicky „posúvať“ doľava. (viď ukážku)



## Spoločné vlastnosti textových reťazcov, n-tíc a zoznamu

**Indexovanie** - na prístup k danému prvku textových reťazcov, n-tíc aj zoznamu používame rovnaký spôsob **premenná[index]**.

**Rezy (slice)** - na výber viacerých prvkov (rez) textových reťazcov, n-tíc aj zoznamu používame rovnaký spôsob **premenná[odkial:pokial:krok]**.

**Operácia in** - funguje v textových reťazcoch, n-ticiach aj v zozname **prvok in premenná**.

**Funkcia len()** - funguje v textových reťazcoch, n-ticiach aj v zozname **len(premenná)**.

**Operácia +** - funguje v textových reťazcoch, n-ticiach aj v zozname - spojí obsah dvoch typovo rovnakých premenných do jednej rovnejšej štruktúry **a+b**.

**Operácia \*** - funguje v textových reťazcoch, n-ticiach aj v zozname - viacnásobne zopakuje obsah premennej do novej štruktúry **a\*3**.

**Prechádzanie for cyklom** - funguje v textových reťazcoch, n-ticiach aj v zozname - pomocou for cyklu môžeme prechádzať po jednotlivých prvkoch štruktúry - **for i in premenná**.

## 5.2 Užitočné funkcie a metódy na prácu so zoznamom

Rovnako ako textový reťazec a n-tica aj zoznam (list, pole) má v Pythone definované svoje metódy, čiže funkcie v špeciálnom zápise v tvare **meno\_zoznamu.funkcia(parametre)**. Zoznam je na rozdiel od textového reťazca a n-tice meniteľný typ, to znamená, že niektoré z týchto metód nevrátia výsledok, ale budú mať nejaký účinok na zoznam (niečo v ňom zmenia). Najčastejšie budeme používať tieto metódy:

**meno\_zoznamu.append(hodnota)** - pridá na koniec zoznamu prvak so zadanou hodnotou

**meno\_zoznamu.insert(index, hodnota)** - pridá nový prvak so zadanou hodnotou pred zadaný index zoznamu

**meno\_zoznamu.pop()** - odstráni posledný prvak zoznamu a vráti jeho hodnotu

**meno\_zoznamu.pop(0)** - odstráni prvý prvak zoznamu a vráti jeho hodnotu

**meno\_zoznamu.index(hodnota)** - vyhľadá hodnotu a vráti index prvého výskytu hodnoty v zozname

**meno\_zoznamu.count(hodnota)** - vráti počet výskytov hodnoty v zozname

**meno\_zoznamu.sort()** - utriedi prvky zoznamu vzostupne (priamo v pamäti)

V nasledujúcich dvoch ukážkach (v shelli) vidíme vytvorenie zoznamu z piatich náhodných čísel. Najprv využijeme metódu **append()**.

```
>>> from random import *
>>> cisla = []
>>> type(cisla)
<class 'list'>
>>> for i in range(5):
    cisla.append(randrange(10))

>>> cisla
[4, 8, 5, 1, 6]
>>>
```

V druhej ukážke nepoužívame metódu **append()**, ale vytvoríme nový **zoznam** spojením existujúceho zoznamu a zoznamu s jedným prvkom (náhodné číslo, ktoré chceme do zoznamu pridať).

```
>>> from random import *
>>> cisla = []
>>> type(cisla)
```

```
<class 'list'>
>>> for i in range(5):
    cisla = cisla + [randrange(10)]

>>> cisla
[6, 7, 3, 9, 7]
>>>
```

Rovnaký efekt bude mať aj vytvorenie 5-prvkového zoznamu s hodnotami 0, ktoré následne zmeníme na náhodné číslo.

```
>>> from random import *
>>> cisla = [0] * 5
>>> type(cisla)
<class 'list'>
>>> for i in range(5):
    cisla[i] = randrange(10)

>>> cisla
[9, 5, 5, 5, 3]
>>>
```

Niekedy nechceme vytvoriť zoznam s konkrétnymi hodnotami. Vtedy môžeme použiť aj špeciálnu, ešte nedefinovanú hodnotu **None**. Neskôr ju zmeníme podľa potreby.

```
>>> cisla = [None] * 5
>>> cisla
[None, None, None, None, None]
>>>
```

Zoznam môžeme vytvoriť aj transformáciou z inej štruktúry (napríklad z textového reťazca, n-tice) pomocou funkcie **list()**.

```
>>> s = 'Python'
>>> l = list(s)
>>> l
['P', 'Y', 't', 'h', 'o', 'n']
>>>
```

```
>>> n = (1, 2, 3, 4, 5)
>>> l = list(n)
>>> l
[1, 2, 3, 4, 5]
>>>
```

Podobne zo zoznamu alebo z textového reťazca môžeme funkciou **tuple()** vytvoriť n-ticu.

```
>>> l = [1, 2, 3, 4, 5]
>>> n = tuple(l)
>>> n
(1, 2, 3, 4, 5)
>>>
```

Funkciu **list()** môžeme podhodiť aj generátor **range()**, ktorý poznáme z for cyklu.

```
>>> l = list(range(5))
>>> l
[0, 1, 2, 3, 4]
>>>
```

```
>>> l = list(range(5, -5, -1))
>>> l
```

```
[5, 4, 3, 2, 1, 0, -1, -2, -3, -4]
>>>
```

## Otzky:

14. Zistite, čo vypíše tento program. Ktorý zo zoznamov bude mať viac prvkov?

```
adam_navstivil = ['Bratislava', 'Praha', 'Budapešť']
boris_navstivil = adam_navstivil
boris_navstivil.append('Viedeň')
print(adam_navstivil)
print(boris_navstivil)
```

15. Zistite, čo vypíše tento program. Ktorá z n-tíc bude mať viac prvkov?

```
adam_navstivil = ('Bratislava', 'Praha', 'Budapešť')
boris_navstivil = adam_navstivil
boris_navstivil = boris_navstivil + ('Viedeň',)
print(boris_navstivil)
```

16. Zistite, čo vypíše tento program. Ktorý zo zoznamov bude mať viac prvkov?

```
adam_navstivil = ['Bratislava', 'Praha', 'Budapešť']
boris_navstivil = adam_navstivil[:]
boris_navstivil.append('Viedeň')
print(adam_navstivil)
print(boris_navstivil)
```

Ked' vytvoríme zoznam `adam_navstivil`, v premennej `adam_navstivil` je len referencia na zoznam (pole) s prvками 'Bratislava', 'Praha', 'Budapešť'. Zoznam je uložený v pamäti a referencia je len akýmsi odkazom na toto pamäťové miesto. Po priradení `boris_navstivil = adam_navstivil` sa nevytvorí kópia celého zoznamu, ale aj premenná `boris_navstivil` obsahuje referenciu (má odkaz) na rovnaký zoznam, čiže hodnoty uložené v pamäti. Takže je jedno, či nejako zmeníme zoznam `adam_navstivil` alebo `boris_navstivil`, vždy spravíme zmenu na rovnakom pamäťovom mieste, na ktoré referujú (odkazujú) obe premenné.

## Vizualizácia údajov programu

Na vizualizáciu programu môžeme použiť stránku <http://www.pythontutor.com>.

```
Python 3.6
1 adam_navstivil = ['Bratislava','Praha','Budapešť']
2 boris_navstivil = adam_navstivil
3 boris_navstivil.append('Viedeň')
4 print(adam_navstivil)
5 print(boris_navstivil)

Edit code | Live programming
line that has just executed
next line to execute
```

Print output (drag lower right corner to resize)

Frames			Objects		

Global frame

adam\_navstivil → list  
0 "Bratislava" | 1 "Praha" | 2 "Budapešť"

```
Python 3.6
1 adam_navstivil = ['Bratislava','Praha','Budapešť']
2 boris_navstivil = adam_navstivil
3 boris_navstivil.append('Viedeň')
4 print(adam_navstivil)
5 print(boris_navstivil)

Edit code | Live programming
line that has just executed
next line to execute
```

Print output (drag lower right corner to resize)

Frames			Objects		

Global frame

adam\_navstivil → list  
0 "Bratislava" | 1 "Praha" | 2 "Budapešť"

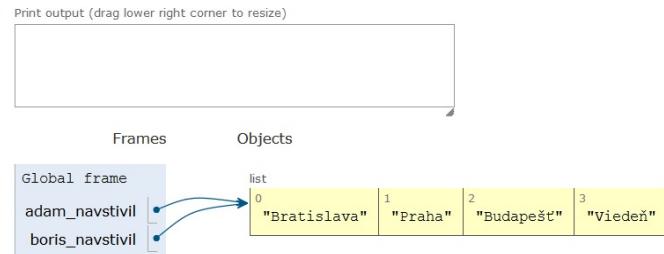
```

Python 3.6
1 adam_navstivil = ['Bratislava','Praha','Budapešť']
2 boris_navstivil = adam_navstivil
3 boris_navstivil.append('Viedeň')
4 print(adam_navstivil)
5 print(boris_navstivil)

Edit code | Live programming

```

line that has just executed  
next line to execute



**Ak potrebujeme vytvoriť kópiu zoznamu (listu), využijeme rez (slice). Použitím rezu vždy vytvoríme kópiu prvkov (daného rezu). My sme použili rez na celý zoznam.**

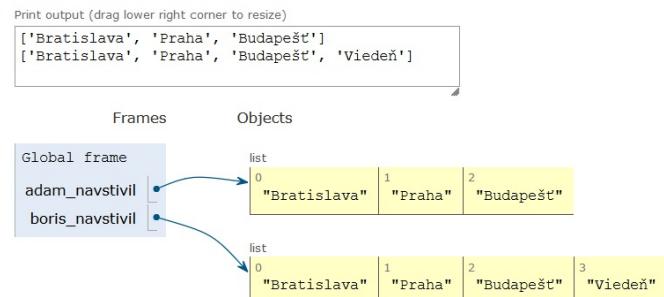
```

Python 3.6
1 adam_navstivil = ['Bratislava','Praha','Budapešť']
2 boris_navstivil = adam_navstivil[:]
3 boris_navstivil.append('Viedeň')
4 print(adam_navstivil)
5 print(boris_navstivil)

Edit code | Live programming

```

line that has just executed  
next line to execute



Doposiaľ sme sa s takouto situáciou nestretli, pretože reťazec ani n-tica nie sú meniteľné typy.

### Otázky:

17. Vytvorme 10-prvkové pole, naplnené hodnotami 0, 1 až 9 - `p = list(range(10))`. Pomocou shellu, alebo vizualizáciou na stránke <http://pythontutor.com> zistite, čo bude výsledkom takýchto úprav pôvodného zoznamu:
- `p[3:4] = [0]`
  - `p[3:4] = [0, 0]`
  - `p[3:4] = []`
  - `p[3:7] = []`
  - `p[2:5] = p[4:1:-1]`

## 5.3 Rozdelenie textového reťazca do zoznamu

Vytvorme program, ktorý sa nás spýta na navštívené hlavné mestá EÚ (zadáme ich oddelené medzerou) a vypíše nám počet navštívených miest a zoradí ich podľa abecedy.

```

odpoved = input('Napiš hlavné mestá EÚ, ktoré si navštívil (oddel ich medzerou):')
odpoved = odpoved.strip()
zoznam = []
mesto = ''
for i in range(len(odpoved)):
    if odpoved[i] == ' ':
        zoznam.append(mesto)
        mesto = ''
    else:
        mesto = mesto + odpoved[i]
zoznam.append(mesto) # (1)*
zoznam.sort()
print('Počet navštívených hlavných miest:', len(zoznam))
print('Abecedne zoradené hlavné mestá:')
for mesto in zoznam:
    print(mesto, end=' ')

```

V programe sme použili metódu `sort()`, ktorá zoradí prvky zoznamu.

```
Napiš hlavné mestá, ktoré si  
navštívil (oddel ich medzerou):Bratislava Praha Budapešť Viedeň Berlin  
Počet navštívených hlavných miest: 5  
Abecedne zoradené hlavné mestá:  
Berlin Bratislava Budapešť Praha Viedeň  
>>>
```

#### Otázky:

18. Zdôvodnite, prečo je potrebný v programe riadok označený komentárom `# (1) *`.
19. Zistite a popíšte situácie (rôzne vstupy), pre ktoré program nebude fungovať úplne správne.
20. Čo by sme mali zmeniť v programe, aby korektnie fungoval aj v prípade, že na konci riadku bude medzera? Alebo funguje korektnie aj v takomto prípade? Svoju odpoveď zdôvodnite.
21. Ako by sme mali program upraviť, aby fungoval správne aj v prípade, že jednotlivé mestá oddelíme viac ako jednou medzerou?

Namiesto cyklu, ktorý po nájdení medzery pridá do zoznamu mesto, môžeme využiť metódu textových reťazcov `retazec.split()`. Táto metóda rozdelí vstupný reťazec na jednotlivé prvky a vytvorí z nich `zoznam` jednotlivých znakov. Štandardne reťazec rozdelí podľa medzier. Parametrom môžeme určiť znak, podľa ktorého sa má reťazec rozdeľovať `retazec.split(oddelovač)`.

```
>>> odpoved = 'Bratislava Praha Budapešť'  
>>> zoznam = odpoved.split()  
>>> zoznam  
['Bratislava', 'Praha', 'Budapešť']  
>>>
```

```
>>> odpoved = 'Bratislava;Praha;Budapešť'  
>>> zoznam = odpoved.split(';')  
>>> zoznam  
['Bratislava', 'Praha', 'Budapešť']  
>>>
```

Výsledný program s funkciou `split()` je teraz kratší:

```
odpoved = input('Napiš hlavné mestá, ktoré  
si navštívil (oddel ich medzerou):')  
odpoved = odpoved.strip()  
zoznam = odpoved.split()  
zoznam.sort()  
print('Počet navštívených hlavných miest:', len(zoznam))  
print('Abecedne zoradené hlavné mestá:')  
for mesto in zoznam:  
    print(mesto, end=' ')
```

#### Úlohy:

- 11** V textovom súbore `mesta1.txt` sú hlavné mestá Európy. Na každom riadku je názov mesta, bodkočiarka a názov krajiny. Vytvorte program, ktorý vyzve používateľa, aby zadal hlavné mestá Európy, ktoré návštívil. Program porovná zoznam navštívených miest s mestami, ktoré sú v súbore, a používateľovi napíše, ktoré mestá ešte nenavštívil. Textový súbor môžeme v programe prečítať len raz. Ukážka súboru:

```
Amsterdam;Holandsko  
Andorra;Andorra  
Atény;Grécko
```

Belehrad; Srbsko  
Berlin; Nemecko  
Bern; Švajčiarsko  
Bratislava; Slovensko  
Brusel; Belgicko

12

V textovom súbore `mesta1.txt` sú hlavné mestá Európy. Na každom riadku je názov mesta, bodkočiarka a názov krajiny. Vytvorte program, ktorý vytvorí textový súbor `krajiny.txt`. V tomto súbore budú len názvy krajín (každá krajina bude na samostatnom riadku). Názvy krajín v súbore budú abecedne zoradené.

13

Vytvorte program, ktorý vyzve používateľa, aby zadal hlavné mestá Európy, ktoré návštívil on a ktoré jeho kamarát. Program porovná tieto zoznamy a vypíše mestá, ktoré už navštívili obaja.

14

Vytvorte program, v ktorom používateľ zadá slovo. Toto slovo sa vypisuje postupne po jednotlivých znakoch tak, že po každej sekunde sa objaví ďalšie písmeno. (V programe využite zoznam identifikátorov nakreslených útvarov.)

- a) Keď už bude vypísané celé slovo, začnú písmená postupne od prvého po posledné miznúť.
- b) Keď už bude vypísané celé slovo, začnú písmená postupne od posledného miznúť.
- c) Upravte program tak, aby písmená „prilietavalí“ sprava a po vypísaní celého slova „odlietavalí“ doľava.
- d) Upravte jednotlivé verzie programov z úloh a), b), c), aby sa táto animácia opakovala neustále dookola.

15

Vytvorte program, ktorý bude vedieť zobrazovať obrázok zakódovaný v textovom súbore `obrazok.txt`. Každý riadok súboru popisuje jeden objekt – obdĺžnik, ovál alebo čiaru. Program vykreslí obrázok, ktorý je zložený z týchto objektov.

Ukážka vstupných riadkov zo súboru:

```
r 10 20 150 200      # Obdĺžnik s danými súradnicami  
o 30 50 210 305      # Ovál s danými súradnicami  
l 20 100 170 180      # Čiara s danými súradnicami
```

Pridajte na koniec riadkov v textovom súbore informáciu o farbe jednotlivých objektov a upravte program, aby obrázok vykreslil v príslušných farbách.



16

Upravte anketový program z kapitoly 3.4 (úloha 15) tak, aby vedel pracovať nielen s tromi možnosťami na odpoveď, ale s ľubovoľným počtom možností. Jednotlivé možné odpovede budú taktiež uložené v textovom súbore.

Ktorú zeleninu máte najradšej?

Paprika	
Paradajky	
Uhorky	
Kapusta	
Brokolica	

## 5.4 Využívame zoznam v programoch

### Need for Speed

Naprogramujeme si zjednodušenú hru Need for Speed. V hre bude cesta a auto. Veľkosť grafického plátna bude 640 x 480 pixelov. Na začiatku je cesta rovná a je umiestnená presne v strede grafického plátna. Šírka cesty je 100 pixelov. V dolnej časti je auto umiestnené na ceste. Šípkami vľavo a vpravo hýbeme autom a cieľom hráča je udržať auto na ceste. Cesta sa skladá zo 48 riadkov, každý riadok má výšku 10 pixelov. V zvislej polohe je auto stále na rovnakom mieste. V hre sa posúva cesta, nie auto. Cesta sa hýbe (roluje) smerom dole. Vrch cesty (prvý riadok) sa náhodne generuje, ale tak, aby bol o plus mínus 10 pixelov posunutý voči druhému riadku. Čiže na vrchu sa generujú nové riadky a tie sa postupne rolujú dole po obrazovke až z nej odídu. V programe je zabezpečené, aby cesta neodišla mimo ľavý alebo pravý okraj obrazovky (aby sme ju mohli hrať).



#### Otázky:

22. Ktoré údaje si budeme musieť pamätať v programe?
23. V akých štruktúrach bude vhodnejšie si ich pamätať?
24. Ako je vhodné realizovať posun cesty o jeden riadok a generovať nový riadok?
25. Budeme využívať `canvas.move`, alebo budeme cestu vymazávať a znova kresliť?
26. Čo všetko je potrebné urobiť v časovači?
27. V ktorej časti programu budeme hýbať autom, kde ho musíme prekreslovať?
28. V ktorej časti programu je dobré testovať, či je auto na ceste?
29. Ako zabezpečíme, aby cesta neodišla z bokov plátna?

Najprv vytvoríme `canvas`, v premennej `riadkov` si budeme pamätať počet riadkov cesty, aby sme prípadne mohli náš program neskôr ľahko upraviť. Canvas bude zelený a my budeme kresliť na zelené pozadie len bielu cestu. Je to jednoduchšie ako na biele pozadie kresliť zvlášť ľavú a pravú časť plochy vedľa cesty. Takto nám namiesto dvoch obdĺžnikov stačí nakresliť len jeden obdĺžnik v strede.

Každý riadok cesty si budeme pamätať v zozname `cesta`, označme si ich nultý až 47. riadok. Napríklad v prvku s indexom 0 si pamäťame súradnicu nultého riadku, v prvku s indexom 1 súradnicu prvého riadku atď., až v prvku s indexom 47 si pamäťame súradnicu posledného riadku (úplne spodného). Súradnica riadku je v našom prípade x-ová súradnica ľavého okraja cesty. Na začiatku je cesta rovná úplne v strede (stred má hodnotu 320, šírka cesty je 100), takže ľavý okraj má súradnicu  $320 - 50 = 270$ .

```
import tkinter
from random import *
canvas = tkinter.Canvas(width = 640, height = 480, bg = 'green')
canvas.pack()
```

```
riadkov = 48
cesta = [270]*riadkov
```

Ďalším krokom bude kreslenie cesty. Vytvoríme funkciu `kresli_cestu()`. Funkcia najprv zmaže nakreslenú cestu (podľa tagu pre cestu). Následne for cyklom prejdeme všetky riadky cesty a podľa x-ovej súradnici v zozname `cesta` pre príslušný riadok nakreslíme biely obdĺžnik (cestu). Y-ovú súradnicu si nikde nemusíme pamätať, pretože každý riadok má podľa zadania výšku 10 pixelov, a teda si vieme y-ovú súradnicu vypočítať podľa `i` (indexu prvku zoznamu, pre ktorý cestu kreslíme).

```
def kresli_cestu():
    canvas.delete('cesta')
    for i in range(riadkov):
        canvas.create_rectangle(cesta[i], i*10, cesta[i]+100, i*10+10,
                               fill = 'white', outline = '', tags = 'cesta')
```

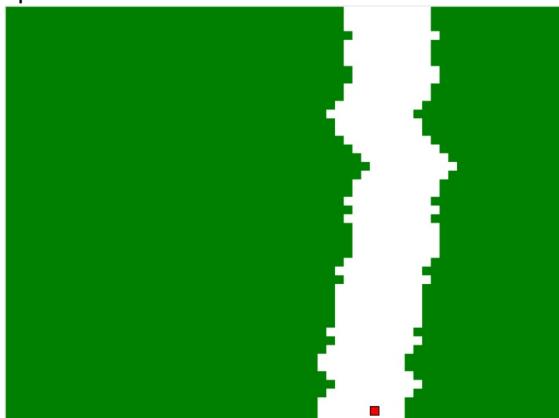
V časovači zavoláme funkciu `posun_cestu()`, ktorá posunie cestu o jeden riadok dole a vygeneruje nový riadok. Následne cestu vykreslíme a budeme plánovať ďalšie spustenie animácie. Zrejme tam neskôr doplníme ďalšie veci.

```
def animacia():
    posun_cestu()
    kresli_cestu()
    canvas.after(100, animacia)
```

Teraz musíme vytvoriť funkciu `posun_cestu()`. V nej si najprv vypočítame súradnicu nového riadku, ktorý vložíme na vrch plátna. Vypočítame si ho zo súradnice terajšieho nultého riadku a k nemu pričítame náhodne +10 alebo 0 alebo -10 (treba si uvedomiť, že k tomu nepotrebujeme `if`). **If**-om ošetríme, aby nová súradnica nebola mešia ako 0 a väčšia ako 540 (640 - 100), aby cesta ostala na zobrazenej časti plátna. Túto novú x-ovú súradnicu nultého riadku vložíme na začiatok zoznamu (pred terajší nultý prvok) metódou `insert`. Zaroveň musíme z konca zoznamu odobrať posledný riadok metódou `pop()`, aby mala cesta stále len 48 riadkov.

```
def posun_cestu():
    novy = cesta[0] + randint(-1, 1)*10
    if novy < 0:
        novy = 0
    if novy > 540:
        novy = 540
    cesta.insert(0, novy)
    cesta.pop()
```

Ešte musíme spustiť funkciu `animacia()`. Program je teraz už funkčný - teda posúvanie cesty. Ešte musíme vyriešiť samotné auto. O aute si potrebujeme pamätať x-ovú a y-ovú súradnicu. Zadefinujeme na globálnej úrovni premennú `autox` a `autoy` a priradíme im hodnoty tak, aby bolo auto v strede cesty na predposlednom riadku. Zároveň zviažeme udalosť stlačenia šípky vľavo a vpravo s funkciou, ktorá bude riešiť posun auta vľavo a vpravo.



Auto by sme mali nakresliť po prvom spustení animácie, ale to nebude stačiť. Kedže v animácii zakaždým zmažeme a znova nakreslíme cestu, cesta nám prekryje obrázok auta. Potrebujeme teda v našom riešení zakaždým kresliť auto aj v animácii (časovači). A takto (zatial) zapíšeme náš celý program:

```
import tkinter
from random import *
canvas = tkinter.Canvas(width = 640, height = 480, bg = 'green')
canvas.pack()

riadkov = 48
cesta = [270]*riadkov

def kresli_cestu():
    canvas.delete('cesta')
    for i in range(riadkov):
        canvas.create_rectangle(cesta[i], i*10, cesta[i]+100, i*10+10,
                               fill = 'white', outline = '', tags = 'cesta')

def posun_cestu():
    novy = cesta[0] + randint(-1, 1)*10
    if novy < 0:
        novy = 0
    if novy > 540:
        novy = 540
    cesta.insert(0, novy)
    cesta.pop()

def animacia():
    posun_cestu()
    kresli_cestu()
    kresli_auto()
    canvas.after(100, animacia)

def kresli_auto():
    canvas.delete('auto')
    canvas.create_rectangle(autox, autoy, autox+10, autoy+10, fill = 'red',
                           tags = 'auto')

def vlavo(event):
    global autox
    autox -= 10

def vpravo(event):
    global autox
    autox += 10

autox = 320
autoy = 460
canvas.bind_all('<Left>', vlavo)
canvas.bind_all('<Right>', vpravo)
animacia()
kresli_auto()
```

### Úlohy:

**17** Doplňte do programu test, ktorý zistí, či je alebo nie je auto mimo cesty. Otestujte aj správnosť umiestnenia tohto testu v programe.

**18** Upravte program tak, aby hráčovi povolil maximálne tri nehody (opustenia cesty) a po tomto limite hra zastane a vypíše informáciu, kolko riadkov cesty (aký dlhý úsek) sme prešli.

**19**

Upravte program tak, aby sa postupne zvyšovala náročnosť hry pre hráča. Napríklad zužovaním cesty, zmenou rýchlosťi, alebo ich kombináciou.

**20**

Vylepšite spôsob generovania cesty tak, aby išla v dlhších úsekoch vľavo alebo vpravo.

**21**

Upravte program tak, aby priebeh celej hry (aj pohyb cesty, aj auta) zaznamenal do textového súboru. Tako uložený záznam si môžeme prehrať (bud' rovnakým alebo iným programom). Diskutujte a zvažujte, ako a ktoré informácie je vhodné do súboru zapísat, aby sme zaznamenali všetko potrebné, ale aby sme zároveň zapísali čo najmenej údajov (veľkosť súboru).

**22**

Doplňte do programu tlačidlo, ktorým zapneme režim autopilota (počítač bude hrať hru sám).

**23**

Vylepšite grafiku hry, napríklad cesta bude jemnejšie vykreslená a jeden riadok nebude mať 10 pixelov.

## Vyhľadávanie prvku s požadovanými vlastnosťami

V textovom súbore **cyklotrasa1.txt** sú uložené údaje o cyklotrase z Hlohovca do Dubnice nad Váhom. V každom riadku sú tri informácie o jednom mieste trasy. Prvou informáciou je názov miesta trasy. Druhú informáciu tvorí prírastok nadmorskej výšky daného miesta voči predchádzajúcemu miestu. Špeciálne v prvom riadku je nadmorská výška miesta štartu. Tretí údaj je vzdialenosť daného miesta od predchádzajúceho miesta. Informácie v riadku sú oddelené bodkočiarkou. Ukážka súboru:

```
Hlohovec;156;0
Koplotovce;9;2
Sokolovce;-5;10
Ratnovce;30;3
Síňava;-32;2
Pod červenou Vežou;4;0.5
Piešťany - Kúpeľný most;0;0.6
Moravany nad Váhom;3;2.2
Zástavka pod Hubinou;20;2.5
Ducové;-5;1
Modrovka;-10;3.5
Lúka;30;0.5
Hrádok;0;6
Hôrka nad Váhom;-7;3
Nová Ves nad Váhom;-13;1.5
Kočovce;3;1.5
Beckov;10;5
Trenčianske Stankovce;37;11
Trenčín;-19;10
Kubrá;19;4
Trenčianska Teplá;-10;8
Dubnica nad Váhom;22;3.5
```



Vytvoríme program, ktorý prečíta textový súbor, vypočíta nadmorské výšky jednotlivých miest, vypočíta dĺžku

trasy a vykreslí terénny profil trasy. Program zistí najvyššie miesto trasy, vypíše jeho geografický názov a nadmorskú výšku.

```
import tkinter
from random import *
canvas = tkinter.Canvas(width = 1000, height = 500, bg = 'white')
canvas.pack()

miesta = []
vysky = []
vzdialenosťi = []
f = open('cyklotrasa1.txt', 'r')
riadok = f.readline().strip()
info = riadok.split(';')
miesta.append(info[0])
vysky.append(int(info[1]))
vzdialenosťi.append(float(info[2]))
for riadok in f:
    riadok = riadok.strip()
    info = riadok.split(';')
    miesta.append(info[0])
    vyska = vysky[-1] + int(info[1])
    vysky.append(vyska)
    vzdialenosťi.append(float(info[2]))
f.close()

x = 50
maxy = 500
trasa = ()
for i in range(len(miesta)):
    x += vzdialenosťi[i] * 10
    y = maxy - vysky[i]*2
    trasa = trasa + (x, y)
    canvas.create_oval(x-4, y-4, x+4, y+4, fill = '#4cb050', outline = '')
    canvas.create_text(x, y-10, text = vysky[i])
    canvas.create_text(x, maxy-200, text = miesta[i], angle = 90)
canvas.create_line(trasa)
dlzka_trasy = sum(vzdialenosťi)
najvyssia_vyska = max(vysky)
index_najvyssej_vysky = vysky.index(najvyssia_vyska)
miesto_najvyssej_vysky = miesta[index_najvyssej_vysky]
canvas.create_text(500, 440, text = 'V mieste: '+miesto_najvyssej_vysky+
                  ' bola najvyššia nadmorská výška: '+str(najvyssia_vyska)+'m',
                  font = 'Arial 20', fill = '#4cb050')
canvas.create_text(500, 470, text = 'Dĺžka celej trasy je: '+str(dlzka_trasy)+'
                  font = 'Arial 20', fill = '#4cb050')
```

V programe sme využili funkcie `sum()` a `max()`. Funkcia `sum()` vypočíta súčet všetkých prvkov postupnosti (n-tice, zoznamu, ...). Funkcia `max()` vráti hodnotu maximálneho prvku z postupnosti (n-tice, zoznamu, ...). Na zistenie najmenšieho prvku môžeme využiť funkciu `min()`.

### Úlohy:

**24** Upravte program tak, aby nám vypísal aj najnižšie položené miesto na trase.

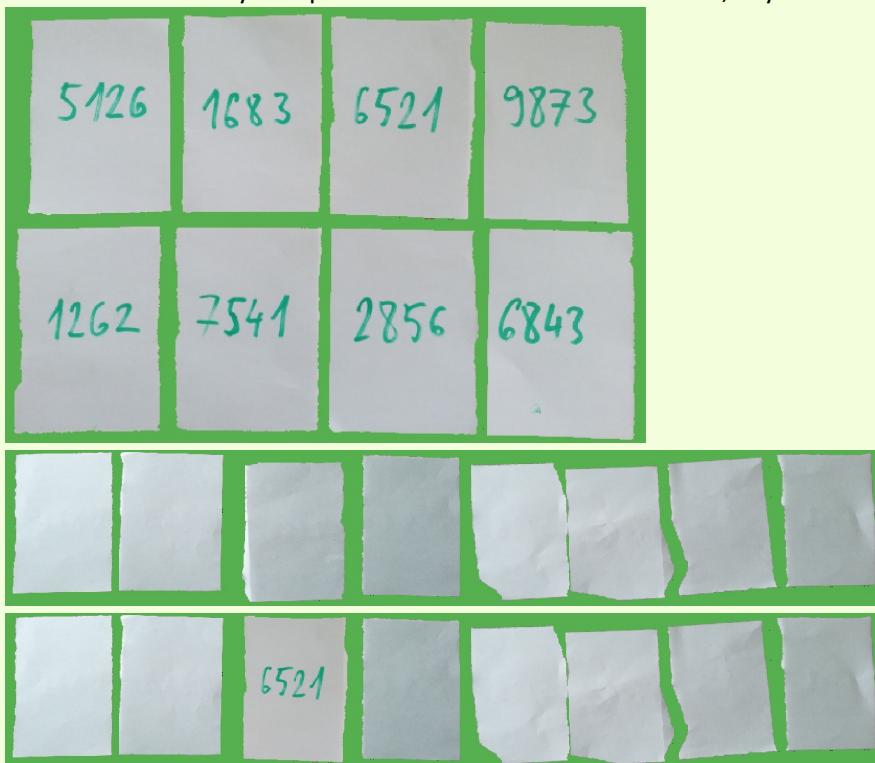
**25** Upravte program tak, aby sme pochymom myši po názvoch geografického miesta mohli určiť úsek od začiatku trasy po dané miesto, a program nám vypočíta a vypíše na danom úseku najvyššie prevýšenie a dĺžku trasy od štartu. Prevýšenie je rozdiel dvoch nadmorských výšok medzi dvomi bodmi trasy (môže byť aj kladné, aj záporné). Fungoval by váš program aj na klesajúcej trase?

**26** Upravte program tak, aby vytvoril nový textový súbor `cyklotrasa2.txt`, v ktorom budú geografické názvy miest, ich absolútна nadmorská výška a celková vzdialenosť od štartu.

**27** Naprogramujte si svoju funkciu `sucet`, ktorá spočíta súčet prvkov zoznamu (nepoužite funkciu `sum`).

**28** Naprogramujte si svoju funkciu `priemer`, ktorá vypočíta priemer prvkov zoznamu.

**29** Zoberte si papier A4, zložte ho na polovicu a rozdeľte ho. Získané polovice opäť zložte na polovicu a rozdeľte ich. Teraz každú (všetky štyri) časť ešte raz zložte na polovicu a rozdeľte ju. Získate 8 kartičiek. Na kartičky si napište náhodné štvorciferné čísla tak, aby ich nebolo vidieť z opačnej strany.



Všetky kartičky otočte tak, aby ste nevideli napísané čísla. Teraz máte pred sebou osemprvkový zoznam a môžete robiť s ním len to, čo vie robiť počítač (nemôžete sa pozrieť na všetky čísla naraz). Čiže pozrieť sa na jedno číslo, prípadne zapamätať si jedno číslo. Odkryté môžete mať maximálne dve kartičky, ale môžete stále odkrývať iné dve kartičky, môžete ich navzájom porovnávať a zapamätať si maximálne jedno číslo. Vymyslite algoritmus, ktorým zo zoznamu (z kartičiek) zistíte najväčšie číslo. Skúste tento algoritmus zapisať programom a overiť jeho funkčnosť a správnosť. (V programe nesmieme použiť metódu `sort()`, ale ani funkciu `min()` a `max()`, ideme si ju sami naprogramovať).

**30** Použite kartičky z predchádzajúcej úlohy a vymyslite teraz algoritmus, ktorým nájdeme na kartičkách najmenšie číslo. Skúste tento algoritmus zapisať programom a overiť jeho funkčnosť a správnosť. (V programe nesmieme použiť metódu `sort()`, ale ani funkciu `max()` alebo `min()`, ideme si ju sami naprogramovať).

Ked' váš program funguje, zmeňte niektoré čísla v zozname na záporné a zistite, či bude fungovať aj pre takýto vstup. Svoj program upravte tak, aby sme ho nemuseli upravovať podľa toho, aký rozsah čísel je zadaný na vstupe.

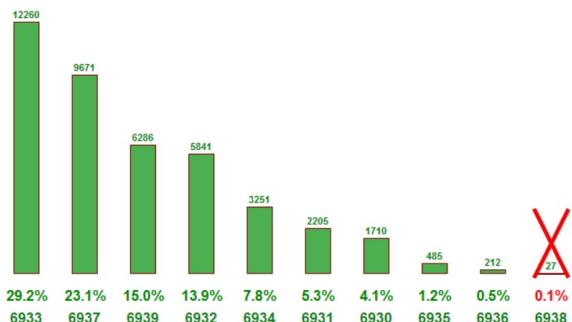
**31** V skupine (dvoj až trojčennej) si pozrite vytvorené programy a hľadajte v nich kritické miesta, resp. taký vstup, pre ktorý nebudú fungovať správne. Programy podľa potreby upravte, vylepšite.

## SMS hlasovanie, metóda sort

Súťažná televízna reality show umožňuje divákom hlasovať zaslaním SMS. V súťaži je desať účastníkov. Každý

súťažiaci má pridelené SMS číslo (6930, 6931, 6932 až 6939), na ktoré môžu diváci zaslať SMS hlas. Všetky príchodzie SMS sa zaznamenávajú v jednom textovom súbore **sms.txt**. Súbor na každom riadku obsahuje SMS číslo, na ktoré divák zaslal svoj hlas. My vytvoríme program, ktorý spočíta zaslané hlasy jednotlivým súťažiacim a vykreslí graf. V grafe budú znázornené scítané hlasy súťažiacich, zoradené od najvyššieho počtu hlasov po najnižší. Budeme vidieť aj počet hlasov, aj percento zo všetkých hlasov a SMS číslo. Súťažiaci s najnižším počtom hlasov vypadáva a v grafe bude prečiarknutý.

Hlasovalo: 41948 ľudí.



### Otázky:

30. Ktoré údaje si budeme musieť v programe pamätať?
31. V akých štruktúrach bude vhodnejšie si ich pamätať?
32. Potrebujeme si počítať celkový počet hlasov?
33. Ako môžeme počty získaných hlasov zoradiť?
34. Čo spraví nasledujúci program?
  - a) cisla = [20, 4, 10, 5]  
cisla.sort()
  - b) cisla = [(0,20), (1,4), (2,10), (3,5)]  
cisla.sort()
  - b) cisla = [(20,0), (4,1), (10,2), (5,3)]  
cisla.sort()

Ukážka súboru **sms.txt**:

```
6930
6935
6935
6937
6930
6937
6932
6937
6930
```

V programe máme počítať počet hlasov pre jednotlivé čísla. Súťažiacich je desať, tak si počty ich hlasov budeme pamätať v desaťprvkovom zozname. Zoznam bude mať indexy 0 až 9, my však máme SMS čísla 6930 až 6939. Odčítaním čísla 6930 od SMS čísla si vyrábíme index 0 až 9 a k tomuto prvku pričítame hlas (keď je také číslo prečítané z riadku textového súboru).

```
import tkinter
WIDTH, HEIGHT = 720, 480
canvas = tkinter.Canvas(width = WIDTH, height = HEIGHT, bg = 'white')
```

```

canvas.pack()

subor = 'sms.txt'
hlasy = [0] * 10

def citaj_subor(subor):
    f = open(subor, 'r')
    for cislo in f:
        cislo = int(cislo.strip())-6930
        hlasy[cislo] += 1
    f.close()

citaj_subor(subor)
pocet_hlasujucich = sum(hlasy)

```

V shelli vypíšeme obsah zoznamu **hlasy**:

```

>>> hlasy
[1710, 2205, 5841, 12260, 3251, 485, 212, 9671, 27, 6286]

```

Vidíme, že súťažiaci s SMS číslom 6930 získal 1710 hlasov, súťažiaci s SMS číslom 6931 získal 2205 hlasov. Teraz potrebujeme zoradiť pridelené hlasov. Musíme si uvedomiť, že nestačí len zoradiť pole **hlasy.sort()**, lebo zoradením by sme stratili informáciu o tom, ktorému súťažiacemu daný počet hlasov prislúcha. Tento problém môžeme riešiť rôznymi spôsobmi. My si vytvoríme kópiu zoznamu s názvom **hlasy2**, kde prvkom budú dvojprvkové n-tice. N-ticu bude tvoriť počet hlasov a SMS číslo súťažiaceho. Až tento zoznam **hlasy2** zoradíme.

```

hlasy2 = []
for i in range(len(hlasy)):
    hlasy2.append((hlasy[i], i+6930))
hlasy2.sort()

```

Pre kontrolu si v shelli vypíšeme obsah zoznamu **hlasy2**:

```

>>> hlasy2
[(27, 6938), (212, 6936), (485, 6935), (1710, 6930), (2205, 6931),
(3251, 6934), (5841, 6932), (6286, 6939), (9671, 6937), (12260, 6933)]

```

Teraz už môžeme podľa zoznamu **hlasy2** vykresliť graf.

```

maxH = HEIGHT*2

def graf():
    x1 = WIDTH // (len(hlasy2)+1)
    y1 = HEIGHT-75
    canvas.create_text(WIDTH/2, 15, text = 'Hlasovalo: ' +
                       str(pocet_hlasujucich) + ' ľudí.',
                       font = 'Arial 12 bold', fill='#15791c')

    index = 10
    for info in hlasy2[::-1]:
        index -= 1
        sms = info[1]
        pocet = info[0]
        percento = (pocet/pocet_hlasujucich)*100
        percento = '{:.1f}%'.format(percento)
        y2 = int((pocet/pocet_hlasujucich)*maxH)
        canvas.create_rectangle(x1, y1, x1+28, y1-y2, fill = '#4cb050')
        canvas.create_text(x1+14, y1+50, text = sms, font = 'Arial 12 bold',
                           fill = '#15791c')
        canvas.create_text(x1+14, y1-y2-8, text = pocet, font = 'Arial 8 bold',
                           fill = '#15791c')
    if index == 0:

```

```

        canvas.create_line(x1-5, y1-y2-72, x1+33, y1+5, width = 5,
                            fill = 'red')
        canvas.create_line(x1-5, y1+5, x1+33, y1-y2-72, width = 5,
                            fill = 'red')
        canvas.create_text(x1+15, y1+25, text = percento,
                           font = 'Arial 12 bold', fill = 'red')
    else:
        canvas.create_text(x1+15, y1+25, text = percento,
                           font = 'Arial 12 bold', fill = 'green')
    x1 += WIDTH // (len(hlasy2)+1)

graf()

```

#### Otázky:

35. Prečo for cyklus prechádza poľom týmto spôsobom `hlasy2[::-1]` a nastačí tam zapísť `for info in hlasy2?`
36. Čo by sme museli zmeniť v programe (a ako), ak by bol for cyklus zapísaný takto: `for i in hlasy2?`

#### Úlohy:

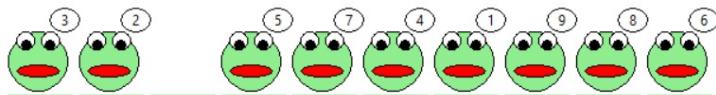
**32**

Textový súbor `sms.txt` je pri počte hlasovaní 41 948 ľudí pomerne veľký. Vymyslite iný spôsob jeho zápisu tak, aby zaberal súbor menej miesta, ale tak, aby sme neprišli o informáciu postupnosti hlasovania. Tento spôsob zrealizujte a konvertujte súbor `sms.txt` na súbor `sms2.txt`. Porovnajte veľkosť oboch súborov a diskutujte o svojich nápadoch v rámci triedy.

## Žaby, vymieňanie prvkov v zozname

Vytvoríme hru skákanie žiab. Na začiatku hry sa vykreslí vedľa seba 10 lekien. Na deviatich leknánoch sú náhodne usadené žaby s číslami 1 - 9. Jedno z lekien je voľné. Kliknutím na žabu ju môžeme presunúť na voľné lekno, ale len v prípade, ak žaba susedí s voľným leknom, alebo ak pri presune preskočí maximálne 1 žabu. Hráč vyhráva, keď sú žaby usporiadané od 1 po 9 a posledné lekno je voľné.

Náhodné usporiadanie na začiatku hry:



Výherná pozícia:



**Vyhral si!**

#### Otázky:

37. Ktoré údaje si budeme musieť v programe pamätať?
38. V akých štruktúrach bude vhodnejšie si ich pamätať?
39. Potrebujeme si pamätať o každej žabe aj číslo lekna, na ktorom sedí?
40. Ako vytvoríme náhodne poradie žiab na leknánoch?
41. Ako zistíme, na ktorú žabu sme klikli myšou (teda číslo lekna)?
42. Ako zistíme, či žaba splňa kritériá na skok?

### 43. Ako zistíme, či sú žaby usporiadane vo výhernej pozícii?

Poradie žiab si môžeme pamätať v desaťprvkovom zozname. Index prvku zoznamu bude v skutočnosti číslom lekna a hodnota prvku bude číslo žaby, ktorá na lekne sedí. Ak je lekno prázdne, môžeme si tam pamätať napríklad hodnotu 0. Vhodnejšie ako náhodné Žrebovanie poradia žiab bude, keď si ich najprv vygenerujeme v usporiadanom poradí a potom ich náhodne povymieňame. Tým sa vyhneme opakovanejmu vyžrebovaniu toho istého čísla. Vymenie viacerých hodnôt môžeme v Pythone urobiť aj takto: `a, b = b, a` (v skutočnosti sme pracovali s dvojprvkovými n-ticami).

```
import tkinter
from random import *
canvas = tkinter.Canvas(width = 610, height = 270, bg = 'white')
canvas.pack()

sirka_zaby = 60

zaby = list(range(10))
print(zaby)

for i in range(len(zaby)):
    ktory = randrange(10)
    zaby[i], zaby[ktory] = zaby[ktory], zaby[i]

print(zaby)
```

Teraz môžeme žaby aj s leknami vykresliť. Ak je na lekne hodnota 0, žabu kresliť nebudeme.

```
def kresli_zabu(x, y, cislo):
    canvas.create_line(x, y+55, x+55, y+55, width = 3, fill = 'green')
    if cislo != 0:
        canvas.create_oval(x, y, x+50, y+50, fill = 'lightgreen')
        canvas.create_oval(x+7, y+28, x+43, y+38, fill = 'red')
        canvas.create_oval(x+7, y, x+22, y+15, fill = 'white')
        canvas.create_oval(x+28, y, x+43, y+15, fill = 'white')
        canvas.create_oval(x+11, y+7, x+19, y+15, fill = 'black')
        canvas.create_oval(x+31, y+7, x+39, y+15, fill = 'black')
        canvas.create_oval(x+35, y-20, x+60, y, fill = 'white')
        canvas.create_text(x+48, y-10, text = cislo)

def kresli():
    canvas.delete('all')
    for i in range(len(zaby)):
        kresli_zabu(i*sirka_zaby+10, 100, zaby[i])

kresli()
```

Ešte budeme reagovať na ľavý klik myši. Z x-ovej súradnice si celočiselným delením veľkosťou jedného polička vypočítame index žaby `index = (x-10)//sirka_zaby`. Najprv odčítame hodnotu 10, lebo vykreslovanie žiab je posunuté o 10 px vpravo. Napríklad, ak sme na súradnici `x = 68`, tak sme klikli ešte na nulté lekno `index = 68-10 // 60. index = 58 // 60, index = 0`. Ak by sme hodnotu 10 neodpočítali, vyšiel by nám index lekna = 1 a to už je druhé lekno, nie prvé (nulté).

Pomocou metódy `zaby.index(0)` zistíme, ktoré lekno je voľné. Odpočítaním indexu lekna, na ktoré sme klikli, a voľného lekna, získame ich vzdialenosť (v počte lekien). Iba ak je táto vzdialenosť menšia ako tri, môže nastať výmena prvkov. Skontrolujeme, či nejdeme vymieňať neexistujúce prvky (s indexom mimo rozsahu zoznamu). Pri výpočte vzdialenosť použijeme absolútну hodnotu `abs()`, lebo odpočítaním týchto dvoch indexov môžeme získať aj záporné číslo.

```
def klik(sur):
    index = (sur.x - 10)//sirka_zaby
    prazdne = zaby.index(0)
```

```

vzdialenosť = abs(index-prázdne)
if vzdialenosť < 3 and 0 <= index <= 9:
    zaby[index], zaby[prázdne] = zaby[prázdne], zaby[index]
    kresli()
    if vyherná_pozícia():
        gratulacia()

canvas.bind('<Button-1>', klik)

```

Ešte musíme vytvoriť funkciu `vyherná_pozícia()`, ktorá vráti hodnotu `True` v prípade, že žaby sú správne usporiadané. Funkcia gratulácia nám len vypíše, že sme vyhrali.

```

def vyherná_pozícia():
    for i in range(len(zaby)-1):
        if i != zaby[i]-1:
            return False
    return True

def gratulacia():
    canvas.create_text(300, 200, text = 'Vyhral si!', font = 'Arial 50',
                      fill = 'green')

```

### Otázky:

44. Ak by sme nepoznali funkciu `abs()`, ako inak môžeme program upraviť?
45. Čo a v ktorých situáciách by robil program inak, ak by sme vynechali túto podmienku: `0 <= index <= 9`?
46. Koľkokrát sa zopakuje telo `for` cyklu vo funkcií `vyherná_pozícia()`?
47. Čo sa stane, ak po výhre klikneme na niektorú žabu, ktorá splňa kritérium na presunutie?
48. Nasledujúce riešenie funkcie `vyherná_pozícia()` je chybné. Len v niektorých situáciách správne vyhodnotí výhernú pozíciu. Nájdite, pre ktoré vstupy bude fungovať funkcia správne a pre ktoré nesprávne. Ako by sme mali funkciu upraviť, aby fungovala správne (za predpokladu, že nezmeníme podmienku v `if-e`)?

```

def vyherná_pozícia():
    ok = False
    for i in range(len(zaby)-1):
        if i == zaby[i]-1:
            ok = True
    return ok

```

### Úlohy:

- 33** Upravte program tak, aby nám počítal počet presunov.
- 34** Upravte program tak, aby sme si mohli zadať štartovnú pozíciu a na nej sme mohli trénovať minimálny počet pokusov na usporiadanie.
- 35** Upravte program tak, že počítač sám hrá hru žaby (náhodnými ťahmi).
- 36** Upravte program tak, že ťah bude animovaný - žaba pomaly preskočí na nové lekno. Ak nemôže skákať, iba poskočí na mieste.
- 37** Upravte program tak, že do textového súboru zapíše počiatočnú pozíciu žab a počas hry zapíše do súboru historiu ťahov hráča. Z takto vytvoreného súboru bude vedieť program znova prehrať zaznamenanú hru.

**38**

(\*) Upravte program tak, že počítač sa sám naučí optimálne hrať hru.

**39**

Vytvorte simulátor trávnika. Program vykreslí naspodu grafickej plochy 100 zelených čiar – stiebel trávy – rozumnej náhodnej dĺžky.



- Po kliknutí na tlačidlo **Poliať trávnik** tráva trochu narastie (o rozumné náhodné hodnoty).
- Po kliknutí na tlačidlo **Pokosiť sa** čiary skráťia na približne rovnakú dĺžku.
- Upravte program tak, aby tráva postupne rástla.

**40**

Každý záhradkár pozná trápenie sa s burinou. Tá, samozrejme, rastie najrýchlejšie. Upravte program z predchzajúcej úlohy (simulátor trávnika). Pridajte do programu možnosť zadať číslo – koľko percent trávnika je zaburineného. Burinu vykreslite iným odtieňom zelenej.

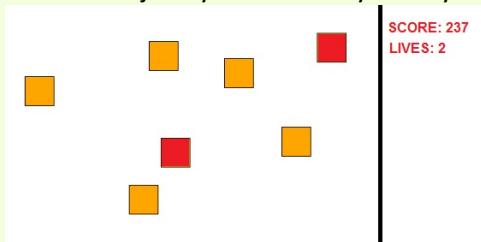


- Po kliknutí na tlačidlo **Poliať trávnik** stieblá buriny narastú oveľa výraznejšie ako zvyšok trávnika a zároveň namiesto jedného náhodného stiebla trávy začne rásť nová burina.
- Pridajte tlačidlo **Vyplieť**, ktoré vytrhá 90% buriny (tak, aby vždy zostala aspoň jedna).
- Pridajte tlačidlo **Herbicíd**, ktoré simuluje postupné vyschnutie buriny po postreku herbicídmi (burina bude postupne meniť farbu až do šedej farby - vyschne).

**41**

Naprogramujte hru **Klikačka**.

- V grafickej ploche sa postupne objavujú štvorčeky a hráč ich odstreľuje klikaním na ne, pričom sa mu pripočítavajú body. Hráč sa tak snaží zabrániť, aby sa plocha preplnila nad úroveň kritického množstva štvorčkov.
- Štvorčeky budú po určitom čase miznúť, za čo sa odrátajú body. Hráč musí stihnuť odstreliť štvorček.
- Ak hráč klikne, ale netrafí žiadny štvorček, odrátajú sa mu body.
- Vylepšite túto hru podľa vlastných predstáv – dajte jej meno, vymyslite, kedy má hra skončiť, pridajte štvorčeky inej farby, na ktoré sa musí kliknúť viackrát, či „dobré“ štvorčeky, na ktoré sa nemá klikať. Pridajte zrýchľovanie hry a body na záver uložte do textového súboru – do tabuľky High score.



# 6 Obrázky

## 6.1 Načítavanie a kreslenie obrázkov gif a png

Doposiaľ sme v programoch sami kreslili obrázky pomocou obdĺžnikov, čiar, elíps a pod.. Pomocou knižnice `tkinter` môžeme v programe využiť aj obrázky zo súboru gif alebo png.

Najprv si vytvoríme špeciálnu obrázkovú premennú, do ktorej načítame obrázok zo súboru:

```
premena = tkinter.PhotoImage(file = 'meno súboru')
```

Obrázok sa ešte nikde neukáže, len sa vytvorí premenná, v ktorej si Python obrázok pamätá. Pokiaľ uvedieme len meno súboru s príponou, bude Python hľadať tento súbor v tom istom priečinku, kde máme uložený program. Ak ho tam nenájde, ohlási nám chybu. V mene súboru môžeme uvádzať aj celú cestu k súboru alebo relatívnu cestu. Vhodnejšie je používanie relatívnej cesty, pretože aj po premiestnení nášho programu do iného počítača spolu s rovnakou štruktúrou priečinkov s obrázkami bude naša relatívna cesta správna a nemusíme ju opravovať. Napríklad, ak máme svoj program umiestnený na disku D:\ v priečinku `python` a v tomto priečinku je priečinok `obrazky`, kde je súbor s názvom `zaba1.gif`, zapíšeme:

```
obrazok1 = tkinter.PhotoImage(file = 'obrazky/zaba1.png')
```

Obrázok z premennej môžeme teraz umiestniť do canvasu príkazom:

```
canvas.create_image(50, 50, image = obrazok1)
```

Tento program nakreslí obrázok z premennej `obrazok` na mieste kliknutia myši:

```
import tkinter
canvas = tkinter.Canvas(width = 600, height = 400)
canvas.pack()

obrazok1 = tkinter.PhotoImage(file = 'obrazky/zaba1.png')

def klik(sur):
    canvas.create_image(sur.x, sur.y, image = obrazok1)

canvas.bind('<Button-1>', klik)
```



Pozor, pamäťajme na to, že pri použití príkazu `canvas.create_image` sa obrázok neodtlačí do canvasu. Obrázok je len zobrazený podľa referencie na načítaný obrázok (čiže ukazuje to, čo je raz načítané niekde v pamäti). Ak by sme zmenili premennú `obrazok1` na inú hodnotu (alebo aj iný obrázok), zobrazený obrázok zmizne. Python nevie, že my tú pôvodnú referenciu v `canvas.create_image` používame, a tak z pamäte odstráni načítaný obrázok, lebo premennou `obrazok1` teraz referujeme na niečo nové.

```
>>> import tkinter
>>> canvas = tkinter.Canvas(width = 600, height = 400)
>>> canvas.pack()
```

```

>>> obrazok1 = tkinter.PhotoImage(file = 'obrazky/zaba1.png')
>>> canvas.create_image(50, 50, image = obrazok1)
1
>>> obrazok1 = 'nová informácia'

```

Z podobného dôvodu nebude fungovať ani tento program:

```

import tkinter
canvas = tkinter.Canvas(width = 600, height = 400)
canvas.pack()

def klik(sur):
    obrazok1 = tkinter.PhotoImage(file = 'obrazky/zaba1.png')
    canvas.create_image(sur.x, sur.y, image = obrazok1)

canvas.bind('<Button-1>', klik)

```

Vždy po kliknutí ľavého tlačidla myši sa zavolá funkcia **klik**. Tam sa vytvorí lokálna premenná **obrázok1**, do ktorej sa načíta obrázok žaby. Príkazom **canvas.create\_image** sa obrázok zobrazí a hned' potom sa skončí vykonávanie funkcie **klik**, a teda sa zabudnú lokálne premenné, čiže aj premenná **obrazok1**. Takže Python z pamäte odstráni načítaný obrázok a my nič neuvidíme, lebo pre **canvas.create\_image**, ktorý si pamäta referenciu na obrázok, zmizli z pamäte dáta samotného obrázku. Preto sme hned' v úvode použili premennú **obrazok1** ako globálnu premennú.

Upravíme program tak, aby sme ľavým tlačidlom myši kreslili obrázok **zaba1.png** a pravým tlačidlo myši obrázok **zaba2.gif**.

```

import tkinter
canvas = tkinter.Canvas(width = 600, height = 400)
canvas.pack()

obrazok1 = tkinter.PhotoImage(file = 'obrazky/zaba1.png')
obrazok2 = tkinter.PhotoImage(file = 'obrazky/zaba2.gif')

def klik(sur):
    canvas.create_image(sur.x, sur.y, image = obrazok1)

def klik2(sur):
    canvas.create_image(sur.x, sur.y, image = obrazok2)

canvas.bind('<Button-1>', klik)
canvas.bind('<Button-3>', klik2)

```



Teraz vyskúšame jednu žabu kresliť s obrázkom v globálnej premennej a druhú s obrázkom v lokálnej premennej a do funkcie **klik2** doplníme aj zdržiavací **while** cyklus s priebežným **update** canvasu. Budeme vidieť, že druhá žaba sa aspoň na chvíľu po kliknutí ukáže (počas života lokálnej premennej).

```

import tkinter
canvas = tkinter.Canvas(width = 600, height = 400)
canvas.pack()

```

```

obrazok1 = tkinter.PhotoImage(file = 'obrazky/zaba1.png')

def klik(sur):
    canvas.create_image(sur.x, sur.y, image = obrazok1)

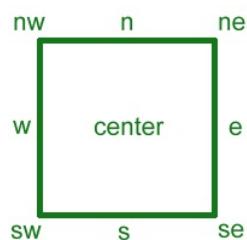
def klik2(sur):
    obrazok2 = tkinter.PhotoImage(file = 'obrazky/zaba2.gif')
    canvas.create_image(sur.x, sur.y, image = obrazok2)
    a = 50000
    while a>0:
        a -= 1
        canvas.update()

canvas.bind('<Button-1>', klik)
canvas.bind('<Button-3>', klik2)

```

Vidíme, že na mieste kliknutia sa umiestní stred obrázku. Niekedy potrebujeme, aby sa na mieste kliknutia nakreslil ľavý horný bod obrázku. V takom prípade stačí v príkaze zadať parameter **anchor** s hodnotou '**nw**' (northwest - severozápad) a podobne aj iné body v obrázku:

```
canvas.create_image(x, y, image = obrazok2, anchor = 'nw')
```



Parameter **anchor** môžeme využiť pri určení základného bodu aj v príkaze **canvas.create\_text()**.

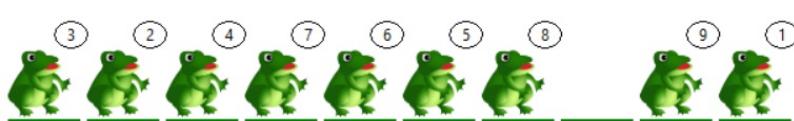
#### Otázky:

1. Môžeme v našich programoch používať ktorokoľvek obrázky stiahnuté z internetu? Ako to je s ich licenciou?
2. Čo znamená licencia creative commons license - s označením CC0? Ako môžeme nájsť na internete obrázky, ktoré majú takúto licenciu? Môžeme ich používať vo svojich programoch? Môžeme ich upravovať? Môžeme ich ďalej šíriť aj komerčne?
3. Ako zabezpečíme, aby mal obrázok načítaný zo súboru priesvitnú farbu (bol priehľadný)?

#### Úlohy:

**1**

Upravte program (hru) žaby z predchádzajúcej kapitoly tak, aby sa žaby nekreslili pomocou grafických útvarov, ale aby sa načítali z obrázkového súboru gif alebo png. Môžete si aj sami nakresliť obrázok v grafickom editore, alebo ho nájsť na internete.



**2**

Upravte program (hru) Need for Speed z predchádzajúcej kapitoly a doplňte do nej obrázok auta (gif alebo png).

## 6.2 Zoznam obrázkov

V súboroch `kocka_1.png`, `kocka_2.png`, `kocka_3.png`, `kocka_4.png`, `kocka_5.png`, `kocka_6.png` sú obrázky hracích kociek s hodnotami 1 až 6.



### Otázky:

4. Ako môžeme pri načítaní týchto obrázkov do pamäte využiť for cyklus?

```
import tkinter
canvas = tkinter.Canvas(width = 610, height = 150)
canvas.pack()

obrazky = []

def nacitaj(nazov, typ, pocet, obrazky):
    for i in range(1, pocet+1):
        obrazok = tkinter.PhotoImage(file = nazov + str(i) + '.'+typ)
        obrazky.append(obrazok)

def nakresli(x,y,obrazky):
    for obrazok in obrazky:
        canvas.create_image(x, y, image = obrazok, anchor = 'nw')
        x += 100

nacitaj('obrazky/kocka_', 'png', 6, obrazky)
nakresli(10, 30, obrazky)
```

Program načíta do zoznamu `obrazky` všetky obrázky `kocka_1.png` až `kocka_6.png`. Jednotlivé obrázky číta v cykle do lokálnej premennej `obrazok` a následne referenciu na načitaný obrázok pridá do zoznamu. V skutočnosti sú teda obrázky uložené ako referencia v globálnej premennej (zozname) `obrazky`, a teda nevadí, že sme ich pôvodne načítali do lokálnej premennej. Názov súboru si vyrábame z názvu priečinka a prvej časti súboru, následne čísla (z hodnoty premennej `i`) a typu súboru. Pri vykreslení pomocou for cyklu prejdeme cez všetky prvky zoznamu a obrázky zobrazíme.

### Otázky:

5. Zamyslite sa a zistite, čo bude robiť nasledujúci program.

```
import tkinter
from random import *
canvas = tkinter.Canvas(width = 310, height = 150)
canvas.pack()

def nacitaj(nazov, typ, pocet, obrazky):
    for i in range(1, pocet+1):
        obrazok = tkinter.PhotoImage(file = nazov + str(i) + '.'+typ)
        obrazky.append(obrazok)

def kresli_jednu(x, y, hodnota, obrazky):
    canvas.create_image(x, y, image = obrazky[hodnota-1], anchor = 'nw',
                        tags = 'kocka')

def animacia():
    global padnute_hodnoty
    canvas.delete('kocka')
```

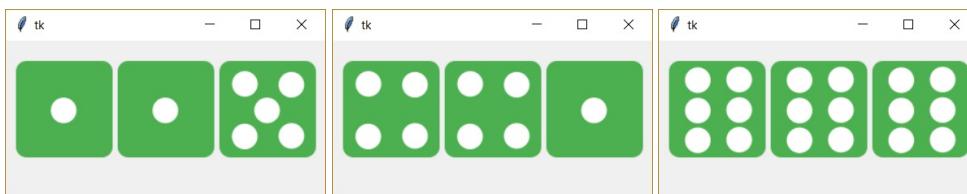
```

for i in range(3):
    padnute_hodnoty[i] = randint(1, 6)
    kresli_jednu(i*100+10, 20, padnute_hodnoty[i], obrazky)
    canvas.after(400, animacia)

def klaves(event):
    global body
    canvas.delete('body')
    if padnute_hodnoty.count(padnute_hodnoty[0]) == 3:
        body += 2
    elif padnute_hodnoty.count(padnute_hodnoty[0]) == 2:
        body += 1
    elif padnute_hodnoty.count(padnute_hodnoty[1]) == 2:
        body += 1
    else:
        body -= 1
    canvas.create_text(150, 10, text = body, tags = 'body')

obrazky = []
nacitaj('obrazky/kocka_', 'png', 6, obrazky)
padnute_hodnoty = [0]*3
body = 0
animacia()
canvas.bind_all('<space>', klaves)

```

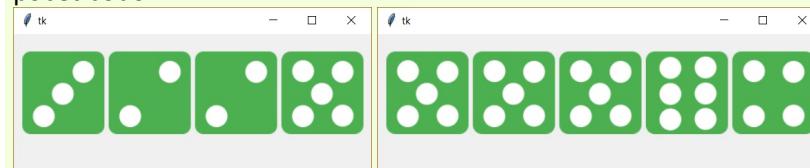


#### Otázky:

6. Akým iným spôsobom môžeme vyriešiť test vo funkcií `klaves`? Diskutujte o rôznych spôsoboch riešenia.

#### Úlohy:

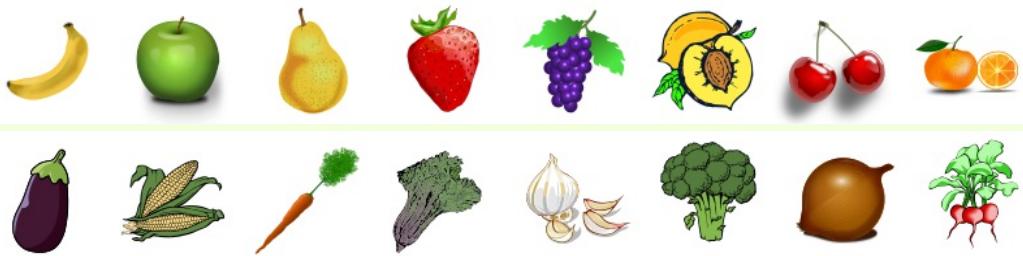
- 3** Upravte predchádzajúci program tak, aby sme pred spustením animácie mohli zadať počet hracích kociek. Program vyhodnotí aj počet rovnakých hodnôt na viacerých kockách a podľa toho nám zvýší počet bodov.



- 4** Upravte program tak, aby sme mohli tlačidlom (alebo niektorým klávesom) prepínať medzi pôvodnými obrázkami kociek a obrázkami kociek so symbolmi počasia, ktoré sú v súboroch `kocka_7.png`, `kocka_8.png`, `kocka_9.png`, `kocka_10.png`, `kocka_11.png`, `kocka_12.png`. Prípadne si vytvorte hracie kocky s vlastnými obrázkami.



- 5** V súboroch `oz_1.png`, `oz_2.png` až `oz_8.png` sú obrázky ovocia a v súboroch `oz_9.png`, `oz_10.png` až `oz_16.png` sú obrázky zeleniny:



Vytvorte program, ktorý vedľa seba zobrazí náhodnú 9-prvkovú postupnosť obrázkov ovocia a zeleniny (navzájom sú náhodne premiešané). Postupnosť sa vykreslí na obrazovku. Pod každým obrázkom a tiež vedľa posledného obrázku je nakreslená vodorovná čiara - polička, na ktorej je umiestnené ovocie alebo zelenina. Kliknutím na obrázok vybraný obrázok presunieme na prázdnú poličku, ale iba v prípade, že medzi obrázkom a prázdnou poličkou je maximálne jedna obsadená polička. Cieľom hráča je usporiadať obrázky tak, aby na prvých poličkách bolo ovocie a až za ním bola umiestnená zelenina. Náhodná postupnosť na začiatku:



Jedna z výherných pozícii:



## 6.3 Vizualizácia predpovede počasia

V spoločnosti na predpovedanie počasia chcú vytvoriť program, ktorý im umožní podľa aktuálnej predpovede počasia vytvoriť vizuálnu predpoved.



V súbore [slovakia.png](#) je obrázok slepej mapy Slovenska:



V súboroch [png](#) s názvami [jasno.png](#), [polojasno.png](#), [polooblacno.png](#), [oblacno.png](#), [dazd.png](#), [snezenie.png](#), [burka.png](#) sa nachádzajú obrázky počasia.



V programe sa najprv ukáže prázdna mapa krajiny. V pravom hornom rohu bude umiestnený obrázok počasia, ktorý ideme umiestňovať (aktuálne umiestňované počasie). Kliknutím myši v canvase umiestníme aktuálny obrázok počasia a pod obrázkom vypíšeme predpovedanú teplotu. Dolnú hranicu teploty zadáme pomocou **entry**, hornú hranicu program vypočíta (teplota je udávaná v štvorstupňovom rozsahu).

Program zadané informácie zapíše do textového súboru. Tento textový súbor predpovedná spoločnosť zašle svojim odberateľom (televíziám), a tie budú predpovedeť vizualizovať podľa informácií v súbore. Ak sa v editovaní počasia pomýlime, stlačením medzery môžeme začať úplne od začiatku (doposiaľ zadané informácie sa zmažú).

#### Otázky:

7. Ktoré údaje si budeme musieť v programe pamätať?
8. V akých štruktúrach bude vhodnejšie si ich pamätať?
9. Ako budeme vykresľovať aktuálne umiestňované počasie? Je vhodnejšie použiť jeden canvas alebo pridať ďalší?
10. Ktoré údaje a v akom formáte by sme mali zapisovať do textového súboru? Ktorý formát bude vhodnejší na opäťovné čítanie súboru? Čo si v súbore nemusíme zapísť (aby sme šetrili miestom)?
11. Akým spôsobom zabezpečíme, aby sme kliknutím na aktuálne umiestňované počasie ho cyklicky menili na ďalšie počasie (stále dookola)?
12. V ktorej časti programu, ako často a akým spôsobom je vhodné otvoriť a zatvoriť súbor?

Vytvoríme dva canvasy. V jednom bude mapa SR a umiestnené počasie. Druhý canvas bude umiestnený vpravo hore, bude mať menšie rozmery a v ňom budeme zobrazovať aktuálne umiestňované počasie. Na zobrazenie tohto canvasu nepoužijeme metódu **pack()**, ale metódu **place()**, v ktorej určíme aj **x** a **y** umiestnenia tohto canvasu v rámci celého okna aplikácie. Napríklad **canvas2.place(x = 300, y = 0)**. Po vytvorení prvého canvasu si do premennej **mapa** načítame mapu SR, následne ju umiestníme do canvasu. Pomocou metód **width()** a **height()** si zistíme veľkosť mapy SR a následne na rovnakú veľkosť upravíme veľkosť canvasu (Napr. **canvas['width'] = 600, canvas['height'] = 400**).

```
import tkinter
canvas = tkinter.Canvas()
canvas.pack()
canvas2 = tkinter.Canvas(width = 100, height = 100)

mapa = tkinter.PhotoImage(file = 'obrazky/slovakia.png')
canvas.create_image(0, 0, image = mapa, anchor = 'nw')
sirka = mapa.width()
vyska = mapa.height()
canvas['width'] = sirka
canvas['height'] = vyska
canvas2.place(x = sirka-100, y = 0)
```

V premennej **sirka** si pamäタme šírku mapy. Druhý canvas umiestníme na **x**, ktoré je o 100 px vľavo od šírky mapy. Pozor, nemôžeme použiť aj metódu **pack()** aj **place()** (iba jednu z nich).

Obrázky si budeme pamätať v globálnej premennej zoznam **obrazky**. Potrebujeme si pamätať aj názvy počasia, napríklad v premennej **pocasia**. Môže to byť n-tica alebo zoznam. Vymenovaním názvov si zabezpečíme, že môžeme obrázky čítať v cykle. Na načítanie obrázkov do poľa si vytvoríme funkciu **nacitaj()**. Index počasia, ktoré je pripravené na umiestňovanie, si budeme pamätať v globálnej premennej **aktualne**. Ako prvé aktuálne nastavíme počasie s indexom **0** (čiže jasno) a rovno ho do **canvas2** vykreslíme.

```

def nacitaj(cesta, typ, nazvy, obrazky):
    for nazov in nazvy:
        obrazok = tkinter.PhotoImage(file = cesta+nazov+'.'+typ)
        obrazky.append(obrazok)

pocasia = ('jasno', 'polojasno', 'polooblacno', 'oblacno', 'dazd',
           'snezenie', 'burka')
aktualne = 0
obrazky = []
nacitaj('obrazky/', 'png', pocasia, obrazky)

canvas2.create_image(50, 50, image = obrazky[aktualne])

```

Program po spustení vykreslí mapu a vpravo hore vidíme obrázok jasného počasia. Teraz budeme reagovať na udalosť ľavého kliku myši v prvom a druhom canvase a na medzere, ktorá vytvorené veci vymaže. Tiež vytvoríme **entry**, do ktorého na začiatok vložíme číslo 0 (**entry1.insert(0, '0')**).

Pri kliknutí na prvý canvas sa nakreslí na mieste kliknutia aktuálne vybraté počasie (zo zoznamu **obrazky** vyberieme obrázok s indexom **aktualne**). Z **entry** zistíme hodnotu, zmeníme ju na číslo a vypočítame aj hornú hranicu teplotného intervalu. Rozsah teplôt nakreslíme pod obrázok počasia. Zároveň pri kliknutí otvoríme súbor na pridanie riadku a do riadku zapíšeme súradnice, názov počasia a dolnú hranicu teplotného intervalu. Následne súbor zatvoríme.

Kliknutím na **canvas2** zvýšime hodnotu premennej **aktualne** a ak je vyššia ako počet obrázkov počasia, opäť ju nastavíme na **0**. Následne zmažeme **canvas2** a nakreslíme tam aktuálny obrázok počasia **obrazky[aktualne]**. Pri stlačení medzery otvoríme súbor na zápis a hned' ho zatvoríme - tým zmažeme jeho obsah. Tiež zmažeme nakreslené počasia pomocou tagu '**'predpovede'**', aby sme si nezmazali mapu.

```

def klik(sur):
    x, y = sur.x, sur.y
    canvas.create_image(x, y, image = obrazky[aktualne],
                       tags = 'predpovede')
    teplota1 = int(entry1.get())
    teplota2 = teplota1+4
    teploty = str(teplota1) +'/' +str(teplota2)
    canvas.create_text(x, y+50, text = teploty, font = 'Arial 20 bold',
                       fill = 'white', tags = 'predpovede')
    f = open('predpoved.txt', 'a')
    f.write(str(x)+ ' ' +str(y)+ ' '+pocasia[aktualne]+ ' '+str(teplota1)+ '\n')
    f.close()

def dalsiepočasie(event):
    global aktualne
    aktualne += 1
    if aktualne == len(pocasia):
        aktualne = 0
    canvas2.delete('all')
    canvas2.create_image(50, 50, image = obrazky[aktualne])

def vymaz(event):
    f = open('predpoved.txt', 'w')
    f.close()
    canvas.delete('predpovede')

canvas.bind('<Button-1>', klik)
canvas2.bind('<Button-1>', dalsiepočasie)
entry1 = tkinter.Entry()
entry1.pack()
entry1.insert(0, '0')
canvas.bind_all('<space>', vymaz)

```

Teraz už môžeme naklikáť predpoveď a pozrieť si textový súbor:

```
106 371 snezenie -10
```

```

234 425 snezenie -8
440 71 oblacno -17
310 130 oblacno -19
539 155 oblacno -23
169 251 snezenie -8
310 318 snezenie -10
422 256 oblacno -15
548 287 oblacno -17
654 225 polooblacno -17
635 104 polooblacno -18
764 207 polojasno -15
793 96 polojasno -16
853 270 jasno -12
884 151 jasno -12

```

Tomuto textovému súboru zodpovedá táto vizualizovaná predpoved:



Vytvorili sme editor počasia, ktorý použije spoločnosť na predpovedanie počasia. Vytvorený textový súbor zašle televízii, ktorá k nemu potrebuje program (Viewer). Program prečíta textový súbor a vytvára k počasiu animáciu príchodu predpovede (posunom) z pravého okraja na obrazovku a animáciu odchodu počasia k ľavému okraju obrazovky. Čiže program ovládame ľavým a pravým tlačidlom myši. Ľavé tlačidlo posunie predpovede z aktuálneho miesta o celú šírku obrazovky vľavo a pravé tlačidlo naopak - vpravo.

#### Otázky:

13. Čo budú mať spoločné oba programy a v čom budú rozdielne?
14. Ako budeme čítať súbor?
15. Čo a v akej štruktúre si potrebujeme pamätať?
16. Ako je najvhodnejšie vytvoriť animáciu príchodu a odchodu predpovede?

Prvá časť programu bude takmer rovnaká s editorom. V programe len vynecháme vytvorenie druhého canvasu. Obrázky si budeme pamätať rovnako - v zozname **obrazky**, názvy počasia v n-tici **pocasia**. Nepotrebujeme premennú **aktualne**.

```

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

mapa = tkinter.PhotoImage(file = 'obrazky/slovakia.png')
canvas.create_image(0, 0, image = mapa, anchor = 'nw')
sirka = mapa.width()
vyska = mapa.height()
canvas['width'] = sirka
canvas['height'] = vyska

def nacitaj(cesta, typ, nazvy, obrazky):
    for nazov in nazvy:
        obrazok = tkinter.PhotoImage(file = cesta+nazov + '.'+typ)
        obrazky.append(obrazok)

```

```

pocasia = ('jasno', 'polojasno', 'polooblacno', 'oblacno', 'dazd',
           'snezenie', 'burka')
obrazky = []
nacitaj('obrazky/', 'png', pocasia, obrazky)

```

Teraz môžeme prečítať súbor, metódou `split()` rozdeliť jednotlivé údaje z riadku do zoznamu, zo zoznamu vytiahnuť súradnice a zmeniť ich na čísla. Na číslo zmeníme aj dolnú hranicu teplotného intrevalu a z neho vypočítame hornú hranicu. Následne zobrazíme obrázky a všetky ich posunnieme vpravo o šírku obrazovky (aby boli pripravené na príchod na obrazovku). Najprv môžeme program vyskúšať bez odsunutia obrázkov, aby sme sa presvedčili, že sme súbor správne prečítali a všetko sa zobrazilo.

```

f = open('predpoved.txt', 'r')
for riadok in f:
    predpoved1 = riadok.split()
    typ_pocasia = predpoved1[2]
    index_obrazku = pocasia.index(typ_pocasia)
    x = int(predpoved1[0])
    y = int(predpoved1[1])
    canvas.create_image(x, y, image = obrazky[index_obrazku],
                        tags = 'predpovede')
    teplota1 = int(predpoved1[3])
    teplota2 = teplota1+4
    teploty = str(teplota1) +'/' +str(teplota2)
    canvas.create_text(x, y+50, text = teploty, font = 'Arial 20 bold',
                       fill = 'white', tags = 'predpovede')

f.close()
canvas.move('predpovede', sirka, 0)

```

Už len doprogramujeme animáciu. Budeme reagovať na ľavé a pravé tlačidlo myši. Rozdiel bude len v smere posúvania, ten si zapamätáme do premennej `smer` (-1 alebo +1). V premennej `pocet_krokov` si nastavíme, na koľko krokov chceme animáciu vykonať. V premennej `posun` si vypočítame veľkosť posunu jedného kroku (šírka obrazovky // počet krokov posunu). Následne spustíme animáciu pomocou timera a v nej si počítame počet realizovaných krokov (premenná `i`). Keď sme všetky kroky animácie urobili, nebudeme plánovať ďalšie spustenie časovača.

```

def animacia():
    global pocet_krokov, i
    canvas.move('predpovede', posun, 0)
    i -= 1
    if i > 0:
        canvas.after(10, animacia)

smer = 0
pocet_krokov = 20
i = pocet_krokov
posun = 0

def posun_vlavo(event):
    global smer, pocet_krokov, i, posun
    smer = -1
    i = pocet_krokov
    posun = smer * sirka // pocet_krokov
    animacia()

def posun_vpravo(event):
    global smer, pocet_krokov, i, posun
    smer = 1
    i = pocet_krokov
    posun = smer * sirka // pocet_krokov
    animacia()

```

```
canvas.bind('<Button-1>', posun_vlavo)
canvas.bind('<Button-3>', posun_vpravo)
```



### Úlohy:

6

Upravte program na zobrazenie počasia v televízii tak, že nám bude animovať postupne tri rôzne predpovede zo súborov `predpoved1.txt`, `predpoved2.txt` a `predpoved3.txt`.

7

Upravte oba programy Predpoved počasia tak, aby sme k predpovedi mohli zadať aj dátum predpovede, ten sa tiež uloží do súboru a v televízii ho zo súboru zobrazia a bude súčasťou animácie. Následne z troch súborov bude možné vizualizovať postupne predpovede počasia na najbližšie tri dni.

## Lambda funkcie

Iste ste si všimli, že funkcie `posun_vlavo()` a `posun_vpravo()` sú skoro rovnaké, líšia sa len nastavením premennej `smer`. Môžeme program zovšeobecniť a vytvoriť len jednu funkciu, ktorej ešte pridáme parameter na nastavenie smeru. Problém však je v tom, že `bind` nedovoľuje napísat volanie funkcie s vlastnými parametrami a `bind` vždy posielá práve jeden svoj parameter. Na takéto a iné účely sa často používa špeciálny zápis, tzv. `lambda` funkcia. Ide o tzv. anonymnú funkciu (nie je potrebné ju deklarovať pomocou `def`). Takáto funkcia má v tele len jeden riadok a neobsahuje `return`. Napríklad takúto jednoriadkovú funkciu `xnay`:

```
def xnay(x, y):
    return x**y

print(xnay(2, 3))
```

pomocou `lambda` funkcie zapíšeme takto (funkciu si zapamätáme do premennej `xnay`):

```
>>> xnay = lambda x, y: x**y
>>> xnay(2, 3)
8
>>>
```

Teda `lambda` funkcia sa zapisuje v tvare `lambda parametre: výraz`.

My vytvoríme len jednu funkciu s názvom `posun_start` (pozor, nemôže sa volať iba `posun`, lebo v programe už máme premennú s takým menom).

```
def posun_start(novy_smer):
    global smer, pocet_krokov, i, posun
    smer = novy_smer
    i = pocet_krokov
    posun = smer * sirka // pocet_krokov
    animacia()
```

A teraz upravíme príkazy `bind`.

```
canvas.bind('<Button-1>', lambda event: posun_start(-1))
canvas.bind('<Button-3>', lambda event: posun_start(1))
```

Do `bind` sme doplnili `lambda` funkciu, ktorá príjme parameter `event` (musí ho prijať) a zavolá funkciu

`posun_start(-1)` resp. `posun_start(1)`. Ako vidno, parameter `event` sa nemusí ďalej do funkcie `posun_start` posielat, lebo ho v nej nepotrebuje.

Keby sme potrebovali spracovať aj parameter `event`, riešenie zapíšeme takto:

```
def posun_start2(event, nsmer):
    global smer, pocet_krokov, i, posun
    smer = nsmer
    i = pocet_krokov
    posun = smer * sirka // pocet_krokov
    animacia()

canvas.bind('<Button-1>', lambda event: posun_start2(event, -1))
canvas.bind('<Button-3>', lambda event: posun_start2(event, 1))
```

### Otzáky:

17. Ako by ste využili funkciu `lambda` v prípade, že kliknutím myši chceme spustiť časovač (v úplne jednoduchom programe, kde len spúšťame časovač)?

V nasledujúcich dvoch programoch vidíte ukážku využitia funkcie `lambda`. V programe sú dve tlačidlá, jedno posúva nakreslené útvary hore a druhé dole. Oba programy robia to isté, len sú riešené rôznymi spôsobmi.

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

def posun(dy):
    canvas.move('all', 0, dy)

canvas.create_line(10, 100, 210, 100, width = 5)
button1 = tkinter.Button(text = '-', command = lambda:posun(-5))
button1.pack()
button2 = tkinter.Button(text = '+', command = lambda:posun(5))
button2.pack()
```

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

canvas.create_line(10, 100, 210, 100, width = 5)
button1 = tkinter.Button(text = '-', command = lambda:canvas.move('all', 0, -5))
button1.pack()
button2 = tkinter.Button(text = '+', command = lambda:canvas.move('all', 0, 5))
button2.pack()
```

# 7 Matematické výpočty a geometria

V programovaní je za riešením mnohých úloh veľa matematických výpočtov. Niektoré operácie a funkcie Pythonu už poznáme, teraz si ukážeme ďalšie. Niektoré z nich sú súčasťou modulu `math`, niektoré sú štandardnou súčasťou Pythonu.

## Otázky:

1. Ako by ste vypočítali vzdialenosť dvoch bodov, ak poznáme súradnice týchto bodov?
2. Akú operáciu / funkciu potrebujeme využiť vo výpočte vzdialosti dvoch bodov?

Funkcia `abs(číslo)` vráti absolútну hodnotu čísla. V kapitole číslo 4 sme si naprogramovali vlastné riešenie.

```
>>> abs(-3.58)  
3.58
```

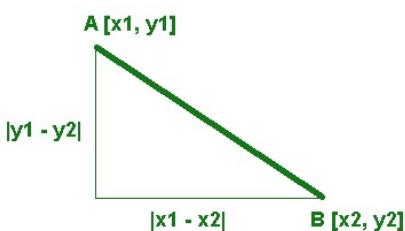
Funkcia `round(číslo, počet miest)` zaokrúhli zadané číslo na zadaný počet miest. Ak zadáme len prvý parameter, zaokrúhli vstupné číslo na celé číslo.

```
>>> round(2.58)  
3  
>>> round(2.48)  
2  
>>> round(2.58, 1)  
2.6  
>>> round(2.48, 1)  
2.5  
>>> round(2.48, -1)  
0.0  
>>> round(1253.48, -1)  
1250.0  
>>> round(1253.48, -2)  
1300.0
```

Funkcia `sqrt(číslo)` z modulu `math` vráti druhú odmocninu zo zadaného čísla.

```
>>> import math  
>>> math.sqrt(4)  
2.0  
>>> math.sqrt(9)  
3.0  
>>> math.sqrt(8)  
2.8284271247461903
```

Vzdialenosť dvoch bodov vypočítame z Pytagorovej vety.



Dĺžku odvesien si vieme vypočítať ako rozdiel x-ových súradníc a y-ových súradníc (teda ich absolútnej hodnoty). Keď poznáme v pravouhlom trojuholníku dve odvesny, Pytagorovou vetou vypočítame veľkosť prepony.

```

import math

def vzdialenos(a, b):
    return math.sqrt((a[0]-b[0])**2 + (a[1]-b[1])**2)

vzd = vzdialenos((50, 100), (150, 200))
print(round(vzd))

```

### Úlohy:

**1** Vytvorte program, v ktorom používateľ zadá polomer kruhu. Program vypočíta a vypíše jeho obvod a obsah.

**2** Vytvorte program, ktorý načíta od používateľa tri reálne čísla – dĺžky strán trojuholníka. Program vypočíta obsah trojuholníka pomocou Herónovho vzorca.  
([https://sk.wikipedia.org/wiki/Her%C3%A3nov\\_vzorec](https://sk.wikipedia.org/wiki/Her%C3%A3nov_vzorec))

**3** Vytvorte program, v ktorom používateľ môže naklikáť začiatočný a koncový bod čiary. Bod sa v grafickej ploche zobrazí ako malý krížik.

a) Následne (po druhom kliknutí alebo po kliknutí na tlačidlo) program vypočíta dĺžku čiary.

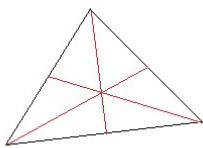
Dĺžka čiary je: 168.4



b) Upravte program tak, aby pri treťom kliknutí krížiky zmizli a vykreslil sa trojuholník.

c) Vypíšte obvod a obsah tohto trojuholníka.

d) Vykreslite vo vzniknutom trojuholníku inou farbou ďažnice.

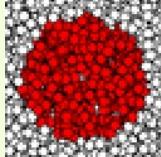


e) Pri štvrtom kliknutí sa trojuholník zmaže a body sa môžu naklikávať odnova.

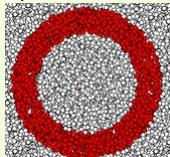
**4** Vytvorte program, ktorý vykreslí jeden dostatočne veľký náhodne vygenerovaný farebný kruh. Program potom pri každom kliknutí otestuje a vypíše, či používateľ klikol do tohto kruhu alebo mimo neho. (Príprava na odstreľovanie nepriateľských krúžkov v hre.)

**5** Vytvorte program, ktorý vygeneruje 1000 rovnako malých krúžkov na náhodných pozíciach. Krúžky budú červené alebo biele.

a) Krúžok bude červený, ak súradnice jeho stredu „padnú“ do veľkého mysleného kruhu v strede grafickej plochy, inak bude krúžok biely.



b) Červené krúžky sa nakreslia iba vtedy, keď padnú do kolesa (medzikružia).

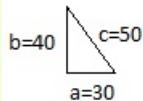


c) Červené a biele krúžky budú tvoriť takýto terč.



6

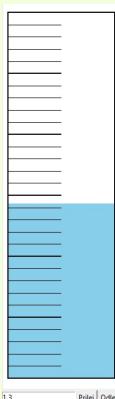
Vytvorte program, v ktorom používateľ zadá dve hodnoty – dĺžky odvesien  $a$ ,  $b$  (v pixeloch) pravouhlého trojuholníka. Program zobrazí trojuholník tak, že strana  $a$  bude vodorovná a strana  $b$  zvislá. Pri všetkých stranach bude zobrazená ich dĺžka.



7

Vytvorte program, ktorý zobrazí odmerný valec so stupnicou v centilitroch (viď obrázok).

a) Používateľ môže klikaním na tlačidlá priliať resp. odliat zadané množstvo vody (reálne číslo) v centilitroch. Program tiež vie zobraziť hlášku „Valec je plný“/ „Valec je prázdný“.

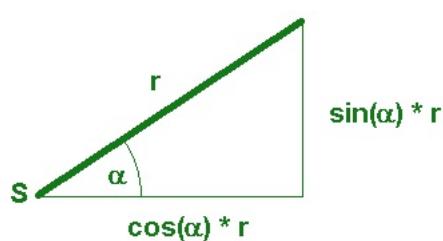


b) Program k dielikom vykresľuje aj hodnoty a používateľ môže určiť celkový objem odmerného valca a počet dielikov. Na obrázku bude odmerný valec vždy rovnako veľký. Program si vypočíta veľkosť jedného dielika a podľa toho aj upraví výpis hodnôt k dielikom. Ak sa priliata voda nezmestí do odmerného valca, program vedľa nakreslého ďalšieho valec a tam náleje nadbytočnú vodu.

### Otázky:

3. Ako by sme nakreslili otáčajúcu sa sekundovú ručičku? Aké vypočty potrebujeme na otáčanie čiary?

Skôr než nakreslíme otáčajúcu sa sekundovú ručičku, vykreslime všetky polohy sekundovej ručičky (v celých sekundách) na ručičkových hodinách. Tých polôh je 60, a teda jednej sekunde zodpovedá uhol  $360 / 60 = 6$  stupňov. My poznáme stred, okolo ktorého sa ručička otáča (bod  $s$  so súradnicami  $sx, sy$ ), a tiež dĺžku ručičky - teda polomer kružnice ( $r$ ) a uhol otočenia ručičky. Goniometrickými funkciemi sínus a kosínus si vypočítame dĺžku oboch odvesien pravouhlého trojuholníka. Kedže naša kružnica nie je jednotková, musíme dĺžku odvesien vynásobiť polomerom  $r$  (je  $r$ -krát väčšia ako jednotková kružnica). Z dĺžky odvesien a zo súradníc bodu  $s$  vypočítame koncový bod sekundovej ručičky.



```
import tkinter, math
```

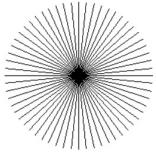
```

canvas = tkinter.Canvas(height = 500, width = 500, bg = 'white')
canvas.pack()

sx = 300
sy = 300
r = 100

for uhol in range(0, 360, 6):
    uhol_rad = math.radians(uhol)
    x = sx + math.cos(uhol_rad) * r
    y = sy - math.sin(uhol_rad) * r
    canvas.create_line(sx, sy, x, y)

```



V programe sme využili funkciu `math.sin(uhol)`, ktorá vypočíta sínus pre uhol zadaný v radiánoch a funkciu `math.cos(uhol)`, ktorá vypočíta kosínus tiež pre uhol zadaný v radiánoch. Kedže my sme `for` cyklom generovali veľkosť uhla v stupňoch, funkciou `math.radians()` sme si ju premenili na radiány. Na prepočet z radiánov na stupne slúži funkcia `math.degrees()`. Prepočet si však môžete aj sami naprogramovať, k tomu môžete využiť konštantu `math.pi`.

Program môžeme zmeniť. Doplňme časovač a pohybujúcu sa ručičku. Aby sme dlho nečakali, čas si nastavme na kratší ako jednu sekundu.

```

import tkinter, math
canvas = tkinter.Canvas(height = 500, width = 500, bg = 'white')
canvas.pack()

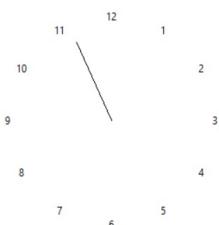
uhol = 90
sx = 200
sy = 200
r = 100

def rucicka():
    global uhol
    uhol -= 6
    uhol_rad = math.radians(uhol)
    x = sx + math.cos(uhol_rad) * r
    y = sy - math.sin(uhol_rad) * r
    canvas.delete('all')
    canvas.create_line(sx, sy, x, y)
    canvas.after(100, rucicka)

rucicka()

```

Doplníme do programu vykreslenie ciferníka:



```

import tkinter, math
canvas = tkinter.Canvas(height = 500, width = 500, bg = 'white')

```

```

canvas.pack()

uhol = 90
sx = 200
sy = 200
r = 100

def rucicka():
    global uhol
    uhol -= 6
    uhol_rad = math.radians(uhol)
    x = sx + math.cos(uhol_rad) * r
    y = sy - math.sin(uhol_rad) * r
    canvas.delete('rucicka')
    canvas.create_line(sx, sy, x, y, tags = 'rucicka')
    canvas.after(100, rucicka)

def cifernik(sx, sy, r):
    for i in range(1,13):
        alfa = 90 - (i * 360/12)
        alfa_rad = math.radians(alfa)
        x = sx + math.cos(alfa_rad) * (r + 20)
        y = sy - math.sin(alfa_rad) * (r + 20)
        canvas.create_text(x, y, text = i)

cifernik(sx, sy, r)
rucicka()

```

### Otázky:

4. Čo sa stane, keď vo vykresľovaní textu doplníme `angle = alfa` alebo `angle = alfa - 90`?

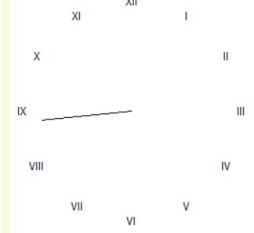
### Úlohy:

**8**

Upravte program so sekundovou ručičkou a ciferníkom tak, aby sa pri kliknutí myši hodiny presunuli na miesto kliknutia (stred hodín).

**9**

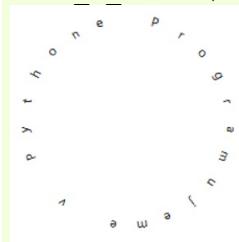
Upravte program so sekundovou ručičkou a ciferníkom tak, aby bol ciferník vypísaný rímskymi čislami.



**10**

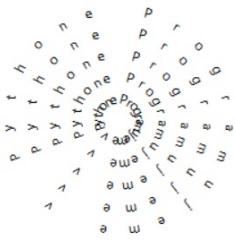
Vytvorte funkciu `text_v_kruhu`, ktorej zadáme súradnice stredu, polomer a text, a funkcia vykreslí text rovnomerne umiestnený po obvode kruhu a spodné časti jednotlivých znakov budú otočené smerom k stredu kruhu. Ukážka pre volanie

```
text_v_kruhu(200, 200, 80, 'Programujeme v Pythone '):
```



**11**

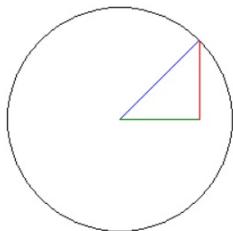
Pomocou funkcie `text_v_kruhu` z predchádzajúcej úlohy vytvorte funkciu, ktorá viacnásobne nakreslí tento text pre rôzne polometry.

**12**

V súbore `hviezda.png` je obrázok žltej päťcípej hviezdy. Vytvorte program, ktorý nakreslí vlajku EÚ.



Vytvorme program, ktorý bude animovať pohyb bodu po kružnici a jeho x-ovú súradnicu a y-ovú súradnicu (vzhľadom na stred kružnice, čiže x-ovú a y-ovú vzdialenosť od stredu kružnice) bude prenášať do grafu.



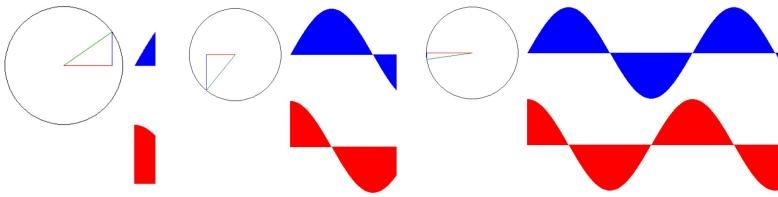
```
import tkinter, math
canvas = tkinter.Canvas(height = 500, width = 1000, bg = 'white')
canvas.pack()

uhol = 0
sx = 120
sy = 120
r = 100
px = 0

def animacia():
    global uhол, px
    canvas.delete('k')
    uhол_rad = math.radians(uhol)
    x1 = math.cos(uhол_rad) * r
    y1 = math.sin(uhол_rad) * r
    canvas.create_line(sx, sy, sx+x1, sy-y1, fill = 'green', tag = 'k')
    canvas.create_line(sx+x1, sy, sx+x1, sy-y1, fill = 'blue', tag = 'k')
    canvas.create_line(sx, sy, sx+x1, sy, fill = 'red', tag = 'k')
    canvas.create_line(sx+r+20+px, sy, sx+r+20+px, sy-y1, fill = 'blue')
    canvas.create_line(sx+r+20+px, sy+200, sx+r+20+px, sy+200-x1, fill = 'red')
    uhол += 1
    px += 1
    canvas.after(10, animacia)

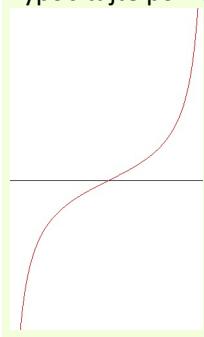
canvas.create_oval(sx-r, sy-r, sx+r, sy+r)
animacia()
```

Program postupne nakreslí takéto grafy:



### Úlohy:

- 13** Upravte program na kreslenie grafu v predchádzajúcej ukážke tak, aby sa oba grafy nakreslili ihneď po spustení programu pre jednu períodu  $2\pi$ . Následne pohybom myši v grafe sa nakreslí príslušná úsečka (výšky v oboch grafoch) a súčasne sa nakreslí na kružnici prislúchajúci uhol aj s trojhulníkom.
- 14** Vytvorte program – kreslič goniometrických funkcií. Program zobrazí časť X-ovej osi od 0 po  $2\pi$  a pomocou malých čiaročiek vykreslí krivku zvolenej goniometrickej funkcie. Y-ovú súradnicu pre dané X vypočítajte pomocou funkcií z modulu `math`.



Upravme ešte náš program, v ktorom sa pohybuje bod po kružnici. V premenných si okrem súradníc stredu, polomeru a počiatočného uhlu budeme pamätať aj hodnotu, o ktorú budeme uhol meniť (delta uhol - **du**). A do programu doplníme iný pohyb bodu po inej kružnici s nejakými inými nastaveniami. Aby sme premenné pre obe kružnice odlíšili, pridáme k nim číslo 1 a 2 (čiže premenné prvej kružnice a druhej kružnice).

```
import tkinter, math
canvas = tkinter.Canvas(height = 500, width = 1000, bg = 'white')
canvas.pack()

u1 = 0
du1 = 1
sx1 = 250
sy1 = 250
r1 = 100

u2 = 0
du2 = 4
sx2 = 300
sy2 = 200
r2 = 30

def animacia():
    global u1, u2
    canvas.delete('k')

    u1_rad = math.radians(u1)
    x1 = math.cos(u1_rad) * r1
    y1 = math.sin(u1_rad) * r1
    canvas.create_line(sx1, sy1, sx1+x1, sy1-y1, fill = 'green', tags = 'k')

    u2_rad = math.radians(u2)
    x2 = math.cos(u2_rad) * r2
    y2 = math.sin(u2_rad) * r2
```

```

        canvas.create_line(sx2, sy2, sx2+x2, sy2-y2, fill = 'orange', tags = 'k')

        u1 += du1
        u2 += du2
        canvas.after(10, animacia)

animacia()

```

Program zatiaľ nerobí nič zaujímavé, ale ešte ho mierne upravme. Stred druhej kružnice nebude stále na rovnakom mieste, ale bude sa nachádzať v pohybujúcom sa bode na prvej kružnici. Následne budeme vykreslovať miesta, kde sa nachádzal bod pohybujúci sa po druhej kružnici. A čo myslíte, ako bude vyzerat výsledný obrázok pre tento program?

```

import tkinter, math
canvas = tkinter.Canvas(height = 500, width = 1000, bg = 'white')
canvas.pack()

u1 = 0
du1 = 1
sx1 = 250
sy1 = 250
r1 = 100

u2 = 0
du2 = 4
sx2 = 300
sy2 = 200
r2 = 30

def animacia():
    global u1, u2, sx2, sy2
    canvas.delete('k')

    u1_rad = math.radians(u1)
    x1 = math.cos(u1_rad) * r1
    y1 = math.sin(u1_rad) * r1
    canvas.create_line(sx1, sy1, sx1+x1, sy1-y1, fill = 'green', tags = 'k')

    sx2 = sx1+x1
    sy2 = sy1-y1

    u2_rad = math.radians(u2)
    x2 = math.cos(u2_rad) * r2
    y2 = math.sin(u2_rad) * r2
    canvas.create_line(sx2, sy2, sx2+x2, sy2-y2, fill = 'orange', tags = 'k')
    canvas.create_line(sx2+x2, sy2-y2, sx2+x2+1, sy2-y2+1, width = 2)

    u1 += du1
    u2 += du2
    canvas.after(10, animacia)

animacia()

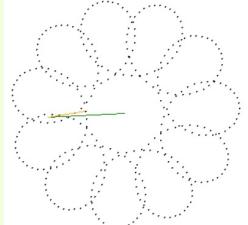
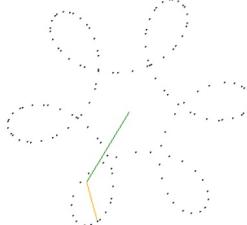
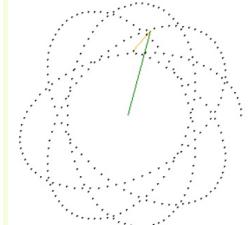
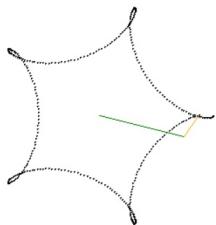
```

Program bude postupne kresliť tento obrázok:



**Úlohy:****15**

Experimentujte s nastaveniami v predchádzajúcom programe a zistujte, aké obrázky môže program nakresliť. Pre pohodlnejšie nastavenia vymyslite vhodný spôsob na zadávanie vstupu. Môžete vytvoriť takéto obrázky?

**16**

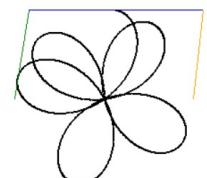
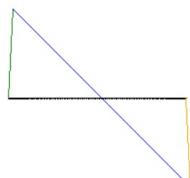
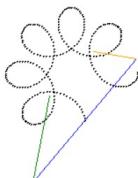
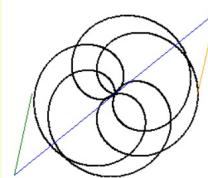
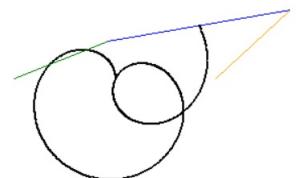
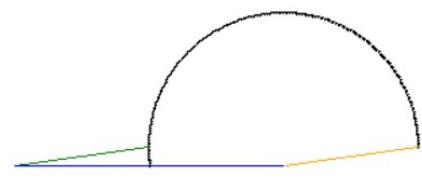
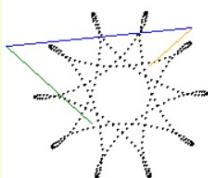
Upravte predchádzajúci program tak, aby príslušné body trajektórie uložil do zoznamu a následne nakreslil spojité čiaru cez všetky body, ktoré program vypočíta.

**17**

(\*) Upravte program pohybujúcich sa bodov po kružnici tak, že aj prvý bod, ktorý je pôvodne statický, sa bude pohybovať (buď po kružnici alebo priamke, v štvorci a pod.). Program môžete vylepšiť tým, že údaje si nebudete pamätať v jednotlivých premenných, ale v nejakej štruktúre.

**18**

Vytvorte program, v ktorom sa nezávisle pohybuje bod A po kružnici k1 a bod B po kružnici k2. Každá z kružníč má určený svoj stred, polomer a každý z bodov sa pohybuje určenou rýchlosťou (napríklad zadaný v počte stupňov otočenia v jednom kroku). Program vykresluje trajektóriu bodu C, ktorý je stredom úsečky AB. Experimentujte s rôznymi nastaveniami a program vylepšite tak, aby používateľ mohol ľahko meniť vstupné hodnoty.

**19**

Vytvorte funkciu, ktorá zadané číslo v desiatkovej sústave prevedie do dvojkovej sústavy (bez použitia formátovania a vstavanej funkcie `bin()`).

**20**

Vytvorte funkciu, ktorá zadané číslo v dvojkovej sústave prevedie do desiatkovej sústavy (bez použitia formátovania).

# 8 Asociatívne polia (slovník - dictionary)

## 8.1 Vytvorenie slovníka, metódy na prácu so slovníkom

Zoznam (list) a n-ticu sme používali na pamäťanie si nejakej štruktúry informácií. Na prístup k jednotlivým prvkom štruktúry sme používali číselný index. Sú situácie, keď potrebujeme, aby indexom neboli len čísla od 0 po n. Napríklad vytvoríme program, ktorému používateľ zadá mená a výšku osôb, následne nám program vypočíta a vypíše ich priemernú výšku. Na požiadanie nám k zadanému menu napiše výšku osoby. S tým, čo zatiaľ vieme, môžeme vytvoriť takýto program:

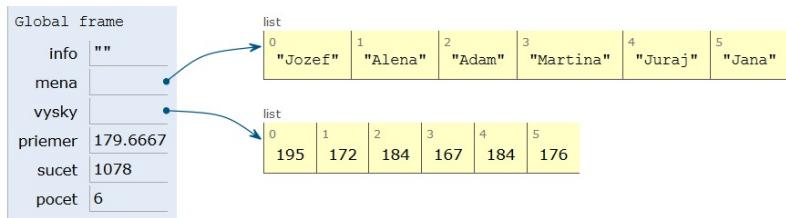
```
print('Meno a výšku oddel medzerou. Zadaním prázdnego vstupu ukončíš zadávanie.')
info = input('Zadaj meno a výšku osoby: ')
mena = []
vysky = []
while info != '':
    info = info.split()
    mena.append(info[0])
    vysky.append(int(info[1]))
    info = input('Zadaj meno a výšku osoby: ')

priemer = 0
sucet = sum(vysky)
pocet = len(vysky)
if pocet > 0:
    priemer = sucet / pocet
print('*'*30)
print('Priemerná výška je: '+'{:.2f} cm'.format(priemer))
print('*'*30)
print('Zistím Ti výšku zadaných osôb (pre skončenie stlač len enter).')
hľadat = input('Zadaj meno: ')
while hľadat != '':
    if hľadat in mena:
        index = mena.index(hľadat)
        print(hľadat+' má výšku: '+str(vysky[index]))
    else:
        print(hľadat+' nie je v zadaných informáciách!')
    hľadat = input('Zadaj meno: ')
```

```
Meno a výšku oddel medzerou. Zadaním prázdnego vstupu ukončíš zadávanie.
Zadaj meno a výšku osoby: Jozef 195
Zadaj meno a výšku osoby: Alena 172
Zadaj meno a výšku osoby: Adam 184
Zadaj meno a výšku osoby: Martina 167
Zadaj meno a výšku osoby: Juraj 184
Zadaj meno a výšku osoby: Jana 176
Zadaj meno a výšku osoby:
=====
Priemerná výška je: 179.67 cm
=====
Zistím Ti výšku zadaných osôb (pre skončenie stlač len enter).
Zadaj meno: Adam
Adam má výšku: 184
Zadaj meno: Monika
Monika nie je v zadaných informáciách!
Zadaj meno: Jana
Jana má výšku: 176
```

```
Zadaj meno:  
>>>
```

Zadané výšky a mená si pamäťame v dvoch rôznych zoznamoch. Pozrite si vizualizáciu (s použitím stránky <http://www.pythontutor.com>):

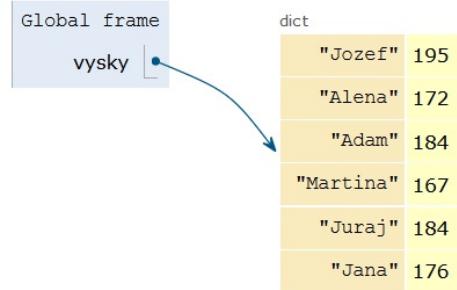


Výšku pre zadané meno hľadáme trochu komplikované. Najprv metódou **index** v zozname **meno** nájdeme index hľadaného mena a následne zo zoznamu **vysky** zistíme hodnotu prvku v zistenom indexe (rovnakom ako v zozname **meno**).

V takýchto prípadoch je vhodnejšie použiť **asociatívne pole (dictionary)**. V asociatívnom poli nepristupujeme k prvkom cez číselný **index**, ale pomocou **klúča**. Ku klúču je priradená (je s ním asociovaná) **hodnota**. Prvkami asociatívneho poľa sú **klúč** a jeho **hodnota**. Takto zapíšeme mená s výškami:

```
>>> vysky = {'Jozef':195, 'Alena':172, 'Adam':184, 'Martina':167, 'Juraj':184, 'Jana':176}
>>> type(vysky)
<class 'dict'>
>>> vysky
{'Alena': 172, 'Adam': 184, 'Juraj': 184, 'Jozef': 195, 'Jana': 176, 'Martina': 167}
>>>
```

Takto dátový typ nazývame **dictionary** alebo slovník. Vidíme, že prvky na rozdiel od zoznamu zapisujeme v zložených zátvorkách. V zložených zátvorkách sú dvojice **klúč:hodnota** (alebo len prázdne zložené zátvorky - tým vytvoríme prázdne asociované pole). Pozor, pri vypísaní **dictionary** môže byť poradie kľúčov a hodnôt v rôznom poradí (aj pri viacnásobnom spustení programu sa môže zmeniť). Ukážme si vizualizáciu (kľúče sú indexmi k hodnotám):



```
>>> vysky['Adam']
184
>>> vysky['Jana']
176
>>>
```

Použitie klasického číselného indexu hlási chybu (číselné indexy nebudú chybou v prípade, že kľúčmi sú také čísla). Chybu hlási aj použitie neexistujúceho kľúča:

```
>>> vysky[0]
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
```

```

vysky[0]
KeyError: 0
>>> vysky['Peter']
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    vysky['Peter']
KeyError: 'Peter'
>>>

```

Aby program nehlásil chybu pri neexistujúcom kľúči, môžeme si zistiť existenciu kľúča operáciou `in`. Operácia `in` prehľadáva len kľúče, nie hodnoty:

```

>>> 'Adam' in vysky
True
>>> 'Roman' in vysky
False
>>> 176 in vysky
False
>>>

```

Alebo použijeme metódu `get(kľúč, náhradná_hodnota)`, ktorá v prípade neexistencie kľúča vráti náhradnú hodnotu, alebo ak nie je zadaná, tak hodnotu `None`:

```

>>> vysky.get('Roman', 0)
0
>>> vysky.get('Roman')
>>> vysky.get('Adam', 0)
184
>>>

```

Takto vytvoríme prázdne asociované pole `vysky = {}`. Zápisom `vysky['Katka'] = 174` vytvoríme asociáciu kľúča `'Katka'` s hodnotou `174`. Ak v asociovanom poli kľúč `'Katka'` už existuje, zmeníme mu hodnotu na `174`.

```

>>> vysky = {}
>>> vysky['Katka'] = 174
>>> vysky
{'Katka': 174}
>>> vysky['Alena'] = 174
>>> vysky
{'Alena': 174, 'Katka': 174}
>>> vysky['Katka'] = 169
>>> vysky
{'Alena': 174, 'Katka': 169}
>>>

```

### Otázky:

1. Môžu byť v jednom asociovanom poli dva kľúče s rôznymi hodnotami (Napríklad: `'Katka': 174` a `'Katka': 169`)?
2. Ktoré dátové typy môžu byť kľúčmi asociovaného poľa (zistite experimentom)?
3. Ktoré z týchto typov: `integer`, `float`, `string`, `boolean`, `tuple` môžu byť hodnotami v asociovanom poli?
4. Môžeme na asociované polia použiť rezy (slice)?
5. Čo vypíše funkcia `len()` pre asociované pole?

Upravme náš program z úvodu tejto kapitoly. Namiesto dvoch zoznamov `mena` a `vysky` vytvoríme jedno asociované pole `vysky`. Najprv bude prázdne. Po načítaní vstupu vytvoríme asociáciu `kľúča` (meno osoby) a

**hodnoty** (výška osoby). Ukážka prvej časti programu (načítania):

```
print('Meno a výšku oddel medzerou. Zadaním prázdného vstupu ukončíš zadávanie.')
info = input('Zadaj meno a výšku osoby:')
vysky = {}
while info != '':
    info = info.split()
    vysky[info[0]] = info[1]
    info = input('Zadaj meno a výšku osoby:')
```

Funkcia **sum()** nefunguje s **dictionary**. Potrebujeme sa dostať len k hodnotám **dictionary**. Metóda **values()** vráti hodnoty asociovaného poľa a z nich môžeme spraviť súčet: **sum(vysky.values())**. Zvyšná časť programu (s použitím **dictionary**):

```
priemer = 0
sucet = sum(vysky.values())
pocet = len(vysky)
if pocet > 0:
    priemer = sucet / pocet
print('*'*30)
print('Priemerná výška je: '+'{:.2f} cm'.format(priemer))
print('*'*30)
print('Zistím Ti výšku zadaných osôb (pre skončenie stlač len enter).')
hľadat = input('Zadaj meno: ')
while hľadat != '':
    hodnota = vysky.get(hľadat, None)
    print(hľadat+' má výšku: '+str(hodnota))
    hľadat = input('Zadaj meno: ')
    print('*'*30)
```

Pre neexistujúci kľúč vypíše program priamo výšku **None**:

```
=====
Priemerná výška je: 179.67 cm
=====
Zistím Ti výšku zadaných osôb (pre skončenie stlač len enter).
Zadaj meno: Adam
Adam má výšku: 184
Zadaj meno: Maria
Maria má výšku: None
Zadaj meno: Jozef
Jozef má výšku: 195
Zadaj meno:
>>>
```

Funkciou **list** môžeme vytvoriť zoznam len z hodnôt **dictionary** (alebo funkciou **tuple** vytvoríme z nich n-ticu):

```
>>> list(vysky.values())
[184, 195, 184, 167, 172, 176]
>>> tuple(vysky.values())
(184, 195, 184, 167, 172, 176)
>>>
```

Metóda **keys()** nám vráti len kľúče z asociovaného poľa a tiež môžeme z nich spraviť funkciou **list()** zoznam a funkciou **tuple()** n-ticu:

```
>>> vysky.keys()
dict_keys(['Adam', 'Jozef', 'Juraj', 'Martina', 'Alena', 'Jana'])
>>> list(vysky.keys())
['Adam', 'Jozef', 'Juraj', 'Martina', 'Alena', 'Jana']
>>> tuple(vysky.keys())
```

```
('Adam', 'Jozef', 'Juraj', 'Martina', 'Alena', 'Jana')
>>>
```

Metódou `items()` získame všetky dvojice kľúča a hodnoty z `dictionary` a tiež z nich môžeme vytvoriť `zoznam` alebo `n-ticu`. V tomto prípade bude jedným prvkom dvojica `(kľúč, hodnota)`, čiže n-tica:

```
>>> vysky.items()
dict_items([('Adam', 184), ('Jozef', 195), ('Juraj', 184), ('Martina', 167),
('Alena', 172), ('Jana', 176)])
>>> list(vysky.items())
[('Adam', 184), ('Jozef', 195), ('Juraj', 184), ('Martina', 167),
('Alena', 172), ('Jana', 176)]
>>> tuple(vysky.items())
((('Adam', 184), ('Jozef', 195), ('Juraj', 184), ('Martina', 167),
('Alena', 172), ('Jana', 176)))
>>>
```

For cyklom môžeme prechádzať aj postupnosťou kľúčov, hodnôt, alebo dvojice kľúč, hodnota z asociovaného poľa:

```
>>> for kluc in vysky.keys():
    print(kluc)

Adam
Jozef
Juraj
Martina
Alena
Jana
>>>
```

```
>>> for hodnota in vysky.values():
    print(hodnota)

184
195
184
167
172
176
>>>
```

```
>>> for dvojica in vysky.items():
    print(dvojica)

('Adam', 184)
('Jozef', 195)
('Juraj', 184)
('Martina', 167)
('Alena', 172)
('Jana', 176)
>>>
```

Alebo to môžeme spraviť aj takto:

```
>>> for kluc, hodnota in vysky.items():
    print(kluc, hodnota)

Alena 172
Adam 184
Jozef 195
Martina 167
Juraj 184
```

**Úlohy:**

**1** Upravte program s výškami osôb tak, aby údaje zadané na vstupe zapísal do textového súboru.

**2** Upravte program s výškami osôb tak, aby údaje o výške osôb načítal do slovníka zo súboru.

V textovom súbore **slovicka.txt** je na každom riadku anglický výraz (slovíčko) a slovenský výraz (slovíčko). Jednotlivé výrazy sú oddelené bodkočiarkou. Ukážka súboru **slovicka.txt**:

```
table;stôl
chair;stolička
lamp;lampa
window;okno
air condition;klimatizácia
door;dvere
```

Vytvoríme program, ktorý prečíta slovíčka do asociatívneho poľa a následne nás bude skúšať slovíčka. Najprv vytvoríme slovník **en\_sk**, v ktorom bude kľúčom anglické slovíčko a hodnotou slovenské slovíčko:

```
subor = open('slovicka.txt', 'r')
en_sk = {}
for riadok in subor:
    riadok = riadok.strip()
    r = riadok.split(';')
    en_sk[r[0]] = r[1]
print(en_sk)
```

Nakoniec vypíšeme celé asociatívne pole (slovník):

```
{'wardrobe': 'šatník', 'picture': 'obraz', 'desk': 'lavica', 'door': 'dvere',
'chair': 'stolička', 'ceiling': 'strop', 'armchair': 'kreslo', 'table': 'stôl',
'hanger': 'vešiak', 'bookcase': 'knížnica', 'window': 'okno', 'clock': 'hodiny',
'lamp': 'lampa', 'air condition': 'klimatizácia', 'carpet': 'koberec',
'sofa': 'pohovka', 'shelf': 'polica'}
>>>
```

Teraz môžeme aj priamo z príkazového riadku zadávať anglické slovíčka ako kľúč slovníka a program nám vypíše slovenský preklad. Ak omylem zadáme slovenské slovíčko, program ohlási chybu, pretože slovenské slovíčka nie sú kľúče, ale hodnoty:

```
>>> en_sk['carpet']
'koberec'
>>> en_sk['chair']
'stolička'
>>> en_sk['pohovka']
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in <module>
    en_sk['pohovka']
KeyError: 'pohovka'
>>> en_sk['desk']
'lavica'
>>>
```

K prekladu zo slovenčiny do angličtiny si vyrobíme obrátený slovník - kľúčmi budú hodnoty pôvodného slovníka a hodnotami budú kľúče pôvodného slovníka:

```
sk_en = {}
for dvojica in en_sk.items():
```

```
|     sk_en[dvojica[1]] = dvojica[0]
| print(sk_en)
```

alebo to môžeme zapísť aj takto:

```
sk_en = {}
for en, sk in en_sk.items():
    sk_en[sk] = en
print(sk_en)
```

Vytvorili sme takéto asociatívne pole:

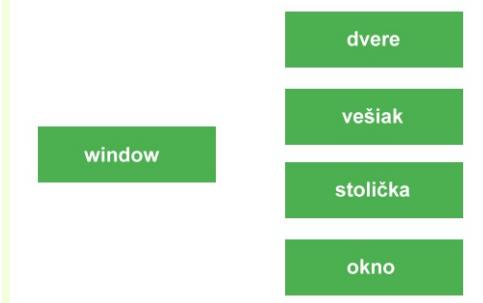
```
{'vešiak': 'hanger', 'šatník': 'wardrobe', 'lavica': 'desk', 'lámpa': 'lamp',  
'hodiny': 'clock', 'obraz': 'picture', 'strop': 'ceiling', 'koberec': 'carpet',  
'polica': 'shelf', 'knížnica': 'bookcase', 'pohovka': 'sofa',  
'stolička': 'chair', 'dvere': 'door', 'okno': 'window', 'stôl': 'table',  
'klimatizácia': 'air condition', 'kreslo': 'armchair'}  
>>>
```

## Úlohy:

- 3 Máme program s dvomi asociatívnymi poliami. Jedno má názov en\_sk (anglicko-slovenský slovník) a druhé má názov en\_es (anglicko-španielsky slovník). Slovníky si bud' priamo skopírujte do programu, alebo ich načítajte zo súboru. V súbore slovicka\_en\_es.txt nájdete slová pre slovník en\_es.

```
en_sk = {'door': 'dvere', 'shelf': 'polica', 'desk': 'lavica', 'carpet': 'koberec', 'bookcase': 'knížnica', 'sofa': 'pohovka', 'table': 'stôl', 'picture': 'obraz', 'ceiling': 'strop', 'air condition': 'klimatizácia', 'wardrobe': 'šatník', 'lamp': 'lámpa', 'hanger': 'vešiak', 'clock': 'hodiny', 'armchair': 'kreslo', 'chair': 'stolička', 'window': 'okno'}  
en_es = {'hanger': 'percha', 'wardrobe': 'armario', 'table': 'mesa', 'desk': 'escritorio', 'air condition': 'aire acondicionado', 'armchair': 'sillón', 'shelf': 'estante', 'chair': 'silla', 'sofa': 'sofá', 'bookcase': 'librero', 'carpet': 'alfombra', 'lamp': 'lámpara', 'clock': 'reloj', 'door': 'puerta', 'ceiling': 'techo', 'window': 'ventana', 'picture': 'cuadro'}
```

- 4** V textovom súbore máme na samostatnom riadku slovo v slovenčine a v riadku pod ním jeho preklad do cudzieho jazyka. Takýchto dvojriadkov je v súbore ľubovoľné množstvo. Vytvorte program, ktorý nás náhodne skúša slovíčka z cudzieho jazyka (v grafickom okne) a vždy nám dá na výber 4 nejaké slová v slovenčine, pričom jedno z nich je správna odpoveď. Ak slovo uhádneme, pripočítá nám bod, ak ho neuhádneme, bude nám ho častejšie zaraďovať do zoznamu slov, ktoré nás má ešte vyskúšať.



## Šifrovanie náhodnou substitúciou

V textovom súbore **clanok.txt** je článok z internetu, ktorý chceme zašifrovať náhodnou substitúciou. To znamená, že každému znaku v texte priradíme nejaký náhodný znak, ktorý je v pôvodnom teste. Platí však, že jeden znak je v celom teste nahradený rovnakým náhodným znakom. Zašifrovaný text uložíme do súboru **clanok\_zasifrovany.txt**. Substitučnú tabuľku (pôvodný znak a nový znak - prevodová tabuľka) si uložíme

do súboru `clanok_kluc.txt`.

#### Otázky:

6. Zamyslite sa, ktoré kroky je potrebné v programe urobiť.
7. Ktoré znaky musí obsahovať prevodová tabuľka? Akým spôsobom môžeme minimalizovať použité znaky v prevodovej tabuľke?
8. Ako zabezpečíme, aby sme si v prevodovej tabuľke pamätali každý znak zo vstupného textu práve raz?

Pozrime si program:

```
import random
subor = open('clanok.txt', 'r')
celytext = subor.read()
subor.close()

znaky = []
for znak in celytext:
    if not znak in znaky:
        znaky.append(znak)
znaky_zamiesane = znaky[:]
random.shuffle(znaky_zamiesane)

substitucia = {}
for i in range(len(znaky)):
    substitucia[znaky[i]] = znaky_zamiesane[i]

celytext_sifrovany = ''
for znak in celytext:
    celytext_sifrovany = celytext_sifrovany + substitucia[znak]

subor_sifrovany = open('clanok_sifrovany.txt', 'w')
subor_sifrovany.write(celytext_sifrovany)
subor_sifrovany.close()

subor_tabulka = open('clanok_kluc.txt', 'w')
for kluc, hodnota in substitucia.items():
    subor_tabulka.write(kluc+';'+hodnota+';')
subor_tabulka.close()
```

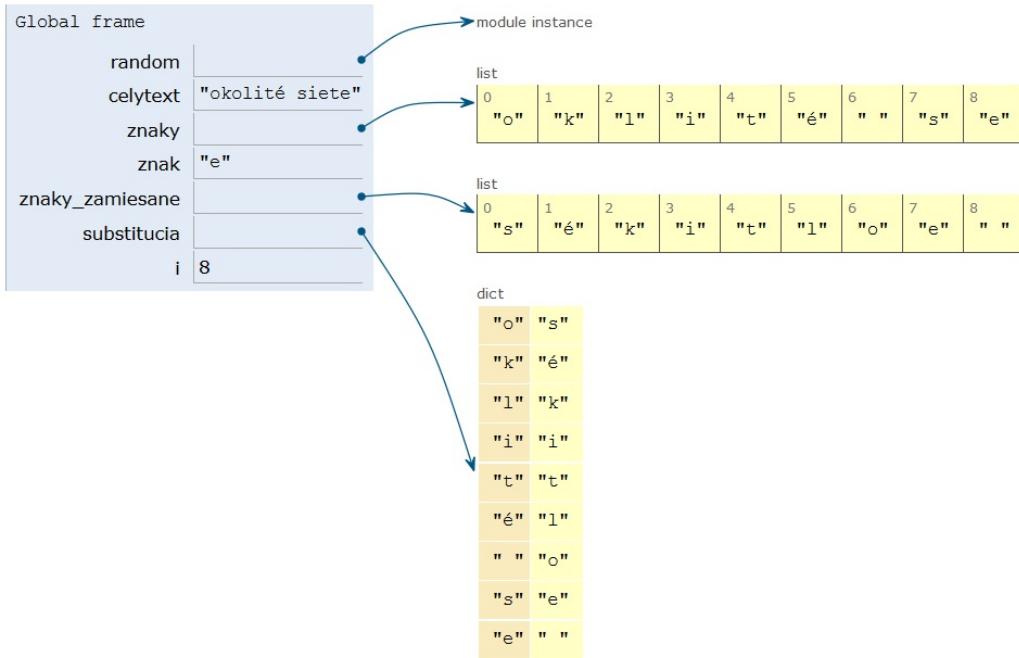
Celý súbor prečítame do jednej premennej `celytext`. Následne precházame po jednotlivých znakoch tejto premennej a v zozname `znaky` si vytvárame vstupnú abecedu (čiže zoznam všetkých použitých znakov vo vstupnom súbore - každý znak do neho pridáme iba raz). Najprv si zistíme, či sme do zoznamu `znaky` už znak pridali, a ak nie, pridáme ho tam. Následne vytvoríme kópiu zoznamu (`znaky_zamiesane = znaky[:]`), v ktorej znaky náhodne zamiešame `random.shuffle(znaky_zamiesane)`. Na zamiešanie použijeme funkciu `shuffle` z modulu `random`.

#### Otázky:

9. Ktoré dátové typy vie funkcia `random.shuffle()` zamiešať? (zistite experimentmi na príkazovom riadku)
10. Ako by sme náhodne zamiešali prvky zoznamu bez funkcie `random.shuffle()`?
11. Prečo sme vytvorili kópiu zoznamu a nestačí zapísť `priradenie znaky_zamiesane = znaky`?
12. Ako program spracuje znak konca riadku v súbore '`\n`'?

Ak by sme v textovom súbore mali len tento text: `okolité siete`, alebo sme túto vetu priradili do premennej

`celytext`, prevodová tabuľka v slovníku by mohla byť zamiešaná napríklad takto:



Teraz stačí prechádzať po znakoch premennej `celytext` a daný znak je kľúčom v slovníku `substitucia` k novej hodnote (ktorú sme získali zamiešaním a je zapamätaná v tomto slovníku).

```
celytext_sifrovany = ''
for znak in celytext:
    celytext_sifrovany = celytext_sifrovany + substitucia[znak]
```

Zašifrovaný text už len uložíme do súboru. Následne uložíme do súboru celý slovník, čiže prevodovú tabuľku.

```
subor_tabulka = open('clanok_kluc.txt', 'w')
for kluc, hodnota in substitucia.items():
    subor_tabulka.write(kluc+';'+hodnota+';')
subor_tabulka.close()
```

### Otázky:

13. Ako by ste spravili program na opačný proces - rozšifrovanie?
14. Za akých okolností nebudú programy na šifrovanie a rozšifrovanie fungovať správne? Ktoré znaky môžu / nemôžu byť použité v textovom súbore pre toto šifrovanie?
15. Ak by sme zo zašifrovaného súboru vytvorili zoznam `znaky` v programe na šifrovanie, v čom by sa líšil zoznam `znaky` zašifrovaného a nezašifrovaného súboru? Alebo by boli zoznamy rovnaké?
16. Je takéto šifrovanie symetrické? Môžeme program na šifrovanie použiť aj na rozšifrovanie?
17. Ak by sme viackrát šifrovali vstupný súbor, budú výstupy rovnaké? Prečo?

Výsledný program na rozšifrovanie:

```
import random
subor = open('clanok_sifrovany.txt', 'r')
celytext_sifrovany = subor.read()
subor.close()

subor = open('clanok_kluc.txt', 'r')
tabulka = subor.read()
tabulka = tabulka[:-1]
```

```

subor.close()
pole = tabulka.split(';')
pole1 = pole[::2]
pole2 = pole[1::2]

substitucia = {}
for i in range(len(pole1)):
    substitucia[pole2[i]] = pole1[i]

rozsifrovane = ''
for znak in celytext_sifrovany:
    rozsifrovane = rozsifrovane + substitucia[znak]
print(rozsifrovane)

```

#### Otázky:

18. Prečo je v programe tento riadok `tabulka = tabulka[:-1]`? Čo by sa stalo, ak by sme ho vyniechali?
19. Čo spravia tieto dve priradenia `pole1 = pole[::2]` `pole2 = pole[1::2]`? Akým spôsobom ich môžeme nahradíť, ak ich nechceme použiť?
20. Keby sme čítali súbor `for` cyklom po riadkoch, ako by sa zmenila funkčnosť a efektívnosť programu? Diskutujte o tom v skupine.

## 8.2 Frekvencia výskytov

### Frekvencia znakov

Pomocou slovníka môžeme vytvoriť frekvenčnú tabuľku znakov, slov a iných údajov. Frekvenčná tabuľka je zoznamom sledovaných hodnôt (čísel, reťazcov, a i., napríklad znakov alebo slov) a informácií o počte výskytov (čiže frevencii) týchto hodnôt v sledovanom súbore týchto hodnôt. Napríklad môžeme vytvoriť frekvenčnú tabuľku o počte výskytov jednotlivých písmen v texte alebo väčšom rozsahu textov.

#### Otázky:

21. Čo si myslíte? Ak analyzujeme väčšie množstvo textu (v jednom jazyku), aká bude frekvencia jednotlivých znakov abecedy v týchto textoch? Budú sa vyskytovať jednotlivé znaky v teste približne v rovnakom počte, alebo ich počty budú mať iné rozdelenie?
22. Ak spravíme frekvenčnú tabuľku znakov vo väčšom teste v dvoch rôznych jazykoch a porovnáme tieto tabuľky, budú sa v niečom lísiť, alebo budú podobné?
23. Samuel Morse vymyslel kódovanie anglickej abecedy (Morseova abeceda, alebo morseovka) do postupnosti krátkeho a dlhého signálu (bodka a čiarka) a od roku 1844 sa používalo toto kódovanie v telegrafii. Znaky boli kódované rôznym počtom bodiek a čiarok. Čo si myslíte, ktoré znaky mali kratší zápis v kódovaní a ktoré dlhší? Svoju odpoveď zdôvodnite.

V textovom súbore `anglicke_texty.txt` máme pripravené tri anglické články zo stránky [www.bbc.com](http://www.bbc.com). Zo súboru vytvoríme frekvenčnú tabuľku znakov písmen anglickej abecedy, zvyšné znaky budeme ignorovať. Aby sme nemuseli zvlášť spočítavať veľké a malé písmená, zmeníme si všetky písmená na malé. Frekvenčnú tabuľku vypíšeme iba jednoducho príkazom `print`.

```

subor = open('anglicke_texty.txt', 'r')
frekvencia = {}
for riadok in subor:
    riadok = riadok.lower()
    for znak in riadok:
        if 'a' <= znak <= 'z':

```

```

        frekvencia[znak] = frekvencia.get(znak, 0) + 1
subor.close()
print(frekvencia)

```

### Otázky:

24. Budú alebo musia byť v slovníku po spracovaní súboru zastúpené všetky písmená malej abecedy? Čím je to ovplyvnené?

Ukážka výstupu programu na príkazovom riadku a tiež zistenie počtu kľúčov v slovníku:

```

{'d': 463, 'f': 243, 'o': 870, 'x': 28, 'v': 149, 'g': 242, 'h': 525, 'z': 2,
'i': 896, 'b': 213, 'n': 882, 'w': 204, 'c': 505, 'p': 279, 'u': 309, 's': 862,
'm': 264, 'e': 1446, 'j': 16, 'q': 6, 'a': 1069, 'k': 88, 'y': 239, 'r': 871,
'l': 526, 't': 1119}
>>> len(frekvencia)
26
>>>

```

Tento výpis je málo prehľadný a pomohlo by nám nejaké zoradenie podľa frekvencie výskytu. My už poznáme metódu `sort` na utriedenie zoznamov `zoznam.sort()`. Slovníky **nemajú** túto metódu, Python bude hlásiť chybu:

```

>>> frekvencia.sort()
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    frekvencia.sort()
AttributeError: 'dict' object has no attribute 'sort'
>>>

```

Môžeme to vyriešiť dvomi spôsobmi. Napríklad si zo slovníka vyrobíme zoznam n-tíc, kde jedna n-tica bude v poradí hodnota a kľúč. Tento zoznam zoradíme metódou `sort`, čiže sa zoradí podľa nultého prvku n-tíc (tam máme hodnotu, čiže frekvenciu výskytu znaku). Alebo použijeme štandardnú funkciu `sorted()`, ktorá z prvkov ľuboľnej postupnosti (reťazca, zoznamu, n-tice...) vytvorí nový zoznam týchto hodnôt, štandardne zoradený vzostupne (od najmenšieho prvku po najväčší). Keď zoradíme slovník funkciou `sorted()`, vytvorí z neho len zoradený zoznam kľúčov, a my potrebujeme zoradenie podľa hodnôt slovníka:

```

>>> sorted(frekvencia)
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
>>>

```

Pre funkciu `sorted()` použijeme ďalšie dva parametre

`sorted(frekvencia, key = frekvencia.get, reverse = True)`. Parametrom `key` určujeme, ktorá funkcia sa použije pri zoradovaní. Čiže funkcia `sorted` bude prechádzať kľúčmi slovníka `frekvencia` a pri zaraďovaní daného kľúča na určenie zoraďovacej hodnoty použije funkciu `get`, ktorá pre daný kľúč zistí hodnotu v slovníku. Čiže zoradenie bude podľa hodnôt slovníka, ale v zoradenej postupnosti budú iba kľúče. To nám až tak nevadí, lebo pre tieto kľúče si ľahko zistíme hodnotu zo slovníka. Parametrom `reverse` určujeme, či chceme postupnosť zoradiť vzostupne alebo zostupne.

```

>>> sorted(frekvencia, key = frekvencia.get, reverse = True)
['e', 't', 'a', 'i', 'n', 'r', 'o', 's', 'l', 'h', 'c', 'd', 'u', 'p', 'm', 'f',
 'g', 'y', 'b', 'w', 'v', 'k', 'x', 'j', 'q', 'z']

```

Výsledný program:

```

subor = open('anglicke_texty.txt', 'r')
frekvencia = {}
for riadok in subor:
    riadok = riadok.lower()

```

```

        for znak in riadok:
            if 'a' <= znak <= 'z':
                frekvencia[znak] = frekvencia.get(znak, 0) + 1
subor.close()
zoradene = sorted(frekvencia, key = frekvencia.get, reverse = True)
for kluc in zoradene:
    print(kluc, frekvencia.get(kluc))

```

Výstup z programu:

```

e 1446
t 1119
a 1069
i 896
n 882
r 871
o 870
s 862
l 526
h 525
c 505
d 463
u 309
p 279
m 264
f 243
g 242
y 239
b 213
w 204
v 149
k 88
x 28
j 16
q 6
z 2

```

Na stránke [https://sk.wikipedia.org/wiki/Medzin%C3%A1rodn%C3%A1\\_Morseova\\_abeceda](https://sk.wikipedia.org/wiki/Medzin%C3%A1rodn%C3%A1_Morseova_abeceda) je pekná tabuľka, ktorá nám pomôže dekódovať Morseovu abecedu.

				Ch	0 / 9
	M	O		Ö	8
T		G	Q	Ñ	
	N	K	Z		7
		D	Y		
			C		
E	A	W	X		
	I	R	B	6	
	S	U	J	1 / '	
			P		
			À		
			L		
			Ó	2	
			F		
			V	3	
			H	4 / 5	

### Otázky:

25. Ako vyčítame z tabuľky, že písmeno A má kód ". -" a písmeno D má kód "- . . -"?
26. Ako z tabuľky rýchlo zistíme, ktoré písmeno má kratší zápis v Moresovej abecede?

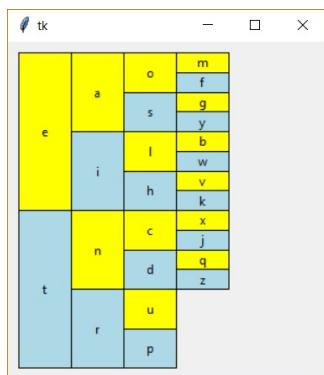
Vytvorme program, ktorý vygeneruje takúto tabuľku zo zistenej frekvencie znakov predchádzajúcim programom. K pôvodnému programu doplníme časť programu na vykreslenie tabuľky. Pokúste sa porozumieť programu, v predstavách si simulovať jeho činnosť ešte pred jeho spustením v počítači.

```
import tkinter
canvas = tkinter.Canvas(width = 300, height = 320)
canvas.pack()

def kresli_stlpec(x, y, dx, dy, znaky):
    fa, fb = 'yellow', 'lightblue'
    for znak in znaky:
        canvas.create_rectangle(x, y, x+dx, y+dy, fill = fa)
        canvas.create_text(x+dx/2, y+dy/2, text = znak)
        y += dy
    fb, fa = fa, fb

velkost = 300
pocet_casti = 2
x = 10
y = 10
dx = 50
spracovat = zoradene[:]

while len(spracovat) > 0:
    kresli_stlpec(x, y, dx, velkost/pocet_casti, spracovat[:pocet_casti])
    spracovat = spracovat[pocet_casti:]
    x += dx
    pocet_casti *= 2
```



#### Otzáky:

27. Koľkokrát sa zavolá funkcia `kresli_stlpec`?
28. Čo bude v zozname `spracovat` po skončení programu?
29. Ako sa zmení nakreslená tabuľka, ak priradenie `pocet_casti = 2` nahradíme priradením `pocet_casti = 3` alebo priradením `pocet_casti = 1`?
30. Ako sa zmení tabuľka, ak do funkcie `kresli_stlpec` vložíme na začiatok priradenie `znaky = znaky[::-1]`?
31. Porovnajte pôvodnú tabuľku z wikipédie s našou tabuľkou. V čom sa líšia? Sú zásadne odlišné, alebo majú niečo spoločné? Diskutujte v skupine.

#### Úlohy:

5

Na internete nájdite niekoľko správ v slovenskom jazyku, uložte ich do textového súboru a vytvorte z neho frekvenčnú tabuľku znakov. Následne vedľa seba vykreslite dve tabuľky, ako by mohlo vyzerať kódovanie Morseovej abecedy z anglických a slovenských textov.

## 6

V súbore s odkazom [http://www.juls.savba.sk/ediela/jc/1999/2/jc1999\\_2.pdf](http://www.juls.savba.sk/ediela/jc/1999/2/jc1999_2.pdf) nájdete na strane 88 túto tabuľku percentuálneho výskytu grafém v slovenskom jazyku:

Tab. 1. Percentuálny výskyt grafém v slovenskom jazyku

Graféma	J. Bosák Výskyt [%]	Graféma	J. Bosák Výskyt [%]
O	9,51	Í	1,20
A	9,06	CH	1,13
E	7,75	Ý	1,10
N	6,34	Č	1,10
I	6,07	Š	0,89
V	5,14	Ú	0,86
R	5,05	Ž	0,85
S	4,74	É	0,81
T	4,54	Ť	0,54
K	3,76	Ľ	0,42
L	3,73	F	0,31
P	3,27	G	0,29
D	3,22	Ó	0,20
M	3,20	Ň	0,16
U	2,65	DZ	0,13
Z	1,99	Á	0,11
Á	1,90	Ó	0,11
J	1,90	Ď	0,11
B	1,69	X	0,06
Y	1,53	Ŕ	0,01
	1,47		0,03

Z vášho textového súboru v slovenskom jazyku vytvorte frekvenčnú tabuľku znakov a prepočítajte ju na percentá. Obe tabuľky navzájom porovnajte.

Váš výsledok porovnajte so zložením dlaždíc a bodovaním dlaždíc v hre Scrabble (<https://sk.wikipedia.org/wiki/Scrabble>).

## 7

Pomocou programu vytvorte histogram zo zostupne zoradenej frekvenčnej tabuľky podľa frekvencie výskytov znakov aj v slovenskom, aj v anglickom jazyku.

Jazykový ústav Ľ. Štúra už niekoľko rokov vytvára Slovenský národný korpus. Slovenský národný korpus (SNK) je elektronická databáza primárne obsahujúca slovenské texty od r. 1955 z rôznych štýlov, žánrov, vecných oblastí, regiónov a pod. v rozsahu poskytnutom autormi a majiteľmi autorských a/alebo distribučných práv na základe licenčnej zmluvy. Texty a slová v korpuse sú obohatené o jazykové informácie a predstavujú referenčný materiálový zdroj poznatkov o slovenčine a jej reálnom používaní, ktoré sa z korpusu získavajú pomocou špecializovaných vyhľadávacích nástrojov. Niektoré výsledky zo spracovania korpusov, ako sú zoznamy slov, spoločné výskyty slov, frekvencia slov atď., sa používajú aj v nelingvistických aplikáciach. Sem patria napr. systémy na spracovanie textov (automatická kontrola pravopisu či gramatiky, strojový preklad textov), systémy na rozpoznávanie reči atď.. Pracovať s korpusom môžete cez web [korpus.sk](http://korpus.sk). Rôzne frekvenčné štatistiky korpusu nájdete na stránke [http://korpus.sk/shk\\_frequencies.html](http://korpus.sk/shk_frequencies.html) [7].

## Frekvencia slov v slovných hodnoteniach

Na istej škole majú študenti na vysvedčení okrem známky aj slovné hodnotenie z každého predmetu a zo správania. Slovné hodnotenia každého predmetu sa skladajú z približne 60 až 120 znakov. Nie sú to súvislé vety, ale slovné spojenia a frázy oddelené čiarkami. Ukážka troch náhodne vybratých slovných hodnotení:

v počítaní je rýchla a presná, má logické myšlenie a s nasadením rieši problémové úlohy  
má výborný logický úsudok, mal by sa viac zaujímať o rozširovanie svojho historického vedomia  
jazyk ovláda na výbornej úrovni, má bohatú slovnú zásobu, gramatiku vie správne aplikovať

V textovom súbore **slovne\_hodnotenia\_all.txt** sú pripravené slovné hodnotenia študentov školy v jednom polroku. Kvôli anonymizácii údajov nie sú pri hodnoteniach uvedené žiadne ďalšie informácie (študent, známka, predmet, ročník a pod.) a poradie hodnotení je náhodne zamiešané. Na každom riadku súboru je práve jedno slovné hodnotenie (jedného študenta v jednom predmete).

Aby sa nám ľahšie vytváral a testoval vytvorený program, máme pripravené aj textové súbory s menším počtom hodnotení. V súbore **slovne\_hodnotenia\_3.txt** sú tri slovné hodnotenia. V súbore **slovne\_hodnotenia\_50.txt** je 50 slovných hodnotení. **Odporučame funkčnosť programu skúsať najprv na týchto kratších súboroch.**

Našou úlohou bude vytvoriť program, ktorý zistí frekvenciu výskytu slov v slovných hodnoteniac a z najfrekventovanejších slov vykreslí stĺpcový histogram. Príčom vykreslený histogram bude interaktívny - pohybom myši sa budú stĺpce po prechode myši farebne zvýrazňovať a nad nimi sa vypíše zväčšené slovo aj s počtom výskytov. Program z najfrekventovanejších slov vynechá predložky a spojky (bude sa o to snažiť, neznamená to, že vynechá všetky spojky, alebo omylem nevynechá aj niečo iné).

### Otázky:

32. Zamyslite sa, ktoré kroky je potrebné v programe urobiť.
33. V čom sa bude tento program lísiť a čo bude mať spoločné s programom na zistenie frekvencie znakov?
34. Ako zabezpečíme interaktivitu histogramu?
35. Ako čo najjednoduchšie vyriešime vyniechanie predložiek a spojok? Ktoré iné slová môže toto riešenie ešte vyniechať a ktoré spojky a predložky, naopak, nevynechá?

Program bude prechádzať vstupný súbor po riadkoch. Spracujeme prečítaný riadok. Odstráime v prečitanom riadku čiarky - čiarku nahradíme ničím pomocou metódy `replace`. Metódou `split` rozdelíme riadok do zoznamu `slova` (podľa medzier), čo odstráni prípadné prebytočné medzery aj na začiatku a konci riadku alebo medzi slovami.

Následne prejdeme zoznamom slov (`slova`) a jednotlivé slová si uložíme do slovníka, kľúčom bude slovo a hodnotou bude frekvencia jeho výskytu. Zistíme, či už je slovo kľúčom slovníka metódou `get` (v prípade, že taký kľúč nie je v slovníku, metóda `get` nám vráti hodnotu 0). Pripočítame počet výskytov a vytvoríme kľúč s novou hodnotou. Potom súbor zatvoríme a v slovníku máme jednotlivé slová.

```
import tkinter
canvas = tkinter.Canvas(width = 1000, height = 800)
canvas.pack()
subor = open('slovne_hodnotenia/slovne_hodnotenia_3.txt', 'r')

pocetnost = {}
for r in subor:
    r = r.replace(',', '')
    slova = r.split()
    for slovo in slova:
        pocetnost[slovo] = pocetnost.get(slovo, 0) + 1

subor.close()
```

Výpis slovníka na príkazovom riadku pre súbor s troma slovnými hodnoteniami:

```
>>> pocetnost
{'pravidlá': 1, 'k': 1, 'prístup': 1, 'do': 1, 'rád': 1, 'pozitívny': 1,
'zapája': 1, 'má': 1, 'koncentráciu': 1, 'zručnosti': 1, 'pracuje': 1,
'priateľsky': 1, 'aktivity': 1, 'sa': 2, 'rešpektuje': 1, 'počítačové': 1,
'snaživo': 1, 'stráca': 1, 'bavia': 1, 'predmetu': 1, 'vybíjanej': 1,
'správa': 1, 'naháňačiek': 1, 'a': 2, 'dobrosrdečne': 1, 'hravý': 1, 'ho': 1,
'je': 1, 'hodinách': 1, 'zvláda': 1, 'občas': 1, 'dohodnuté': 1, 'na': 1,
'základné': 1}
>>>
```

Teraz zoradíme slová podľa frekvencie výskytu. Na ukážku to urobíme iným spôsobom, ako sme to robili pri frekvenčnej tabuľke znakov.

```
zoznam = []
for ntica in pocetnost.items():
    zoznam.append((ntica[1], ntica[0]))

zoznam.sort(reverse = True)
```

Ešte potrebujeme vyhodiť predložky a spojky. Ide nám o približné, nie presné vyhodenie. Kedže nepoznáme slovný druh slov, nevieme presne vyhodiť len predložky a spojky. Napríklad, ak by sme ich mali v nejakom textovom súbore vypísané, mohli by sme to urobiť presnejšie. Predpokladajme, že spojky a predložky sú slová, ktoré majú asi 2 znaky, aj keď tým nepokryjeme spojku alebo. Kedž vyhodíme slová, ktoré majú 2 znaky alebo menej, môžeme vyhodiť aj zámená (napr. sa), slovesá (napr. je) a ďalšie slová. Pre naše riešenie nám to zatiaľ bude postačovať.

```
kopia = []
for prvok in zoznam:
    if len(prvok[1]) >= 3:
        kopia.append(prvok)
```

Teraz už stačí vykresliť histogram zo 40 najfrekventovanejších slov a dorobiť interaktivitu pri pohybe myši. Niektoré parametre pri vykreslovaní nastavíme podľa konkrétnych údajov pre daný súbor.

```
def obdlznik(poradie, vyska, farba):
    canvas.create_rectangle(poradie*20+10, maxy-100, poradie*20+10+10,
                           maxy-100-vyska//2, fill = farba,
                           tags = 'obdlznik'+str(poradie))

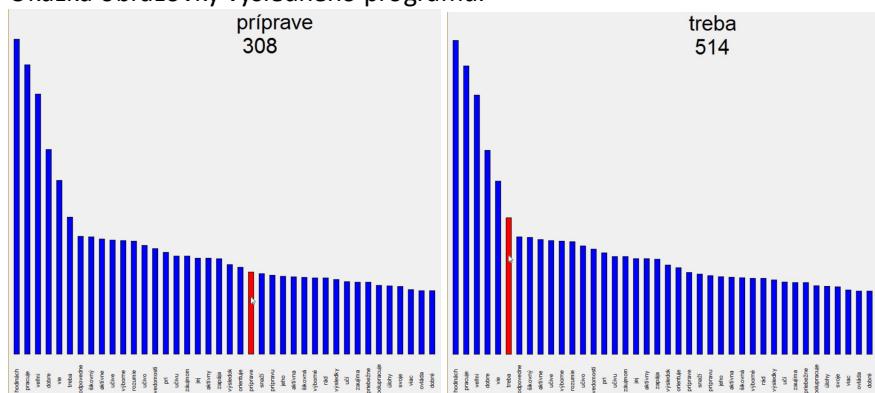
poradie = 0
maxy = 800

for pocet, slovo in kopia[:40]:
    obdlznik(poradie, pocet, 'blue')
    canvas.create_text(poradie*20+15, maxy-50, text = slovo,
                       font = 'Arial 8', angle = 90)
    poradie += 1

def mys(sur):
    global rozsvietene
    if rozsvietene != None:
        canvas.delete('obdlznik'+str(rozsvietene))
        obdlznik(rozsvietene, kopia[rozsvietene][0], 'blue')
    canvas.delete('info')
    if sur.y < 700 and 10 < sur.x < 40*20:
        ktory = (sur.x-10)//20
        canvas.delete('obdlznik'+str(ktory))
        obdlznik(ktory, kopia[ktory][0], 'red')
        canvas.create_text(500, 100, text = kopia[ktory][1]+'\n' +
                           str(kopia[ktory][0]), font = 'Arial 30',
                           tags = 'info')
        rozsvietene = ktory

    rozsvietene = None
    canvas.bind('<Motion>', mys)
```

Ukážka obrazovky výsledného programu:



## Úlohy:

**8** Program sa dá riešiť aj optimálnejšie. Upravte program tak, aby bol efektívnejší (aby napríklad nepoužíval viac pamäte ako potrebuje, alebo aby viackrát zbytočne neprechádzal ten istý zoznam).

**9** Vytvorte zoznam vyhodených slov (každé vyhodené slovo tam bude práve raz), tento zoznam uložte do textového súboru a vizuálne ho skontrolujte. Po vizuálnej kontrole nechajte v súbore len slová, ktoré by sme nemali vynechávať a súbor pomenujte `vynimky.txt`. Následne upravte program tak, aby pri vyhadzovaní slov nevyhodil také slovo, ktoré je v súbore `vynimky.txt`.

**10** Upravte program tak, aby frekvenciu výskytu slov uložil do súboru vo formáte `slovo;počet výskytov;počet výskytov v percentách`. Frekvencia výskytu v percentách bude prepočítaná vzhladom na celkový počet slov.

**11** Na predošlých stranach sme robili šifrovanie náhodnou substitúciou - všetky výskytu daného znaku sme nahradili iným znakom (tiež znakom, ktorý sa nachádza v pôvodnom súbore). Vytvorte teraz šifrovanie slovných hodnotení náhodnou substitúciou, ale nahradzujte rovno celé slová a kľúč v tvare pôvodné slovo; nahradené slovo uložte do súboru.

**12** Mrak slov (word cloud)  
Na vizualizáciu obsahu textu sa používajú mraky slov (word cloud). Z textu sa vytvorí frekvenčná tabuľka slov a najfrekventovanejšie slová sa postupne ukladajú do nejakého obrázku. Frekventovanejšie slová textu sú v mraku napísané väčším písmom a menej frekventované postupne menším písmom. Slová sa v mraku ukladajú do rôznych tvarov a v rôznych farbách. Takéto mraky sa dajú vytvoriť aj pomocou online generátorov, stačí cez Google vyhľadať word cloud a nájdete niekoľko takých generátorov. Tento spôsob vizualizácie textov alebo webu pomôžu čitateľovi získať predstavu, o čom je daný obsah. Na obrázku vidíte mrak slov vytvorený jedným online generátorom z textu 1. kapitoly tejto učebnice.



Ukladanie do presných tvarov a vyplňanie prázdnego miesta je pre nás zatiaľ náročné. No vieme už vykresliť veľmi zjednodušený mrak slov.

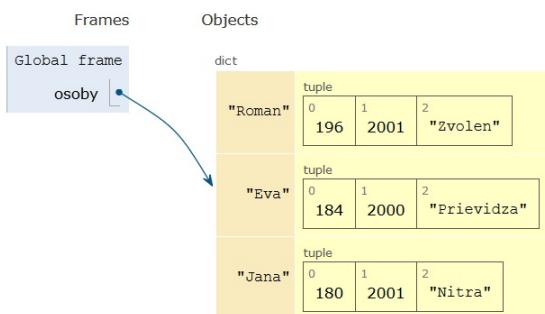
Vytvorte program, ktorý zo slovných hodnotení alebo iného textu vytvorí zjednodušený mrak slov. Na plochu sa náhodne poukladajú slová (náhodne otočené, s náhodnou farbou), pričom bude platiť, že veľkosť fontu slova je závislá od jeho frekvencie výskytu vo vstupnom texte. Pri určovaní veľkosti vymyslite a prakticky otestujte nejakú rozumnú závislosť. Určite nie je najvhodnejšie vykresliť všetky slová, ale len nejakú najfrekventovanejšiu časť.



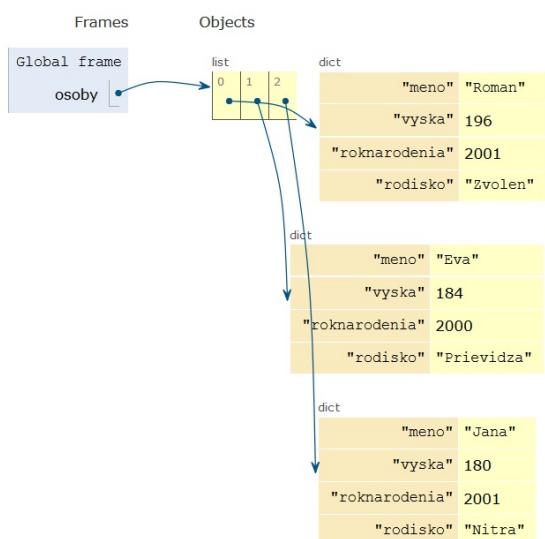
## 8.3 Zoznam asociatívnych polí

V úvode kapitoly sme pomocou slovníka vytvorili zoznam osôb s menom a výškou. Ak by sme k zoznamu osôb chceli doplniť ďalšie údaje, k jednému kľúču (k menu osoby) by mohla byť hodnotou napríklad n-tica alebo zoznam. No museli by sme neustále pri práci s touto štruktúrou myslieť na to, čo ktorý prvak tejto n-tice alebo zoznamu vyjadruje. Napríklad k menu by sme si pamäタali okrem výšky aj rok narodenia a miesto narodenia. Taký slovník by mohol vyzeráť napríklad takto:

```
>>> osoby = {'Roman':(196, 2001, 'Zvolen'), 'Eva':(184, 2000, 'Prievidza'),  
'Jana':(180, 2001,'Nittra')}  
>>> osoby['Roman']  
(196, 2001, 'Zvolen')  
>>> osoby['Roman'][0]  
196  
>>> osoby['Roman'][1]  
2001  
>>> osoby['Roman'][2]  
'Zvolen'  
>>>
```



Ak by sme do štruktúry doplnili ďalšie informácie, napríklad adresu bydliska a pod., už nebude jasné, ktorý údaj čo vyjadruje. Vhodnejšie bude, keď pre každú osobu vytvoríme asociatívne pole (slovník, dictionary) s kľúčmi vyska, roknarodenia, rodisko a z týchto asociatívnych polí (slovníkov) vytvoríme zoznam. Čiže budeme mať zoznam s postupnosťou prvkov o osobách a všetky informácie o jednej osobe budú v jednom slovníku, kde kľúč popisuje danú informáciu. Takto má každá osoba v zozname kartu so svojimi údajmi:



Zoznam osôb môžeme napínať aj postupne:

```
osoby = []  
osoba = {'meno':'Roman', 'vyska':196, 'roknarodenia':2001, 'rodisko':'Zvolen'}
```

```

osoby.append(osoba)
osoba = {'meno':'Eva', 'vyska':184, 'roknarodenia':2000, 'rodisko':'Prievidza'}
osoby.append(osoba)
osoba = {'meno':'Jana', 'vyska':180, 'roknarodenia':2001, 'rodisko':'Nitra'}
osoby.append(osoba)
print(osoby)

```

Vytvoríme textový súbor, kde bude zoznam osôb s týmto údajmi. Jednotlivé údaje sú oddelené bodkočiarkou a načítame ich do zoznamu asociatívnych polí. Ukážka textového súboru:

```

Roman;196;2001;Zvolen
Eva;184;2000;Prievidza
Jana;180;2001;Nitra
Adam;187;2002;Bratislava

```

Program, ktorý prečíta súbor:

```

subor = open('osoby.txt', 'r')
osoby = []
for riadok in subor:
    info = riadok.split(';')
    osoba = {}
    osoba['meno'] = info[0]
    osoba['vyska'] = int(info[1])
    osoba['roknarodenia'] = int(info[2])
    osoba['rodisko'] = info[3].strip()
    osoby.append(osoba)
subor.close()
print(osoby)

```

Z textového súboru sme prečíitali štruktúrované informácie (databázu alebo databázovú tabuľku). Jeden záznam databázy (informácie o jednej osobe) sme vložili ako jeden prvok zoznamu. S takýmito jednoduchými databázami sme zvyknutí pracovať aj v tabuľkovom kalkulátore (napr. Excel, Calc, ...). Priamo v tabuľkovom kalkulátore môžeme databázovú tabuľku uložiť ako **csv** súbor, čo je bežný textový súbor, v ktorom sú stĺpce tabuľky oddelené bodkočiarkou. Súbor **csv** môžeme čítať aj priamo v Pythone ako bežný textový súbor, dokonca nemusíme meniť ani jeho príponu z **csv** na **txt** (**subor = open('osoby.csv', 'r')**).

### Úlohy:

**13** Z načítaného zoznamu osôb zistite a vypíšte mená osôb a počet osôb, ktoré sa narodili v:  
 a) zadanom roku,  
 b) zadanom meste.

**14** Z načítaného zoznamu osôb zistite a vypíšte najvyššiu a najmladšiu osobu.

**15** Nájdite si na internete nejakú tabuľku (napríklad zoznam študentov), skopírujte ho do tabuľkového kalkulátora, uložte ako **csv** súbor a tento súbor načítajte vytvoreným programom v Pythone do zoznamu asociatívnych polí. Následne v tomto zozname nájdite alebo vypočítajte a následne vypíšte napr. priemer, maximum, počet nejakých údajov.

**16** V textových súboroch **eu\_sk.csv**, **eu\_cz.csv** a **eu\_en.csv** sa nachádzajú tabuľky o krajinách (členoch EÚ), ktoré sú vytvorené zo stránok:  
[https://sk.wikipedia.org/wiki/%C4%8Clenovia\\_Eur%C3%B3pskej\\_%C3%BAnie](https://sk.wikipedia.org/wiki/%C4%8Clenovia_Eur%C3%B3pskej_%C3%BAnie)  
[https://cs.wikipedia.org/wiki/%C4%8Clenesk%C3%BD\\_st%C3%A1t\\_Evropsk%C3%A9\\_unie](https://cs.wikipedia.org/wiki/%C4%8Clenesk%C3%BD_st%C3%A1t_Evropsk%C3%A9_unie)  
[https://en.wikipedia.org/wiki/Member\\_state\\_of\\_the\\_European\\_Union](https://en.wikipedia.org/wiki/Member_state_of_the_European_Union).  
 Všetky tri stránky sú pod rovnakým heslom wikipédie, ale v rôznych jazykoch (SK, CZ, EN). Tabuľky uložené v súboroch majú niektoré údaje spoločné, ale mnohé sa môžu lísiť. V prvom riadku súborov sú popísané aj informácie v jednotlivých stĺpcach (csv tabuľky).  
 Ukážky niektorých riadkov jednotlivých súborov (bez prvých riadkov):

```

eu_sk.csv:
Francúzsko;1958;60,65;547 030;29 203;114;74
Nemecko;1958;82,46;357 021;30 150;117;96
Talian sko;1958;58,1;301 230;29 414;114;73

eu_cz.csv:
Francie;1958;65 327 724;643 801;111;106;74;29
Itálie;1958;59 394 207;301 340;97;98;73;29
Německo;1958;81 843 743;357 022;124;125;96;29

eu_en.csv:
France ;Paris ;1958;66352469;632833;2,833,687 ;74
Germany ;Berlin ;1958;81089331;357021;3,874,437 ;96
Italy ;Rome ;1958;61438480;301338;2,147,744 ;73

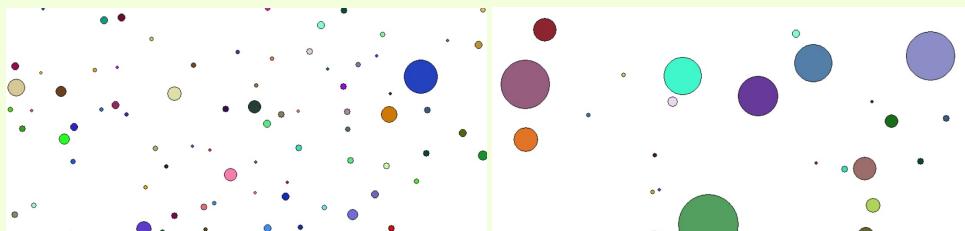
```

Vytvorte program, ktorý prečíta tieto súbory do pamäte a:

- porovná správnosť údajov v jednotlivých tabuľkách, ktoré by mali byť rovnaké (napríklad rozlohu, počet obyvateľov - ak je údaj z rovnakého roku, počet miest v Európskom parlamente). Nesprávne (odlišujúce sa údaje) program vypíše a upozorní na ne,
- vytvorí textový súbor `slovník_krajín.csv`, kde bude zoznam všetkých krajín (ich názvy). V jednom riadku budú názvy krajín vo všetkých troch jazykoch, oddelené bodkočiarkou. Napríklad: France;Francie;Francúzsko,
- vypočíta a vypíše celkovú rozlohu EÚ, celkový počet obyvateľov, celkový počet miest v Európskom parlamente pre všetky jazykové verzie a výsledky vzájomne porovná,
- vypočíta celkovú priemernú hustotu obyvateľstva v EÚ a aj hustotu obyvateľstva v jednotlivých krajinách a vykreslí histogram hustoty obyvateľstva krajín (zoradený od najvyššej hustoty), pričom farebne odliší krajinu s nadpriemernou hustotou obyvateľstva od krajín s podpriemernou hustotou,
- po zadaní roku na vstupe od používateľa vypíše zoznam krajín, ktoré v tom roku boli členmi EÚ, pričom krajinu budú zoradené podľa roku vstupu do EÚ,
- vypočíta a zobrazí nejaké informácie, ktoré si vyberiete.

## 17

Vytvorte program simulácie života špeciálnych baktérií. V programe sa na náhodných miestach vygenerujú špeciálne baktérie (kruhy). Každá baktéria má pri vzniku vygenerovanú náhodnú veľkosť (z malého intervalu) a náhodnú farbu. Program simuluje náhodný pohyb baktérií po ploche. Keď sa stretnú dve baktérie (dotknú sa), väčšia pohltí menšiu a svoju veľkosť zväčší o veľkosť pohltenej baktérie.



S programom experimentujte, skúšajte mu meniť a dopĺňať rôzne parametre a pozorujte simuláciu.

## 18

Upravte program z predchádzajúcej úlohy (simulácie života špeciálnych baktérií) tak, že na začiatku sa vygenerujú baktérie len dvoch farieb F1 a F2. Ak sa stretnú dve baktérie rovnakej farby, väčšia pohltí menšiu. V prípade, že sa stretnú dve baktérie rôznej farby, vytvorí sa nová baktéria (na náhodnom mieste), ktorej veľkosť bude priemerom pôvodných baktérií a jej farba bude náhodná. Program bude priebežne vypisovať aj stav o počte baktérií jednotlivých farieb.

S programom experimentujte a doplňte do neho parameter, ktorým môžeme nastaviť pomer počtu baktérií jednej a druhej farby a následne môžeme pozorovať simuláciu, čo sa bude diať s populáciou.

## 19

V pripravenom textovom súbore `airlines.txt` je zoznam leteckých liniek spoločnosti EuroJet Airlines v tvare `štart;cieľ` na každom riadku. Vytvorte program, ktorý načíta zo súboru zoznam leteckých liniek spoločnosti EuroJet Airlines.

- Vypíšte, koľko liniek z jednotlivých letísk prevádzkuje spoločnosť.
- Vypíšte, z ktorého letiska je prevádzkovaných najviac liniek. Ak je takých viac, vypíšte všetky.

c) Vypíšte tabuľku, z koľkých letísk lieta jedna linka, z koľkých dve atď. až po maximálny počet liniek z jedného letiska.

**20**

Vyhľadajte si na internete medzinárodné telefónne predvoľby pre krajiny Európskej únie. (Napr. Slovensko má +421). V pripravenom textovom súbore `predvolby.txt` sú uložené iba telefónne čísla (jedno telefónne číslo na jednom riadku) na pobočky vašej firmy v rámci EÚ v medzinárodnom tvare (okrem pobočiek v SR).

- a) Vytvorte program, ktorý vypíše zoznam krajín, v ktorých má vaša firma pobočky.
- b) Počítajte s tým, že telefónne čísla môžu byť v tvare +420..., ale aj 00420...
- c) Rátajte aj s číslami, ktoré nemajú medzinárodnú predvoľbu – to sú pobočky na Slovensku.
- d) Vypíšte zoznam, koľko pobočiek sa nachádza v jednotlivých krajinách.

# 9 Vlastnosti útvarov nakreslených v canvase

## 9.1 Zistovanie a zmena nastavení útvarov

Už v prvom diele učebnice sme sa naučili, že útvary nakreslené v canvase (rectangle, line, oval, text) nie sú odtlačené v ploche, ale akosi „žijú“ na ploche. Okrem samotného vytvárania ich zatiaľ vieme posúvať príkazom `canvas.move()` a vymazávať príkazom `canvas.delete()`. Teraz sa naučíme zisťovať a meniť ich vlastnosti. Nakreslíme si pári útvarov, aby sme si na nich v príkazovom riadku vyskúšali nové príkazy.

```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

canvas.create_rectangle(100, 100, 200, 190, fill = 'blue')
canvas.create_rectangle(250, 200, 350, 290, fill = 'red')
canvas.create_oval(120, 220, 190, 290, fill = 'yellow')
canvas.create_oval(270, 120, 340, 190, fill = 'green')
canvas.create_line(100, 350, 150, 300, 200, 350, 100, 350, width = 3)
canvas.create_polygon(250, 350, 300, 300, 350, 350, fill = 'orange')
canvas.create_text(200, 50, text = 'Vlastnosti útvarov', font = 'Arial 20')
```



Príkazom `canvas.itemcget(id_útvaru, 'vlastnosť')` zistíme hodnotu danej vlastnosti útvaru so zadaným identifikátorom. V našom programe má modrý obdĺžnik identifikátor 1, čiže zápisom `canvas.itemcget(1, 'fill')` zistíme farbu jeho výplne. Zápisom `canvas.itemcget(1, 'outline')` zistíme farbu okraja. Pre text môžeme zisťovať aj aktuálny text napríklad `canvas.itemcget(7, 'text')` (identifikátor 7 v našom programe má nadpis).

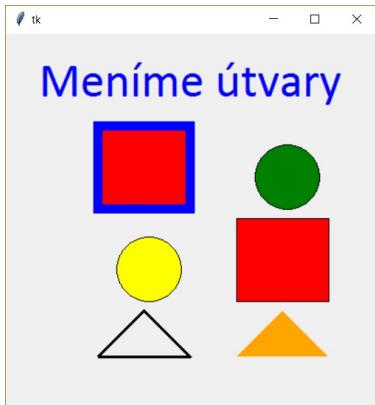
```
>>> canvas.itemcget(1, 'fill')
'blue'
>>> canvas.itemcget(1, 'outline')
'black'
>>> canvas.itemcget(7, 'text')
'Vlastnosti útvarov'
>>> canvas.itemcget(7, 'font')
'Arial 20'
>>>
```

Vlastnosti vykreslených útvarov môžeme aj meniť. Na zmenu vlastnosti nám slúži príkaz `canvas.itemconfig(id_útvaru, vlastnosť = hodnota)`. Napríklad modrý obdĺžnik zmeníme na červený, zmeníme mu hrúbku okraja na 10 pixelov a farbu okraja na modrú. Následne zmeníme text, farbu a font napísaného textu.

```
>>> canvas.itemconfig(1, fill = 'red')
>>> canvas.itemconfig(1, width = 10)
```

```
>>> canvas.itemconfig(1, outline = 'blue')
>>> canvas.itemconfig(7, text = 'Meníme útvary')
>>> canvas.itemconfig(7, fill = 'blue')
>>> canvas.itemconfig(7, font = 'Calibri 40')
>>>
```

Ukážka canvasu po úprave:



#### Otázky:

- Čo sa stane, ak budeme chcieť zistiť alebo nastaviť útvaru vlastnosť, ktorú nemá? Napríklad pre obdĺžnik by sme zisťovali a nastavovali `font` a pre text by sme zisťovali a nastavovali `outline`.
- Na identifikáciu útvarov sme v príkaze `move` a `delete` používali okrem štandardne vytvorených číselných identifikátorov aj značky (`tags`). Budú aj v týchto príkazoch fungovať vytvorené značky? Čo sa stane, ak dva útvary s rôznou vlastnosťou majú rovnakú značku a my budeme zisťovať túto vlastnosť, alebo ju nastavovať?

Príkazom `canvas.coords(id_útvaru)` zistíme súradnice útvaru. Výsledkom je `zoznam`, jeho prvkami sú všetky súradnice. Čiže pre obdĺžnik aj ovál sú to štyri čísla, pre text dve čísla a pre čiaru a polygon je to toľko čísel, koľko súradníc sme zadali pri vytváraní alebo zmene útvaru.

```
>>> canvas.coords(1)
[100.0, 100.0, 200.0, 190.0]
>>> canvas.coords(7)
[200.0, 50.0]
>>> canvas.coords(6)
[250.0, 350.0, 300.0, 300.0, 350.0, 350.0]
>>> canvas.coords(5)
[100.0, 350.0, 150.0, 300.0, 200.0, 350.0, 100.0, 350.0]
>>>
```

Po posunutí útvaru a zistení jeho súradníc vidíme, že sa zmenili.

```
>>> canvas.coords(1)
[100.0, 100.0, 200.0, 190.0]
>>> canvas.move(1, 50, 0)
>>> canvas.coords(1)
[150.0, 100.0, 250.0, 190.0]
>>>
```

Príkaz `coords` môžeme použiť aj na zmenu súradníc útvaru, nové súradnice zadáme ako druhý parameter. Môžu byť zapísané ako n-tica alebo zoznam.

```
>>> canvas.coords(1, [50, 50, 100, 100])
[]
```

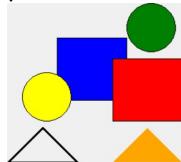
Typ útvaru zistíme príkazom `canvas.type(id_útvaru)`.

```
>>> canvas.type(1)
'rectangle'
>>> canvas.type(3)
'oval'
>>> canvas.type(6)
'polygon'
>>>
```

Zmažme niektorý z útvarov, napríklad `canvas.delete(2)`. Príkaz `canvas.find_all()` vráti n-ticu všetkých identifikátorov existujúcich útvarov v poradí, ako sú na sebe poukladané („na fiktívnych vrstvách“).

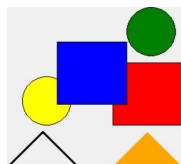
```
>>> canvas.find_all()
(1, 3, 4, 5, 6, 7)
>>>
```

Znovu spustime pôvodný program a vráťme sa k pôvodným útvarom. Posuňme modrý štvorec do stredu, aby sa prekrýval aj s červneným štvorcом, aj so žltým kruhom - `canvas.move(1, 70, 70)`. Vidíme, že sa nachádza pod oboma útvarmi.



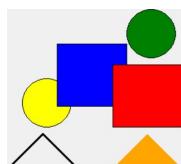
Príkazom `canvas.lift(1)` ho presunieme v poradí útvarov na vrch. Ako parameter použijeme identifikátor útvaru alebo značku (pomocou značky môžeme presunúť aj viac útvarov naraz). Výsledok vidíme aj na obrázku a aj v poradí n-tice útvarov:

```
>>> canvas.lift(1)
>>> canvas.find_all()
(2, 3, 4, 5, 6, 7, 8, 1)
```



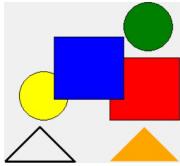
Teraz presunieme hore útvar s identifikátorom 2 - červený obdĺžnik.

```
>>> canvas.lift(2)
>>> canvas.find_all()
(3, 4, 5, 6, 7, 8, 1, 2)
```



Príkazom `canvas.lower(2)` presunieme útvar s identifikátorom 2 na spodok. Aj tu je parametrom identifikátor útvaru alebo značka (pomocou značky môžeme presunúť aj viac útvarov naraz).

```
>>> canvas.lower(2)
>>> canvas.find_all()
(2, 3, 4, 5, 6, 7, 8, 1)
```



Vytvoríme program, ktorý po stlačení tlačidla cyklom prejde cez všetky útvary. Ovály alebo obdĺžníky zmenší na polovicu.

```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

canvas.create_rectangle(100, 100, 200, 190, fill = 'blue', tags = 'obd')
canvas.create_rectangle(250, 200, 350, 290, fill = 'red', tags = 'obd')
canvas.create_oval(120, 220, 190, 290, fill = 'yellow')
canvas.create_oval(270, 120, 340, 190, fill = 'green')
canvas.create_line(100, 350, 150, 300, 200, 350, 100, 350, width = 3)
canvas.create_polygon(250, 350, 300, 300, 350, 350, 300, fill = 'orange')
canvas.create_text(200, 50, text = 'Vlastnosti útvarov', font = 'Arial 20')

def zmensi():
    utvary = canvas.find_all()
    for utvar in utvary:
        if canvas.type(utvar) == 'rectangle' or canvas.type(utvar) == 'oval':
            suradnice = canvas.coords(utvar)
            x1, y1, x2, y2 = suradnice
            stred_x = (x2 + x1)/2
            nsirka = (x2 - x1)/4
            stred_y = (y2 + y1)/2
            nvyska = (y2 - y1)/4
            nove_suradnice = [stred_x - nsirka, stred_y - nvyska,
                               stred_x + nsirka, stred_y + nvyska]
            canvas.coords(utvar, nove_suradnice)

button1 = tkinter.Button(text = 'zmenši obdĺžniky a štvorce', command = zmensi)
button1.pack()
```



### Úlohy:

**1**

Upravte predchádzajúci program tak, aby sa po stlačení tlačidla napísané texty napísali od zadu (program prehodí poradie znakov).

Aby sme v programe mohli pracovať s útvarami pomocou myši, budeme používať funkciu `canvas.find_overlapping(x1, y1, x2, y2)`. Táto funkcia vráti n-ticu všetkých identifikátorov útvarov, ktoré sa nachádzajú v zadanom obdĺžniku (stačí ich časť). Okolo súradníc myši môžeme nakresliť malý obdĺžnik a pomocou tejto funkcie zistíme, ktoré útvary do neho zasahujú. Do programu doplníme reakciu na

udalosť '**<Motion>**'. Výsledok budeme vypisovať do textu, ktorého obsah budeme meniť príkazom **itemconfig**.

```
def mys(event):
    x1, y1, x2, y2 = event.x-5, event.y-5, event.x+5, event.y+5
    podmysou = canvas.find_overlapping(x1, y1, x2, y2)
    info = 'Ste nad útvarmi číslo: '+str(podmysou)
    canvas.itemconfig('oznam', text = info)

canvas.create_text(100, 10, text = '', tags = 'oznam')
canvas.bind('<Motion>', mys)
```



Vytvoríme program, ktorým budeme kresliť čiary. Prvým kliknutím nakreslíme začiatok čiary, ďalším kliknutím pridáme k čiare ďalší bod. Kreslenie čiary ukončíme dvojklikom. Program nám umožní kresliť viacero čiar.

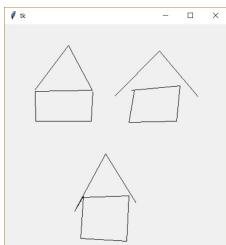
```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

zaciatok = True
pocet = 0
ciara = []

def bod(event):
    global ciara, pocet, zaciatok
    if zaciatok:
        ciara = [event.x, event.y, event.x, event.y]
        pocet +=1
        canvas.create_line(ciara, tags = 'ciara'+str(pocet))
        zaciatok = False
    else:
        ciara.append(event.x)
        ciara.append(event.y)
        canvas.coords('ciara'+str(pocet), ciara)

def koniec(event):
    global ciara, zaciatok
    if not zaciatok:
        ciara.append(event.x)
        ciara.append(event.y)
        canvas.coords('ciara'+str(pocet), ciara)
    zaciatok = True
```

```
canvas.bind('<Button-1>', bod)
canvas.bind('<Double-Button-1>', koniec)
```



### Otázky:

3. Na čo nám slúži v programe premenná `zaciatok` a prečo ju potrebujeme?
4. Prečo je v programe prvý bod v čiare zadaný dvakrát? Ako by sme mohli program upraviť, aby prvý bod čiary nebol v zozname (liste) dvakrát a aby fungoval korektnie?
5. Prečo je v programe použité značkovanie útvarov (`tags`)? Je to potrebné? Dá sa program vytvoriť aj bez `tags`? Ak áno, aké by bolo iné riešenie?

Ešte môžeme program vylepšiť. Pri pohybe myši budeme stále prekreslovať (naťahovať) čiaru do miesta, kde sa chystáme kliknúť (kde je práve myš). Tento bod si však nesmieme zapamätať do zoznamu `ciara`, lebo pri ďalšom potiahnutí myši ho už nepotrebuje. Doplníme do programu reakciu na udalosť '`<Motion>`'. Na túto udalosť budeme reagovať funkciou `natahovanie`.

```
def natahovanie(event):
    if not zaciatok:
        docasna_ciara = ciara[:]
        docasna_ciara.append(event.x)
        docasna_ciara.append(event.y)
        canvas.coords('ciara'+str(pocet), docasna_ciara)

canvas.bind('<Motion>', natahovanie)
```

### Úlohy:

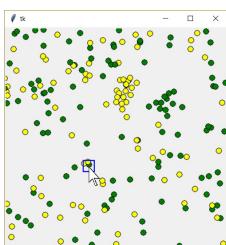
2

Doplňte do programu tlačidlo `save`. Po stlačení tlačidla sa do textového súboru uložia súradnice všetkých nakreslených čiar. Na každom riadku súboru budú súradnice práve jednej čiary.

3

Vytvorte program, ktorý prečíta textový súbor z predchádzajúcej úlohy a vykreslí všetky čiary zapísané v súbore.

Vytvorme pre mladšieho súrodenca interaktívnu aktivitu k rozprávke o Popoluške. V programe sa na začiatku vygeneruje a nakreslí niekoľko zrniek kukurice a hrachu (žlté a zelené krúžky). Zrnká budú náhodne rozhádzané po ploche. Úlohou používateľa bude rozdeliť od seba zrnká kukurice a hrachu. Používateľ prenáša zrnká pomocou myši. Aby to nebolo úplne jednoduché, myš (hrubý prst) zoberie naraz všetky zrnká, ktoré sú v štvorci 20x20 pixelov. Veľkosť jedného zrnka je 10x10 pixelov.



## Otázky:

6. Ako by ste riešili tento program?
7. Na ktoré udalosti by sme mali v programe reagovať? Ako zrealizujeme dragovanie (ťahanie) myšou?
8. Čo si potrebujeme v programe pamätať? Kedže využijeme príkazy na zisťovanie nakreslených útvarov z tejto kapitoly, ktoré informácie si nemusíme pamätať a zistíme ich z vlastností útvarov? Ak by sme nepoznali príkazy z tejto kapitoly, ako inak by sme mohli úlohu riešiť s tým, čo vieme doposiaľ?

Najprv si vykreslíme zrnká na náhodnom mieste, buď žlté alebo zelené.

```
import tkinter, math, random
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

def zrnka():
    for i in range(200):
        sx, sy = random.randrange(400), random.randrange(400)
        f = random.choice(('green', 'yellow'))
        canvas.create_oval(sx-5, sy-5, sx+5, sy+5, fill = f)

zrnka()
```

Pri pohybe myši budeme premiestňovať štvorčekový kurzor o veľkosti 20 x 20 pixelov, ktorý nám označuje, ktoré zrnká by sme zobrali, keď ich začneme myšou ťahať. Obdĺžnik si nakreslíme na začiatku v bode 0, 0 a bude mať nulovú veľkosť. Pri pohybe myši mu už len zmeníme vlastnosti a nemusíme ho zakaždým mazať a vytvárať.

```
def mys(event):
    x1, y1, x2, y2 = event.x-10, event.y-10, event.x+10, event.y+10
    obdlznik = [x1, y1, x2, y2]
    prekryva = canvas.find_overlapping(x1, y1, x2, y2)
    canvas.coords('obdlznik', obdlznik)

canvas.create_rectangle(0,0,0,0, width = 2, outline = 'blue',tags = 'obdlznik')
canvas.bind('<Motion>', mys)
```

Ťahanie (dragovanie) bude mať tri fázy. Prvá fáza je začiatok (pri kliknutí). Vtedy si zapamätáme do globálnej premennej **stara\_pozicia** terajšiu pozíciu myši, aby sme pri ďalšom pohnutí myšou zistili, o koľko sme sa posunuli, a teda o koľko potrebujeme posunúť ťahané útvary. A ešte na začiatku ťahania zistíme, ktoré útvary sú v okolí 20 x 20 px pod kurzorom myši. Tieto útvary si zapamätáme do globálneho zoznamu **taham**, aby sme pri ťahaní posúvali už len tieto útvary.

Čiže po kliknutí ľavého tlačidla myši sme si zapamätali, kde sa nachádza myš, ktoré útvary prekrýva, a teda ich ideme ťahať. Aby sme vizuálne videli, že sme v stave ťahania, obdĺžnik (kurzor) prefarbíme na inú farbu.

Druhou fázou je ťahanie (udalosť '**<B1-Motion>**'). Tam zistujeme, o koľko sa myš posunula voči starej pozícii. O toľko posunieme všetky ťahané útvary a následne si zapamätáme do starej pozície myši terajšiu pozíciu (aby sme vedeli v nasledujúcim kroku, o koľko sme sa posunuli v tomto jednom kroku).

Poslednou fázou je ukončenie ťahania, čiže reakcia na pustenie ľavého tlačidla '**<ButtonRelease-1>**'. Tu len vyprázdnime zoznam **taham**, pretože už nič neťaháme. A zmeníme farbu kurzora (obdĺžnika) na pôvodnú.

Celý výsledný program (bez testu oddelenia kukurice od hrachu):

```
import tkinter, math, random
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

taham = []
stara_pozicia = []

def zrnka():
    for i in range(200):
        sx, sy = random.randrange(400), random.randrange(400)
```

```

f = random.choice(('green','yellow'))
canvas.create_oval(sx-5, sy-5, sx+5, sy+5, fill = f)

def mys(event):
    x1, y1, x2, y2 = event.x-10, event.y-10, event.x+10, event.y+10
    obdlznik = [x1, y1, x2, y2]
    prekryva = canvas.find_overlapping(x1, y1, x2, y2)
    canvas.coords('obdlznik', obdlznik)

def oznamenie(event):
    global taham, stara_pozicia
    x1, y1, x2, y2 = event.x-10, event.y-10, event.x+10, event.y+10
    stara_pozicia = [event.x, event.y]
    obdlznik = [x1, y1, x2, y2]
    taham = canvas.find_overlapping(x1, y1, x2, y2)
    canvas.coords('obdlznik', obdlznik)
    canvas.itemconfig('obdlznik', outline = 'red')

def tahanie(event):
    global stara_pozicia
    posunx = event.x - stara_pozicia[0]
    posuny = event.y - stara_pozicia[1]
    stara_pozicia = [event.x, event.y]
    for utvar in taham:
        canvas.move(utvar, posunx, posuny)

def ukoncenie(event):
    global taham
    taham = []
    canvas.itemconfig('obdlznik', outline = 'blue')

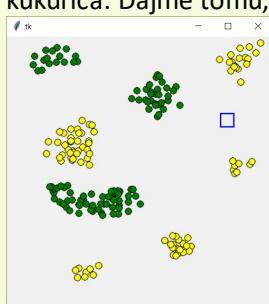
zrnka()
canvas.create_rectangle(0,0,0,0, width = 2, outline = 'blue', tags = 'obdlznik')
canvas.bind('<Motion>', mys)
canvas.bind('<Button-1>', oznamenie)
canvas.bind('<B1-Motion>', tahanie)
canvas.bind('<ButtonRelease-1>', ukoncenie)

```

### Úlohy:

**4**

Doplňte do programu test, ktorý zistí, či sú oddelené zrnká kukurice od zrniek hrachu. Nemusí to byť rozdelenné na dvoch kôpkach, ale dôležité je, že v blízkom okolí žiadneho zrnka hrachu sa nenachádza kukurica. Dajme tomu, že v okolí 50 px zrnka hrachu nie je kukurica.



**5**

Doplňte do programu pomôcku pre používateľa, ktorá mu pomôže presúvať len zrnká kukurice alebo hrachu. Po stlačení klávesu **k** (kukurica) sa budú po kliknutí tăhať len zrnká kukurice, po stlačení klávesu **h** (hrach) sa budú po kliknutí tăhať len zrnká hrachu. Stlačenie medzery vráti program do bežného režimu, v ktorom tăháme aj zrnká hrachu, aj kukurice.

**6**

Upravte program tak, aby sa po stlačení klávesu **enter** zrnká samy oddelili. Animáciou sa budú postupne pohybovať. Do hornej polovice obrazovky prejde hrach a do dolnej polovice prejdú zrnká

kukurice.



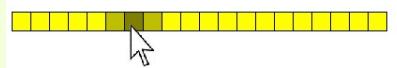
7

Vytvorte program, ktorý vykreslí vedľa seba 20 štvorčekov náhodne vybraných farieb. Pri prechode myši ponad ne sa budú štvorčeky zväčšovať podľa ukážky. Túto úlohu je aj v piatej kapitole, ale teraz ju riešte tak, aby ste štvorčeky nepremazávali, ale aby program menil iba vlastnosti nakreslených útvarov.



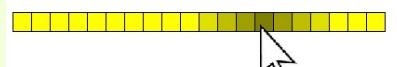
8

Vytvorte program, ktorý vykreslí vedľa seba 20 štvorčekov rovnakej farby. Pri prechode myši ponad ne sa bude farba štvorčekov meniť podľa ukážky – farba štvorčeku, nad ktorým je myš, stmaavne a farba najbližších štvorčekov vľavo a vpravo stmaavne tiež, ale trochu menej (štvorčeky nevymazávajte).



9

Upravte program z predchádzajúcej úlohy tak, že bude univerzálnejší. V programe bude môcť používateľ nastaviť konštantu – koľko susedných štvorčekov sa má postupne stmaavovať.



10

Vytvorte animáciu – najjasnejší štvorček sa bude „pohybovať“ („rozsvecovať“) z jednej strany na druhú, pričom od krajov sa vždy odrazí na opačný smer a za sebou bude zanechávať „chwost“ z postupnosti stále tmavších štvorčekov.



## 9.2 Značky útvarov a ich využitie na pamäťanie si informácie

Z nakreslených útvarov si vieme zistiť rôzne informácie. No do útvaru si môžeme aj zapamätať nejakú informáciu pomocou `tags`, napríklad si informáciu vhodne zakódujeme do značky, ktorou útvar pomenujeme. Jeden útvar môže mať aj viacero značiek, vtedy ich zapíšeme ako n-ticu, napríklad:

```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 400)
canvas.pack()

canvas.create_rectangle(100, 100, 180, 180, fill = 'blue',
                       tags = ('obdlznik', 'farbal'))
canvas.create_rectangle(200, 100, 280, 180, fill = 'red',
                       tags = ('obdlznik', 'farba2'))
canvas.create_oval(100, 200, 180, 280, fill = 'blue', tags = ('kruh', 'farbal'))
canvas.create_oval(200, 200, 280, 280, fill = 'red', tags = ('kruh', 'farba2'))
```

Príkazom `canvas.itemconfig('farbal', fill = 'green')` zmeníme všetky útvary, ktoré majú značku `farbal` na zelené. A napríklad príkazom `canvas.move('obdlznik', -50, 0)` posunieme všetky útvary, ktoré majú značku `'obdlznik'`.

Značky útvaru môžeme zistiť pomocou príkazu `canvas.itemcget(ID_útvaru, 'tags')` alebo príkazom `canvas.gettags(ID_útvaru)`. Vhodnejšie je používať príkaz `gettags()`, pretože ak má útvar viac značiek, vráti nám ich ako n-ticu, kdežto príkaz `itemcget()` ako textový reťazec.

```
>>> canvas.gettags(1)
('obdlznik', 'farbal')
>>> canvas.itemcget(1, 'tags')
'obdlznik farbal'
>>>
```

Ak sa nad útvarom nachádza kurzor myši, bude mať aj značku `'current'`. To môžeme využiť na hľadanie útvarov, nad ktorými je myš, alebo na ktoré sme klikli. Doposiaľ sme to zisťovali cez `find_overlapping()`, ale môžeme využiť funkciu `canvas.find_withtag('current')`. Najprv kurzor myši nie je nad žiadnym útvarom, následne ho presunieme nad útvar s identifikátorom 2 a znova zavoláme funkciu:

```
>>> canvas.find_withtag('current')
()
>>> canvas.find_withtag('current')
(2,)
```

Značky sa dajú meniť tiež príkazom `canvas.itemconfig()`, ale aby nám správne fungovala značka `'current'` - pri klikaní myšou a pohybe myši, odporúčame na pridanie značky používať príkaz `canvas.addtag('nová_značka', 'withtag', ID_útvaru)` a na odobranie značky príkaz `canvas.dtag(ID_útvaru, 'značka')`.

```
>>> canvas.gettags(1)
('obdlznik', 'farbal', 'current')
>>> canvas.addtag('utvar1', 'withtag', 1)
>>> canvas.gettags(1)
('obdlznik', 'farbal', 'current', 'utvar1')
>>> canvas.dtag(1, 'farbal')
>>> canvas.gettags(1)
('obdlznik', 'utvar1', 'current')
>>>
```

Zhrňme si nové príkazy:

`canvas.gettags(ID_útvaru)` - vráti n-ticu značiek zadaného útvaru,

`canvas.find_withtag('hľadaná značka')` - vráti n-ticu ID útvarov so zadanou značkou,

`canvas.addtag('nová značka', 'withtag', ID_útvaru)` - k zadanému útvaru pridá novú značku,

`canvas.dtag(ID_útvaru, 'stará značka')` - zo zadaného útvaru odstráni zadanú značku.

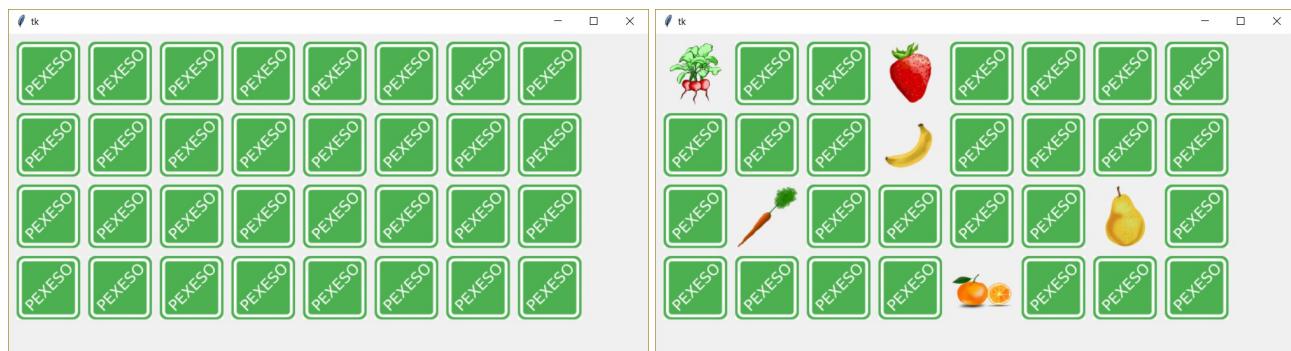
Pomocou značiek útvarov a pamäťania si informácií v nich si naprogramujeme pexeso s obrázkami ovocia a zeleniny. V súboroch `oz_1.png`, `oz_2.png` až `oz_16.png` sú obrázky ovocia a zeleniny :



V súbore `pexeso.png` je zadná strana kartičky pexesa.



Náš program nakreslí 32 kartičiek (každý obrázok dvakrát) v obdĺžniku 8x4 kartičiek. Na začiatku budú všetky kartičky otočené rubom. Najprv zabezpečíme len otáčanie kartičiek, na ktoré klikneme, až neskôr budeme riešiť dodržanie hracích pravidiel pexesa.



### Otázky:

9. Čo všetko musíme v programe riešiť?
10. Ktoré údaje si budeme musieť v programe pamätať? A ktoré z nich si stačí pamätať pomocou vlastností a značiek priamo v útvaroch?
11. V akých štruktúrach bude najvhodnejšie si pamätať potrebné údaje?
12. Ako vytvoríme dvojicu kartičiek a zamiešame ich poradie na začiatku?
13. Akým spôsobom zabezpečíme, aby sme kliknutím myši kartičku otočili?

Obrázky ovocia a zeleniny si zapamätáme v zozname `obrazky`. Zoznam bude mať prvky s indexom `0` až `15`. Obrázok rubu si zapamätáme v premennej `pexeso`. Pomocou vnorených for cyklov nakreslíme všetkých 32 kartičiek, najprv s obrázkom rubu. Ale už pri prvom vykreslovaní kartičky si do značky (tagu) zapamätáme číslo obrázku (zodpovedajúci index v zozname `obrazky`), ktorý tam bude umiestnený. Keďže značka nemôže mať v názve iba číslo, na jej začiatok pridáme text. V ňom si budeme pamätať, ako je kartička práve otočená. Čiže značka bude v tvare `'rub_'` a číslo obrázku alebo `'lice_'` a číslo obrázku. Napríklad obe kartičky s obrázkom číslo 1 budú mať najprv tag `'rub_1'` a po otočení kartičky zmeníme len tej jednej kartičke značku na `'lice_1'`. Ešte pred vykreslením musíme kartičky zamiešať, keďže čísla umiestnených obrázkov na kartičkách

si pamätáme hned pri vytvorení. Zamiešame ich tak, že si vytvoríme zoznam **oznacenia** so všetkými môžnými značkami (pre čísla 0 až 15), ten zduplicujeme (aby tam bola značka aj pre druhú kartičku) a funkciou **random.shuffle()** ho zamiešame. Teraz pri vytváraní kartičiek budeme len vyberať značky v poradí (už zamiešanom) zo zoznamu **oznacenia**.

```
import tkinter, random
canvas = tkinter.Canvas(width = 800, height = 400)
canvas.pack()

def nacitaj(nazov, typ, pocet, obrazky):
    for i in range(1, pocet+1):
        obrazok = tkinter.PhotoImage(file = nazov + str(i) + '.'+typ)
        obrazky.append(obrazok)

def kresli_rub(x, y, oznacenia):
    poradie = 0
    for j in range (4):
        for i in range(8):
            canvas.create_image(x + i*90, y + j*90, image = pexeso,
                                tags = oznacenia[poradie])
        poradie += 1

obrazky = []
nacitaj('obrazky/oz_', 'png', 16, obrazky)

pexeso = tkinter.PhotoImage(file = 'obrazky/pexeso.png')

oznacenia = []
for i in range(16):
    oznacenia.append('rub_'+str(i))
oznacenia = oznacenia + oznacenia
random.shuffle(oznacenia)

kresli_rub(50, 50, oznacenia)
```

Kartičky sú vykreslené, môžeme doplniť reakciu na kliknutie. Tu zistíme, na ktorú kartičku sme klikli **zakliknute = canvas.find\_withtag('current')**. Kedže kartičky sa neprekryvajú, v premennej **zakliknute** bude buď prázdna alebo jednoprvková n-tica. Pokračovať ďalej budeme iba v prípade, že tam je prvok. Ten vyberieme do premennej **zakliknuty**. V nej je teraz **ID** zakliknutého útvaru a my potrebujeme zistiť tagy tohto útvaru **tagy = canvas.gettags(zakliknuty)**. Okrem nami prideleného tagu bude mať útvar aj tag **current**, pretože sme naň práve klikli. A nevieme, v akom poradí je nami pridelený tag a tag **current**, preto sa na to **if**-om spýtame a do premennej **stara\_znacka** si vytiahneme nami pridelený tag. Z neho vieme zistiť, ako je kartička práve otočená a ktorý obrázok (číslo) má byť na nej umiestnený. Takže napríklad v premennej **stara\_znacka** pre kartičku s obrázkom 1 môže byť tag '**rub\_1**' alebo '**lice\_1**'. Metódou **split** si z názvu odčleníme číslo a text **info = stata\_znacka.split('\_')**. Ak je textom **rub**, nová značka bude mať text **lice** a naopak, ak je textom **lice**, nová značka bude mať text **rub** (lebo kartičku práve otáčame). Takže si vytvoríme novú značku, pridáme ju zakliknutému útvaru a vymažeme mu starú značku. Následne kartičke zmeníme obrázok cez príkaz **itemconfig**, pričom nový obrázok bude z premennej **pexeso** alebo zo zoznamu **obrazky** s indexom čísla kartičky (podľa toho, ako má byť kartička otočená).

```
def klik(event):
    zakliknute = canvas.find_withtag('current')
    if len(zakliknute) > 0:
        zakliknuty = zakliknute[0]
        tagy = canvas.gettags(zakliknuty)
        if tagy[0] != 'current':
            stata_znacka = tagy[0]
        else:
            stata_znacka = tagy[1]
        info = stata_znacka.split('_')
        if info[0] == 'rub':
```

```

        nova_znacka = 'lice_'+info[1]
        cislo_obrazku = int(info[1])
        canvas.addtag(nova_znacka, 'withtag', zakliknuty)
        canvas.dtag(zakliknuty, stara_znacka)
        canvas.itemconfig(zakliknuty, image = obrazky[cislo_obrazku])
    if info[0] == 'lice':
        nova_znacka = 'rub_'+info[1]
        canvas.addtag(nova_znacka, 'withtag', zakliknuty)
        canvas.dtag(zakliknuty, stara_znacka)
        canvas.itemconfig(zakliknuty, image = pexeso)

canvas.bind('<Button-1>', klik)

```

### Otázky:

14. Ako by ste upravili program, aby fungoval podľa pravidiel hry pexeso?

Program upravíme, aby sa viac podobal na hru pexeso. To znamená, že hráč môže lícom otočiť dve kartičky. Ak sú kartičky rovnaké, vyradia sa z hry a hráč získava bod a má ďalší ťah. Ak sú kartičky rôzne, otočia sa na rub a v hre pokračuje druhý hráč. Do programu doplníme globálnu premennú `pocet_otocenych`, v ktorej si budeme pamätať počet kartičiek otočených lícom (na začiatku ich bude 0). V zozname `otocene` si budeme pamätať ich značky (na začiatku bude prázdny).

Pri kliknutí zrušíme možnosť klikania na kartičky otočené lícom (dáme preč celý `if info[0] == 'lice'`).

V kliknutí na rub doplníme podmienku, že počet otočených kartičiek je menší ako 2

`if info[0] == 'rub' and pocet_otocenych < 2:`. Následne po kliknutí na rub zvýšime počet otočených kartičiek `pocet_otocenych += 1`. A do zoznamu otočených kartičiek pridáme novú značku otočenej kartičky `otocene.append(nova_znacka)`.

Na konci tohto `if`-u zistíme, či už sú otočené dve kartičky. Ak áno, zistíme, či sú rovnaké, ak sú, vyradíme ich z hry a vynulujeme počet otočených kartičiek a vyprázdnime zoznam `otocene`. Ak sú kartičky rôzne, otočíme ich naspäť a tiež vynulujeme počet otočených kartičiek a vyprázdnime zoznam `otocene`. Aby si to hráči stihli aj všimnúť, hned' za podmienkou, či sú otočené dve kartičky, program na pol sekundy pozdržíme `canvas.update() canvas.after(500)`.

Nezabudnime ešte doplniť `global pocet_otocenych, otocene` do funkcie `klik()`. Zatiaľ program nestrieda hráčov a nepočítava body. A takto vyzerá celý program aj s doplnenými funkiami `rovnak()`, `vyrad_z_hry()`, a `otoc()`.



```

import tkinter, random
canvas = tkinter.Canvas(width = 800, height = 400)
canvas.pack()

def nacitaj(nazov, typ, pocet, obrazky):
    for i in range(1, pocet+1):
        obrazok = tkinter.PhotoImage(file = nazov + str(i) + '.'+typ)
        obrazky.append(obrazok)

def kresli_rub(x, y, oznamenia):

```

```

poradie = 0
for j in range (4):
    for i in range(8):
        canvas.create_image(x + i*90, y + j*90, image = pexeso,
                            tags = oznacenia[poradie])
    poradie += 1

obrazky = []
nacitaj('obrazky/oz_', 'png', 16, obrazky)

pexeso = tkinter.PhotoImage(file = 'obrazky/pexeso.png')

oznacenia = []
for i in range(16):
    oznacenia.append('rub_'+str(i))
oznacenia = oznacenia + oznacenia
random.shuffle(oznacenia)

kresli_rub(50, 50, oznacenia)
pocet_otocenych = 0
otocene = []

def rovnaKE(karticky):
    if karticky[0] == karticky[1]:
        return True
    return False

def vyrad_z_hry(karticky):
    canvas.delete(karticky[0])

def otoc(karticky):
    for karticka in karticky:
        info = karticka.split('_')
        nova_znacka = 'rub_'+info[1]
        ID = canvas.find_withtag(karticka)
        canvas.addtag(nova_znacka, 'withtag', ID)
        canvas.dtag(ID, karticka)
        canvas.itemconfig(ID, image = pexeso)

def klik(event):
    global pocet_otocenych, otocene
    zakliknute = canvas.find_withtag('current')
    if len(zakliknute) > 0:
        zakliknuty = zakliknute[0]
        tagy = canvas.gettags(zakliknuty)
        if tagy[0] != 'current':
            stara_znacka = tagy[0]
        else:
            stara_znacka = tagy[1]
        info = stara_znacka.split('_')
        if info[0] == 'rub' and pocet_otocenych < 2:
            pocet_otocenych += 1
            nova_znacka = 'lice_'+info[1]
            otocene.append(nova_znacka)
            cislo_obrazku = int(info[1])
            canvas.addtag(nova_znacka, 'withtag', zakliknuty)
            canvas.dtag(zakliknuty, stara_znacka)
            canvas.itemconfig(zakliknuty, image = obrazky[cislo_obrazku])
            if pocet_otocenych == 2:
                canvas.update()
                canvas.after(500)
                if rovnaKE(otocene):
                    vyrad_z_hry(otocene)

```

```

        else:
            otoc(otocene)
        otocene = []
        pocet_otocenych = 0

canvas.bind('<Button-1>', klik)

```

### Úlohy:

- 11** Vylepšite hru pexeso. Doplňte do nej počítanie a výpis počtu bodov, striedanie hráčov a záverečné vyhodnotenie.
- 12** Upravte hru pexeso tak, že si môžeme vybrať, s akým počtom dvojíc obrázkov hráme, napríklad od 8 do 16 dvojíc.
- 13** Upravte program pexeso tak, že pravým tlačidlom môžeme kartičky chytiť a presúvať na iné miesto.
- 14** Upravte program pexeso tak, že počiatočný stav hry a jej priebeh sa uloží do textového súboru. Iným programom môžeme takto uložený priebeh hry prehrať. Tiež ho vytvorte.

Nakreslené útvary môžeme aj skrývať alebo znova ukázať vlastnosťou **state**. Aj týmto spôsobom by sme mohli riešiť hru pexeso. Napríklad tak, že rovno na začiatku pre každú kartičku vytvoríme dva útvary - jeden s rubom a druhý s lícom. Po kliknutí na kartičku ju schováme a ukážeme tú druhú s opačnou stranou, ktorá je na rovnakom mieste. Mohli by sme použiť napríklad tieto príkazy:

```

>>> canvas.itemconfig(1, state = 'hidden')
>>> canvas.itemconfig(1, state = 'normal')
>>> canvas.itemconfig('rub_1', state = 'hidden')
>>> canvas.itemconfig('lice_1', state = 'normal')
>>>

```

Treba dať pozor, ak meníme útvary podľa značky a viacero útvarov má rovnakú značku. Vtedy zmeníme všetky tieto útvary.

# 10 Upravujeme vzhľad aplikácií

Táto kapitola neobsahuje úlohy a otázky, pretože je skôr doplňujúcou kapitolou. Informácie v nej obsiahnuté vám pomôžu upraviť vzhľad aplikácií, ktoré ste doposiaľ vytvárali. Naučíme sa vytvárať ovládacie prvky, ktoré použijeme pri vytváraní väčších komplexnejších programov, napríklad pri tvorbe seminárnej práce.

V našich aplikáciách sme doposiaľ bežne používali `canvas`, `button`, `entry`. Nazývali sme ich grafickými súčiastkami. Sú súčasťou grafickej knižnice `tkinter` a v Pythone sa nazývajú `widget` (ovládacie prvky). Predstavíme si niektoré ďalšie ovládacie prvky, aby sme mohli vytvárať komplexnejšie programy, ktoré sú dobre ovládateľné používateľom.

## Metódy zobrazenia widgetov a ich umiestnenie

Ked' sme vytvárali nejaký `widget`, zobrazovali sme ho metódou `pack()`. Ešte si na porozumenie zobrazovania prvkov ukážeme na príklade, čo sa stane, keď pre rôzne prvky zavoláme metódu `pack()` v inom poradí. A tiež uvidíme, že aplikácia môže mať viaceré grafické plátna.

Prvý program poukladá do okna aplikácie v poradí `canvas1`, `canvas2`, `label1`, `entry1` a `button1`. Ak poradie volania metód `pack()` zameníme, ako vidíme v ukážke druhého programu, v hornej časti okna budú najprv `label1`, `entry1` a `button1` a až potom bude `canvas2` a `canvas1`.

```
import tkinter, random

def nakresli():
    x = random.randrange(400)
    y = random.randrange(200)
    canvas1.create_text(x, y, text = entry1.get())
    canvas2.create_text(x, y, text = entry1.get()[:-1])

canvas1 = tkinter.Canvas(width = 400, height = 200, bg = 'yellow')
canvas2 = tkinter.Canvas(width = 400, height = 200, bg = 'green')
label1 = tkinter.Label(text = 'Napiš meno:')
entry1 = tkinter.Entry()
button1 = tkinter.Button(text = 'nakresli', command = nakresli)

canvas1.pack()
canvas2.pack()
label1.pack()
entry1.pack()
button1.pack()
```

```
import tkinter, random

def nakresli():
    x = random.randrange(400)
    y = random.randrange(200)
    canvas1.create_text(x, y, text = entry1.get())
    canvas2.create_text(x, y, text = entry1.get()[:-1])

canvas1 = tkinter.Canvas(width = 400, height = 200, bg = 'yellow')
canvas2 = tkinter.Canvas(width = 400, height = 200, bg = 'green')
label1 = tkinter.Label(text = 'Napiš meno:')
entry1 = tkinter.Entry()
button1 = tkinter.Button(text = 'nakresli', command = nakresli)

label1.pack()
entry1.pack()
button1.pack()
canvas2.pack()
canvas1.pack()
```

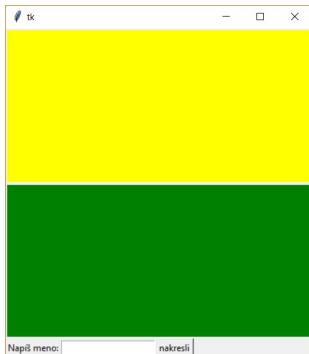
V programe sme použili widget **label**, ktorý slúži na vypísanie nejakého textu, napríklad k entry.



V metóde **pack()** môžeme použiť aj parameter **side** s hodnotou '**left**', '**right**'.

```
canvas1.pack()  
canvas2.pack()  
label1.pack(side = 'left')  
entry1.pack(side = 'left')  
button1.pack(side = 'left')
```

Takéto nastavenie poukladá canvasy pod seba a potom dá vedľa seba vľavo ostatné prvky.



Alebo si môžeme vytvoriť aj takéto usporiadanie:

```
canvas1.pack(side = 'right')  
label1.pack(side = 'left')  
entry1.pack(side = 'left')  
button1.pack(side = 'left')
```



Viacero widgetov môžeme zoskupiť do jedného (mysleného) rámkika - **frame**. A, samozrejme, widgetom môžeme nastavovať aj ďalšie vlastnosti (**width**, **bg**, **fg**, ...).

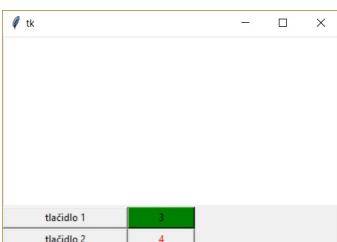
```
import tkinter  
canvas = tkinter.Canvas(width = 400, height = 200, bg = 'white')  
canvas.pack()  
  
frame1 = tkinter.Frame()  
frame1.pack(side = 'left')
```

```

button1 = tkinter.Button(frame1, text='tlačidlo 1', width = 20)
button1.pack()
button2 = tkinter.Button(frame1, text='tlačidlo 2', width = 20)
button2.pack()

frame2 = tkinter.Frame()
frame2.pack(side = 'left')
button3 = tkinter.Button(frame2, text = '3', width = 10, bg = 'green')
button3.pack()
button4 = tkinter.Button(frame2, text = '4', width = 10, fg = 'red')
button4.pack()

```



Na zobrazenie widgetu sa používajú aj metódy **grid()** a **place()**. Pri každom prvku môžeme na jeho zobrazenie použiť len jednu z metód. V metóde **place()** nastavujeme súradnice umiestenia widgetu v rámci okna aplikácie. V metóde **grid()** sa widgety umiestňuju do fiktívnej tabuľky pomocou nastavení stĺpca a riadku (viac nájdete na internete).

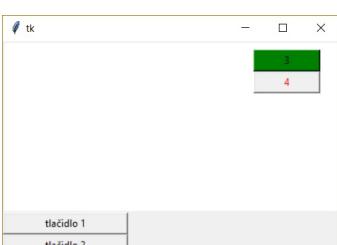
```

import tkinter
canvas = tkinter.Canvas(width = 400, height = 200, bg = 'white')
canvas.pack()

frame1 = tkinter.Frame()
frame1.pack(side = 'left')
button1 = tkinter.Button(frame1, text = 'tlačidlo 1', width = 20)
button1.pack()
button2 = tkinter.Button(frame1, text = 'tlačidlo 2', width = 20)
button2.pack()

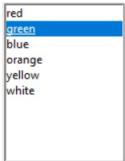
frame2 = tkinter.Frame()
frame2.place(x = 300, y = 10)
button3 = tkinter.Button(frame2, text = '3', width = 10, bg = 'green')
button3.pack()
button4 = tkinter.Button(frame2, text = '4', width = 10, fg = 'red')
button4.pack()

```



## Listbox

Listbox je **widget**, ktorý ukáže v ponuke viaceré možnosti a my môžeme (štandardne) z nich jednu označiť (v inom nastavenom režime sa dá vyberať aj viacero položiek).



Nasledujúci program vytvorí naplnený **listbox** zoznamom farieb. Po dvojkliku na farbu sa **canvas** prefarbí vybranou farbou. V programe je aj **entry** a dve tlačidlá. Do **entry** zadávame farbu a pridávame ju do listboxu tlačidlom. Iným tlačidlom z listboxu vymažeme označenú položku (farbu).

```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 200, bg = 'white')
canvas.pack()

def prefarbi(event):
    oznamene = listbox1.curselection()
    canvas['bg'] = listbox1.get(oznamene)

def pridaj():
    listbox1.insert('end', entry1.get())

def vymaz():
    oznamene = listbox1.curselection()
    if len(oznamene) == 1:
        listbox1.delete(oznamene)

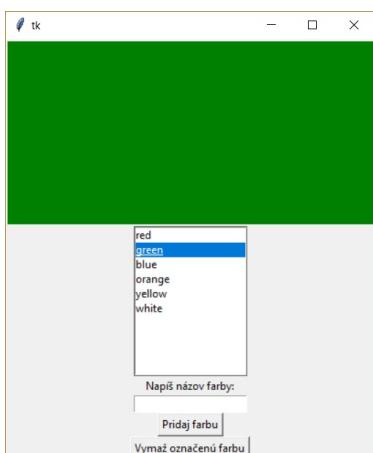
listbox1 = tkinter.Listbox()
listbox1.pack()

farby = ['red', 'green', 'blue', 'orange', 'yellow', 'white']

for prvok in farby:
    listbox1.insert('end', prvok)

listbox1.bind('<Double-Button-1>', prefarbi)

label1 = tkinter.Label(text = 'Napiš názov farby:')
label1.pack()
entry1 = tkinter.Entry()
entry1.pack()
button1 = tkinter.Button(text = 'Pridaj farbu', command = pridaj)
button1.pack()
button2 = tkinter.Button(text = 'Vymaž označenú farbu', command = vymaz)
button2.pack()
```



Vytvorenie listboxu a niektoré jeho metódy:

Listbox má riadky indexované od čísla 0,

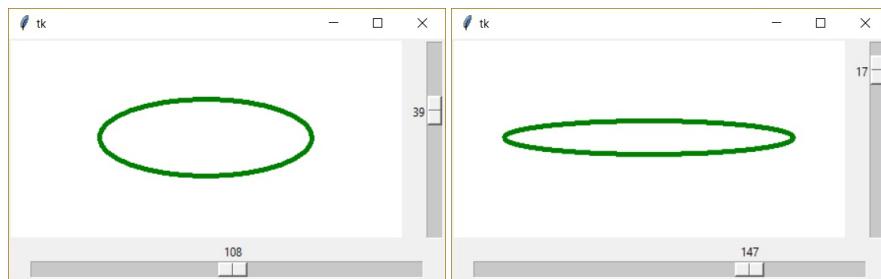
**listbox1 = tkinter.Listbox()** - vytvorenie listboxu. Môžeme nastaviť napríklad **height** - počet

riadkov listboxu.

`listbox1.insert('end', 'hodnota')` - vloží na koniec listboxu zadanú hodnotu,  
`listbox1.insert(index, 'hodnota')` - vloží pred zadaný index listboxu zadanú hodnotu,  
`listbox1.delete(index)` - vymaže hodnotu na zadanom indexe,  
`listbox1.delete(index1, index2)` - vymaže hodnoty medzi zadanými indexmi, vrátane indexov,  
`listbox1.curselection()` - vráti n-ticu označených riadkov (ich indexy),  
`listbox1.get(index)` - vráti hodnotu so zadaným indexom,  
`listbox1.bind('<Double-Button-1>', funkcia)` - zviaže udalosť dvojkliku na riadku listboxu so zadanou funkciou,  
`listbox1.size()` - vráti počet prvkov listboxu (posledný prvek má index o jedno menší),  
`listbox1.config(vlastnosť = hodnota)` - nastaví dodatočne niektorú z vlastností, ak sme ju nenastavili pri vytváraní.  
Napríklad: `listbox1.config(height = 5), listbox1.config(bg = 'red')`.

## Posúvač

Na zmenu číselných hodnôt môžeme použiť widget `scale` (posúvač). Vytvoríme program, ktorý nakreslí `oval` a dvoma posúvačmi budeme interaktívne meniť jeho polomer na osi x a polomer na osi y.



```
import tkinter
canvas = tkinter.Canvas(width = 440, height = 200, bg = 'white')
canvas.pack()

rx, ry = 100, 50
x, y = 200, 100
canvas.create_oval(x-rx, y-ry, x+rx, y+ry, width = 5, outline = 'green',
                   tags = 'oval')

def zmena1(event):
    global rx
    rx = scale1.get()
    prekresli()

def zmena2(event):
    global ry
    ry = scale2.get()
    prekresli()

def prekresli():
    canvas.coords('oval',[x-rx, y-ry, x+rx, y+ry])

scale1 = tkinter.Scale(from_ = 10, to = 200, orient = 'horizontal',
                      length = 400, command = zmena1)
scale1.pack()
scale1.set(rx)

scale2 = tkinter.Scale(from_ = 10, to = 100, orient = 'vertical',
                      length = 200, command = zmena2)
scale2.place(x = 400, y = 0)
scale2.set(ry)
```

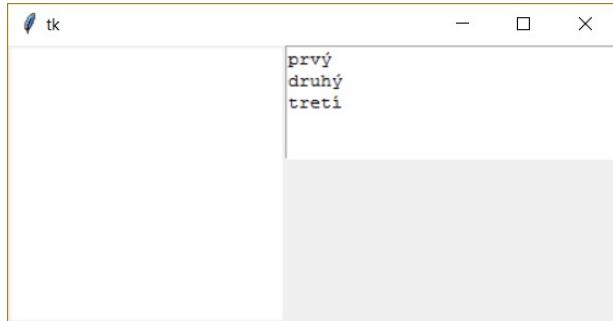
Vytvorenie `scale` a niektoré jeho metódy:

```
scale1 = tkinter.Scale(from_ = 10, to = 200) - vytvorenie widgetu scale. Môže obsahovať ďalšie nastavenia, napríklad: orient = 'horizontal' alebo orient = 'vertical', length = 400 (dĺžku widgetu), command = funkcia (pri zmene zavolá danú funkciu),  
scale1 = tkinter.Scale(from_ = 10, to = 200, orient = 'horizontal', command = zmena)  
scale1.get() - zistí aktuálnu hodnotu,  
scale1.set(hodnota) - nastaví aktuálnu hodnotu,  
scale1.config(from_ = 0) - zmení zadanú vlastnosť na zadanú hodnotu,  
scale1.config(showvalue = 0) - vypne zobrazovanie hodnoty (showvalue = 1 - zapne zobrazovanie hodnoty),  
scale1.config(label = 'polomer v osi x') - nastaví popis súčiastky.
```

## Textová plocha

Doposiaľ sme dlhší text vypisovali do shellu, ale môžeme vytvoriť aj widget `text` priamo v okne aplikácie. Widget `text` nám môže priomínať viacriadkové `entry`. Do tohto widgetu píšeme text príkazmi programu, alebo ho môže editovať priamo aj používateľ. Nasledujúci program vytvorí `canvas` a pod ním widget `text`. Samozrejme, ak nepoužívame `canvas`, nie je potrebné ho vytvárať. Do widgetu `text` pridáme metódou `insert` niekoľko riadkov. Podobne ako pri zápisе textového súboru, koniec riadku zapíšeme pomocou '`\n`'.

```
import tkinter  
canvas = tkinter.Canvas(width = 200, height = 200, bg = 'white')  
canvas.pack()  
  
text1 = tkinter.Text(height = 5, width = 30)  
text1.pack()  
  
text1.insert('end', 'prvy\n')  
text1.insert('end', 'druhy\n')  
text1.insert('end', 'treti\n')
```



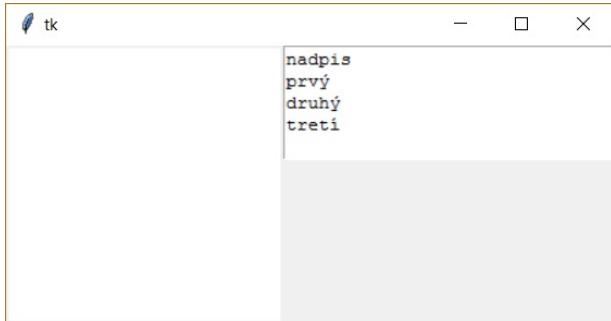
Metóda `get(od, do)` nám vráti obsah widgetu `text` podľa zadaného rozsahu. Rozsah (`od` a `do`) zadávame v tvare `číslo_riadku.číslo_stĺpca` alebo v `do` použijeme '`end`', ak chceme rozsah do konca widgetu. Pozor, prvý riadok má číslo 1, ale prvý stĺpec má číslo 0.

```
>>> text1.get(1.0, 1.2)  
'pr'  
>>> text1.get(1.1, 1.2)  
'r'  
>>> text1.get(1.0, 2.0)  
'prvý\n'  
>>> text1.get(1.0, 3.0)  
'prvý\ndruhý\n'  
>>> text1.get(1.0, 'end')  
'prvý\ndruhý\n\ttreti\n\n'  

```

>>>

Vkladať metódou `insert` nemusíme len na koniec, ale aj na konkrétnu pozíciu, napríklad `text1.insert(1.0, 'nadpis\n')`.



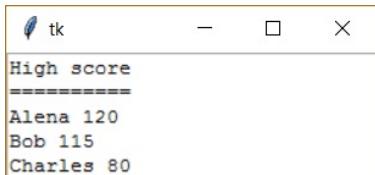
Niekteré metódy widgetu `text`:

`text1.get(od, do)` - vráti text v rozsahu od, do,  
`text1.get('current')` - vráti znak pod kurzorom myši,  
`text1.insert(pozícia, 'retazec')` - vloží na zadanú pozíciu zadaný retazec. Pozícia je v tvare `číslo_riadku.číslo_stĺpca`, alebo `'end'`,  
`text1.count(1.0, 'end')` - vráti počet znakov (n-ticu) medzi zadanými pozíciami,  
`text1.config(vlastnosť = hodnota)` - nastaví novú hodnotu pre zadanú vlastnosť, napríklad `width = 20`,  
`text1.config(state = hodnota)` - nastaví stav (`'normal'` - štandardný stav, `'disabled'` - používateľ nemôže meniť obsah),  
`text1.selection_get()` - ak je označená časť textu (selektovaná), vráti tento text, inak hľasi chybu.

```
import tkinter
text1 = tkinter.Text(height = 5, width = 30)
text1.pack()

f = open('score.txt', 'r')
for riadok in f:
    text1.insert('end', riadok)
f.close()
```

Widget `text` môžeme používať napríklad na zobrazovanie obsahu textových súborov.



Pre dlhšie texty môžeme vytvoriť k widgetu `text` aj posúvač s ním previazaný (analogicky to použijeme s listboxom).

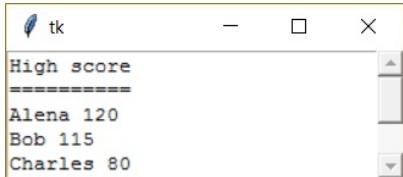
```
import tkinter
text1 = tkinter.Text(height = 5, width = 30)
text1.pack(side = 'left')

scrollbar1 = tkinter.Scrollbar()
scrollbar1.pack(side = 'right', fill = 'y')
    #vyplní s ním celý volný priestor v rozsahu osi y
scrollbar1.config(command = text1.yview)
    #po posunuti sa posunie aj obsah text1
text1.config(yscrollcommand = scrollbar1.set)
    #po posunutí obsahu text1 sa posunie aj scrollbar1
```

```

f = open('score.txt', 'r')
for riadok in f:
    text1.insert('end', riadok)
f.close()

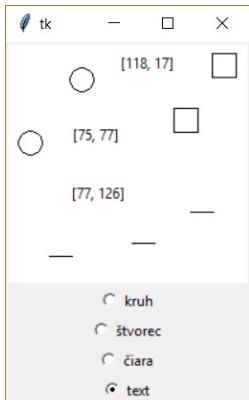
```



## Radiobutton

Widget `radiobutton` sa nepoužíva samostatne, ale ako skupina možností, z ktorých môže byť vybratá maximálne jedna možnosť. Pri radiobuttonoch sa na zisťovanie hodnoty používa špeciálny typ premennej z knižnice `tkinter` (buď `tkinter.IntVar()` alebo `tkinter.StringVar()`). Najprv sa vytvorí táto premenná a previaže sa so všetkými radiobuttonmi v jednej skupine (nastavením vlastnosti `variable = meno_premennej`). Na zistenie hodnoty použijeme metódu `get()` priamo tej premennej, ktorú sme vytvorili a je s radiobuttonmi previazaná. Hodnotu jednotlivých radiobuttonov určujeme nastavením vlastnosti `value = hodnota`. Podľa toho, či sú hodnoty čísla alebo textové reťazce, použijeme na previazanie premenného typu `IntVar()` alebo `StringVar()`.

V nasledujúcim programe budeme pomocou radiobuttonov vyberať, ktorý útvar chceme nakresliť kliknutím myši do canvasu.



```

import tkinter
canvas = tkinter.Canvas(width = 200, height = 200, bg = 'white')
canvas.pack()
v = tkinter.IntVar()
radiobutton1 = tkinter.Radiobutton(text = 'kruh', variable = v, value = 1)
radiobutton1.pack()
radiobutton2 = tkinter.Radiobutton(text = 'štvorec', variable = v, value = 2)
radiobutton2.pack()
radiobutton3 = tkinter.Radiobutton(text = 'čiara', variable = v, value = 3)
radiobutton3.pack()
radiobutton4 = tkinter.Radiobutton(text = 'text', variable = v, value = 4)
radiobutton4.pack()

def klik(sur):
    typ = v.get()
    if typ == 1:
        canvas.create_oval(sur.x-10, sur.y-10, sur.x+10, sur.y+10)
    if typ == 2:
        canvas.create_rectangle(sur.x-10, sur.y-10, sur.x+10, sur.y+10)
    if typ == 3:

```

```

    canvas.create_line(sur.x-10, sur.y, sur.x+10, sur.y)
if typ == 4:
    canvas.create_text(sur.x, sur.y, text = '['+str(sur.x)+',
        '+str(sur.y)+']')

canvas.bind('<Button-1>', klik)

```

Ak by sme v programe nepoužívali canvas, program nám po spustení hlási chybu:

**AttributeError: 'NoneType' object has no attribute '\_root'**. Pri vytváraní canvasu sa automaticky použité widgety previažu s oknom aplikácie. Radiobuttonom sa nepodarí z nejakého dôvodu zistieť, v ktorom prvku (mieste) sú umiestnené, chýba im atribút '**\_root**'.

```

import tkinter

v = tkinter.IntVar()
radiobutton1 = tkinter.Radiobutton(text = 'kruh', variable = v, value = 1)
radiobutton1.pack()
radiobutton2 = tkinter.Radiobutton(text = 'štvorec', variable = v, value = 2)
radiobutton2.pack()
radiobutton3 = tkinter.Radiobutton(text = 'čiara', variable = v, value = 3)
radiobutton3.pack()
radiobutton4 = tkinter.Radiobutton(text = 'text', variable = v, value = 4)
radiobutton4.pack()

```

Riešením je to, že príkazom **okno = tkinter.Tk()** vytvoríme okno hlavnej aplikácie a referenciu naň si zapamätáme do premennej **okno**. Následne pri vytváraní radiobuttonu ako prvý parameter uvedieme premennú **okno**, čiže referenciu na nadradené miesto, v ktorom budú radiobuttony existovať. Všetky widgety môžu mať prvý parameter referenciu na hlavné okno, v ktorom sú umiestnené. My sme to v našich programoch zatiaľ nepoužívali, lebo to nebolo potrebné. Ak ste na internete hľadali ďalšie materiály, tam ste sa s tým určite stretli. Iné materiály najprv vytvárajú okno, napríklad takto: **root = tkinter.Tk()** alebo **master = tkinter.Tk()** a pod..

A takto zapíšeme program, ktorý aj vybranú hodnotu vypíše:

```

import tkinter
okno = tkinter.Tk()
v = tkinter.IntVar()

def vypis():
    print(v.get())

radiobutton1 = tkinter.Radiobutton(okno, text = 'kruh', variable = v,
                                  value = 1, command = vypis)
radiobutton1.pack()
radiobutton2 = tkinter.Radiobutton(okno, text = 'štvorec', variable = v,
                                  value = 2, command = vypis)
radiobutton2.pack()
radiobutton3 = tkinter.Radiobutton(okno, text = 'čiara', variable = v,
                                  value = 3, command = vypis)
radiobutton3.pack()
radiobutton4 = tkinter.Radiobutton(okno, text = 'text', variable = v,
                                  value = 4, command = vypis)
radiobutton4.pack()

```

Na prednastavenie radiobuttonu na zapnutý použijeme metódu **select()** - **radiobutton1.select()** a na odznačenie všetkých radiobuttonov v skupine použijeme metódu **deselect()** - **radiobutton1.deselect()**.

Kedže radiobuttony sú previazané premenou, nemusíme si pri ich vytváraní zapamätať referenciu na ne, a teda môžeme ich vytvoriť aj v cykle zo zoznamu položiek a hodnôt. Aby boli všetky radiobuttony zarovnané podľa ľavého okraja (a nie stredu), nastavíme im v metóde **pack()** **anchor = 'w'** (anchor sme už používali

pri vykresľovaní textu a obrázkov).

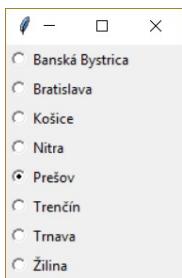
```
import tkinter
okno = tkinter.Tk()

v = tkinter.StringVar()

mesta = [('Banská Bystrica', 'BB'), ('Bratislava', 'BA'), ('Košice', 'KE'),
         ('Nitra', 'NR'), ('Prešov', 'PO'), ('Trenčín', 'TN'), ('Trnava', 'TT'),
         ('Žilina', 'ZA')]

def vypis():
    print(v.get())

for mesto, skratka in mesta:
    rb = tkinter.Radiobutton(okno, text = mesto, value = skratka,
                             variable = v, command = vypis)
    rb.pack(anchor = 'w')
```

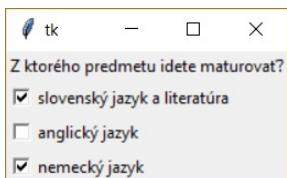


Na zmenu hodnoty z programu použijeme metódu `set()` - premennej, s ktorou sú radiobuttony previazané:

```
>>> v.set('BB')
>>> v.set('TT')
>>>
```

## Checkbutton

Ak potrebujeme z ponuky vyberať viacero možností, namiesto radiobuttonov je vhodnejšie použiť checkbuttony. V ukážke si môže používateľ vybrať maturitné predmety.



```
import tkinter

def vypis():
    predmety = predmet1.get() + ' ' + predmet2.get() + ' ' + predmet3.get()
    predmety = predmety.strip()
    print('Práve máte vybraté predmety:', predmety)

label1 = tkinter.Label(text = 'Z ktorého predmetu idete maturovať?')
label1.pack()
predmet1 = tkinter.StringVar()
checkbutton1 = tkinter.Checkbutton(text = 'slovenský jazyk a literatúra',
                                    onvalue = 'Sjl', offvalue = '', variable = predmet1,
                                    command = vypis)
checkbutton1.pack(anchor = 'w')
```

```

predmet2 = tkinter.StringVar()
checkbutton2 = tkinter.Checkbutton(text = 'anglický jazyk', onvalue = 'AJ',
                                   offvalue = '', variable = predmet2, command = vypis)
checkbutton2.pack(anchor = 'w')

predmet3 = tkinter.StringVar()
checkbutton3 = tkinter.Checkbutton(text = 'nemecký jazyk', onvalue = 'NJ',
                                   offvalue = '', variable = predmet3, command = vypis)
checkbutton3.pack(anchor = 'w')

```

Každý z checkbuttonov je samostatne previazaný s premennou typu `StringVar()` alebo `IntVar()` (podľa typu použitých hodnôt). K hodnote sa dostaneme pomocou previazanej premennej a jej metódy `get()`. Pre checkbutton definujeme hodnotu, pre zakliknutý stav `onvalue` a hodnotu pre neoznačený stav `offvalue`. V našej ukážke bolo vhodné nastaviť `offvalue` na prázdny textový reťazec. Metódou `set()` nastavíme premennej aktuálnu hodnotu, ktorá sa zároveň zobrazí v stave checkbuttonu.

```

>>> predmet1.set('SJL')
>>> predmet1.set('')
>>> predmet2.set('AJ')
>>> predmet2.set('')
>>>

```

Pozor, ak v programe používame iba chceckbuttony, tiež potrebujeme vytvoriť referenciu na hlavné okno a použiť ju pri vytváraní checkbuttonov (podobne ako v radiobuttonoch). My sme to nepotrebovali urobiť, lebo sme vytvorili aj widget `label`.

Nasledujúci program nám vytvorí ponuku checkbuttonov - predmetov zo súboru `predmety.txt`. Na každom riadku je práve jeden maturitný predmet v tvare `celý_názov;skratka`. Kedže ku každému checkbuttonu potrebujeme premennú na čítanie a nastavovanie hodnôt a previazanie s checkbuttonom, vytvoríme si z nich zoznam. Pri výpise prechádzame zoznam premenných jednotlivých checkbuttonov a vytvárame výslednú informáciu na výpis. Kedže widgety sú previazané s premennými (v zozname), nepotrebjeme na každý z nich samostatnú referenciu. Ak by sme ju predsa len potrebovali, aj referencie na vytvorené widgety si môžeme uložiť do zoznamu (môžeme mať zoznam canvasov, buttonov a pod). V programe sme vytvorili aj tlačidlo reset, ktoré prejde celý zoznam premenných zviazaných s checkbuttonmi a nastaví metódou `set()` ich hodnotu na hodnotu nastavenú pre `offvalue`.

```

import tkinter

def vypis():
    predmety = ''
    for prvok in informacie:
        predmety = predmety + ' ' + prvok.get()
    predmety = predmety.strip()
    print('Práve máte vybraté predmety:', predmety)

def reset():
    for informacia in informacie:
        informacia.set('')
    vypis()

subor = open('predmety.txt', 'r')
informacie = []
label1 = tkinter.Label(text = 'Z ktorého predmetu idete maturovať?')
label1.pack()

for riadok in subor:
    riadok = riadok.strip()
    info = riadok.split(';')
    informacie.append(tkinter.StringVar())
    chb = tkinter.Checkbutton(text = info[0], onvalue = info[1], offvalue = '',

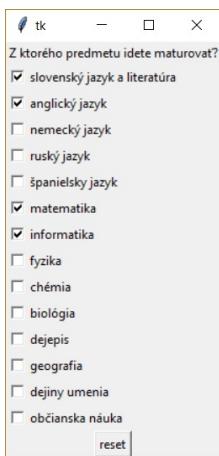
```

```

        variable = informacie[-1], command = vypis)
chb.pack(anchor = 'w')
subor.close()

button1 = tkinter.Button(text = 'reset', command = reset)
button1.pack()

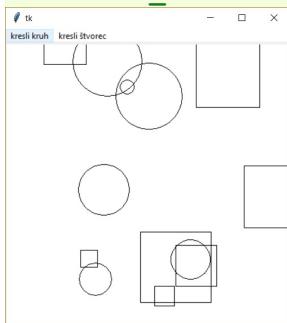
```



## 10.1 Menu

Pomocou knižnice `tkinter` vytvoríme aj hlavné menu. Opäť sa nevyhneme vytvoreniu referencie na hlavné okno aplikácie, podobne, ako sme to robili v radiobuttonoch. Najprv si ukážeme jednoduché hlavné menu, ktoré má len ponuku na hlavnej úrovni a zatiaľ nerozbaľuje ďalšie podmenu. Menu vytvárame podobne ako ostatné widgety `menu1 = tkinter.Menu(okno)`. V parametri uvedieme referenciu na prvok (niečo), v ktorom bude menu umiestnené. Úplne hlavnej lište menu nastavíme referenciu na samotné okno aplikácie. Aby sa `menu1` v okne aplikácie zobrazilo, musíme mu ho nastaviť `okno.config(menu = menu1)`. Potom do `menu1` pridávame metódou `add_command()` položky tohto menu. Pri pridávaní nastavujeme ich popis (label) a meno funkcie, ktorá sa má vykonať po výbere tejto položky -

```
menu1.add_command(label = 'kresli kruh', command = kruh).
```



```

import tkinter, random

okno = tkinter.Tk()
canvas = tkinter.Canvas(width = 400, height = 400, bg = 'white')
canvas.pack()

def kruh():
    x, y = random.randrange(400), random.randrange(400)
    r = random.randint(10,50)
    canvas.create_oval(x-r, y-r, x+r, y+r)

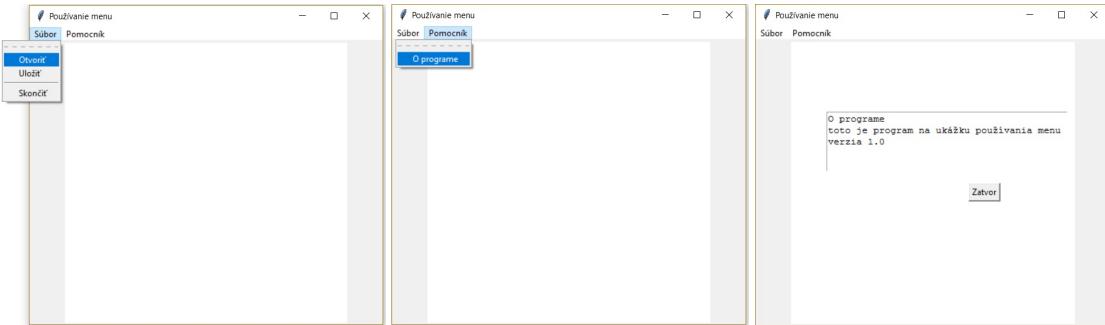
def stvorec():
    x, y = random.randrange(400), random.randrange(400)
    r = random.randint(10, 50)

```

```
    canvas.create_rectangle(x-r, y-r, x+r, y+r)

menu1 = tkinter.Menu(okno)
okno.config(menu = menu1)
menu1.add_command(label = 'kresli kruh', command = kruh)
menu1.add_command(label = 'kresli štvorec', command = stvorec)
```

Menu ďalšej úrovne vytvoríme zvlášť. Pri vytváraní mu nastavíme umiestnenie nie v hlavnom okne, ale v menu v nadradenej úrovni. Následne ho potom pridáme v nadradenom menu metódou `add_cascade()`. V nasledujúcom programe je vytvorené `menu1` - hlavná lišta s dvomi možnosťami, pričom každá z nich obsahuje ďalšie menu (`menu2` a `menu3`). Niektoré funkcie sú zatiaľ prázdne, aby tu Python nehlásil chybu, máme tam príkaz `pass` (príkaz sa používa aj v `if`-e, ak potrebujeme mať vetvu prázdnu). V `menu3` je položka, ktorá ukáže `text` s popisom programu a tlačidlo na zatvorenie popisu a tlačidla. Tu používame metódu `place_forget()` alebo `pack_forget()`, ktorá zruší zobrazenie widgetu (podľa toho, ktorou metódou bol widget zobrazený). V programe vidíme aj nastavenie nadpisu hlavného okna `okno.title('Používanie menu')` a jeho rozmerov `okno.geometry('500x400')`.



Na oddelenie jednotlivých položiek menu zavoláme metódu vloženia separátora `menu2.add_separator()`.

## 10.2 Dialógové okná

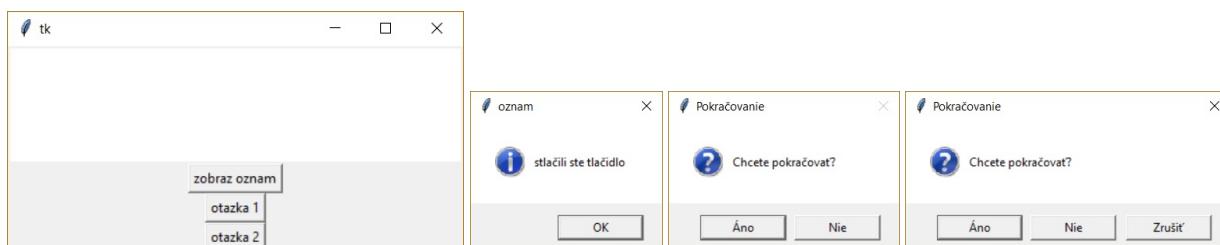
Rôzne oznamy a otázky riešime v programoch pomocou dialógových okien. Ak máme spustené aplikačné okno a importovaný modul `tkinter`, môžeme dialógové okno vyvolať aj priamo zo shellu `tkinter.messagebox.askyesno('titulok', 'text oznamu')`. Je to funkcia, ktorá vráti nejakú hodnotu, podľa toho, ktorú metódu messageboxu sme zavolali (sú rôzne typy: `showerror`, `showinfo`, `askyesno`, `askokcancel`, `askyesnocancel` a pod.).

```
import tkinter
canvas = tkinter.Canvas(width = 400, height = 100, bg = 'white')
canvas.pack()
def oznam():
    vysledok = tkinter.messagebox.showinfo('oznam', 'stlačili ste tlačidlo')
    print(vysledok)

def otazka1():
    vysledok = tkinter.messagebox.askyesno('Pokračovanie', 'Chcete pokračovať?')
    print(vysledok)

def otazka2():
    vysledok = tkinter.messagebox.askyesnocancel('Pokračovanie',
                                                'Chcete pokračovať?')
    print(vysledok)

button1 = tkinter.Button(text = 'zobraz oznam', command = oznam)
button1.pack()
button2 = tkinter.Button(text = 'otazka 1', command = otazka1)
button2.pack()
button3 = tkinter.Button(text = 'otazka 2', command = otazka2)
button3.pack()
```



Na výber farby a mena súboru sa dajú zavolať aj špeciálne dialógové okná. Ich použitie si ukážeme priamo v programe:

```
import tkinter
canvas = tkinter.Canvas(width = 200, height = 200, bg = 'white')
canvas.pack(side = 'left')
```

```

def nacitaj():
    info = tkinter.filedialog.askopenfile()
    if info != None:
        print('Meno súboru:' + info.name)
        print('Kódovanie znakov:' + info.encoding)
        subor = open(info.name, 'r', encoding = info.encoding)
        obsah = subor.read()
        subor.close()
        text1.delete(1.0, 'end')
        text1.insert('end', obsah)

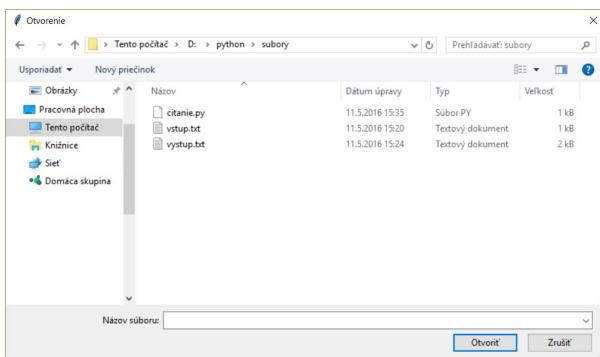
def farba():
    farba = tkinter.colorchooser.askcolor()
    print(farba)
    cislo_farby = farba[1]
    if cislo_farby != None:
        canvas['bg'] = cislo_farby
        print('zmenil som farbu na:', cislo_farby)
    else:
        print('farba nebola zmenená')

text1 = tkinter.Text(width = 60, height = 10)
text1.pack()

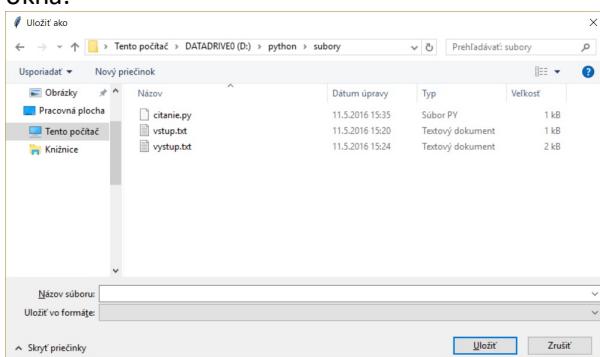
button1 = tkinter.Button(text = 'otvor súbor', command = nacitaj)
button1.pack()
button2 = tkinter.Button(text = 'vyber farbu', command = farba)
button2.pack()

```

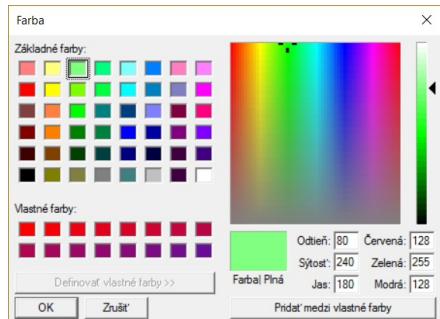
Funkcia `tkinter.filedialog.askopenfile()` otvorí klasické dialógové okno na výber súboru a na výstup vráti informácie o výbere. Ak okno zrušíme, výstupom je `None`. Inak sú výstupom akési zložené údaje. Meno z nich vytiahneme cez `.name` a informáciu o kódovaní súboru cez `.encoding`. Náš program prečíta obsah celého súboru a vloží ho do widgetu `text1`.



Podobne funguje funkcia `tkinter.filedialog.asksaveasfile()`, rozdiel je len v popisoch dialógového okna.



Funkcia `tkinter.colorchooser.askcolor()` otvorí dialógové okno na výber farby. Po jeho zrušení je na výstupe funkcie `None`. Pri výbere farby je na výstupe n-tica, z ktorej je pre nás zaujímavý prvok s indexom 1 - farba v hexadecimálnej sústave. Nás program nastaví pozadie canvasu vybratou farbou.



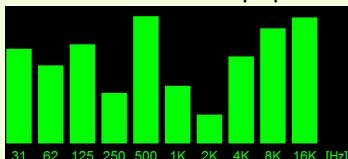
# 11 Úlohy na opakovanie

**1** Vytvorte simulátor záhradky. Niekoľko náhodne rozmiestnených kvetov (farebných krúžkov) postupne vädne – zmenšuje sa. Klikaním na ne ich polejete, a teda obnovíte ich pôvodnú veľkosť. Ak sa ale kvietok zmenší na kritickú veľkosť, zanikne (uschnie).

**2** Doplňte a upravte program z predchádzajúcej úlohy a vytvorte z neho hru. Pridajte zrýchlovanie vädnutia v závislosti od zväčšujúceho sa sucha a bodovanie.

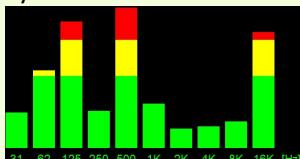
**3** Upravte program tak, že kvet nebude vykreslený iba ako jeden krúžok, ale bude mať aj lupene. Nezabudnite, že kliknúť sa bude dať aj na lupene. Lupene môžete vykresliť vlastnou funkciou, alebo si pripravte png, gif obrázky a použite ich v programe. V hre môžete upraviť bodovanie a náročnosť na zalievanie podľa druhov kvetín (vzácnosť kvetiny, požadované podmienky a pod.). Podmienky rastlín, informácie o nich a meno zodpovedajúceho obrázku môžu byť uložené v textovom súbore.

**4** Vytvorte program, ktorý bude simulať hudobný grafický ekvalizér. Hodnoty pre jednotlivé výšky sa budú meniť v tomto prípade náhodne v rovnakých časových intervaloch, keďže ide iba o simulátor.



Pozn. Hodnoty jednotlivých výškových pásem, ktoré sa vypíšu na osi X, si pripravte do n-tice.

**5** Upravte program s grafickým ekvalizérom tak, aby boli vyššie hlasitosti (nad polovičným rozsahom) vykreslené žltou farbou a najvyššie hlasitosti (v najvyššej štvrtine rozsahu) červenou farbou.



**6** Vytvorte program, ktorý bude slúžiť ako pomôcka pre učiteľa pri ústnom skúšaní. Predpokladajme, že v triede (seminár) je desať študentov a každý študent odpovedá z vopred pripravených otázok. Pre zjednodušenie má každý študent pridelené číslo (podľa abecedy, začína sa číslom 1 a končí číslom 10). Otázky sú číslované číslami 1 až 10. Program má nasledujúce vlastnosti:

- vypíše náhodné poradie, v akom pôjdu študenti odpovedať,
- ku každému študentovi vypíše aj náhodné číslo otázky, z ktorej bude odpovedať,
- otázky, z ktorých študenti odpovedajú, sa nesmú opakovať,
- programu môžeme zadať počet študentov v triede a aj počet otázok. Program zistí, či nie je počet otázok menší ako počet študentov, a na takúto chybu nás upozorní.

**7** Vytvorte program, ktorý bude pre žiakov prvého stupňa vytvárať náhodné príklady na súčet alebo súčin čísel od 1 do 20. Predpokladajme, že program nám zadá maximálne 20 príkladov. Program kontroluje našu odpoveď, počíta nám body a v závere vypíše našu úspešnosť. Zabezpečte, aby sa príklady neopakovali. Program príklady uloží do textového súboru.

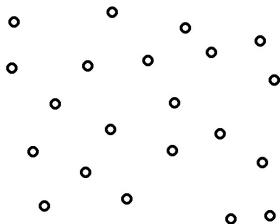
Výsledné úlohy môžu vyzerať napríklad takto:

- 1)       $3 + 4 =$
- 2)       $7 * 8 =$
- 3)       $16 + 8 =$
- 4)       $11 * 2 =$
- 5)       $2 + 18 =$
- 6)       $13 * 3 =$

8

Vytvorte program, ktorý bude fungovať nasledovne:

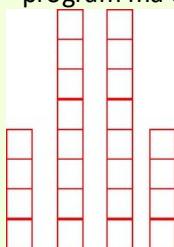
- kliknutím myšou do canvasu pridáme na obrazovku loptičku,
- všetky loptičky padajú smerom rovno dole k spodnému okraju obrazovky (všetky majú rovnakú rýchlosť),
- ak loptička dosiahne dolný okraj, padá znova z horného okraja,
- loptičky môžeme neustále pridávať kliknutím myši, program je pripravený zvládnuť maximálne 100 loptičiek,
- program nespadne pri ovládaní používateľom (ani v prípade, že chce používateľ pridať viac loptičiek ako je program schopný zvládnuť).



9

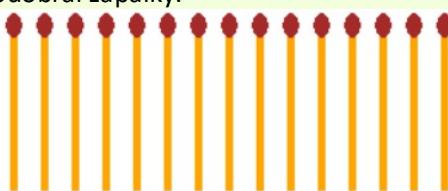
Vytvorte program generátor sídliska:

- na začiatku hry sa na obrazovke ukážu vedľa seba náhodne vysoké paneláky (jeden panelák má šírku 10 pixelov, výška jedného poschodia je 10 pixelov),
- paneláky môžu mať maximálne 30 poschodí,
- v hre sa môžu vyskytnúť aj paneláky s nulovou výškou – teda medzi niektorými môže byť aj medzera,
- po zatlačení tlačidla sa bude na obrazovke simulovať pohyb panelákov od ľavého okraja k pravému okraju,
- panelák, ktorý sa stratí z obrazovky, sa znova ukáže ako prvý na ľavej strane,
- paneláky môžu mať ďalšie vlastnosti, ako napríklad farbu, tabuľku s číslom domu a pod.,
- program má obmedzenie, nie je môžné použiť príkaz move.



10

Vytvorte program, ktorý umožní hrácom hrať hru NIM. Hra NIM býva v rôznych variantoch. Vy vytvorte hru tak, že program vykreslí zadaný počet zápaliek. Zápalky sa vykreslia na obrazovku. Hru hrajú striedavo dva hráči. Program píše, ktorý hráč je na tahu, a umožní mu zobrať 1, 2, alebo 3 zápalky. Prehráva ten hráč, ktorý odobral poslednú zápalku. Program nakoniec vyhodnotí, kto vyhral. Po stlačení tlačidla Repríza program prehrá hru (simuluje ju) a ukáže nám, kto, koľko a v akom poradí odobral zápalky.



11

Vytvorte program, ktorý simuluje preteky autičok.

Na ľavom okraji obrazovky vykreslí 15 autičok. Autička sa nachádzajú na zvislej úsečke. Každé autičko má svoju náhodnú farbu a súťažné číslo. Po stlačení nejakého klávesu spustíme animáciu pretekov autičok. Autička sa budú naraz pohybovať smerom vpravo, no zakaždým sa každé z nich posunie náhodne o 5 až 10 bodov.

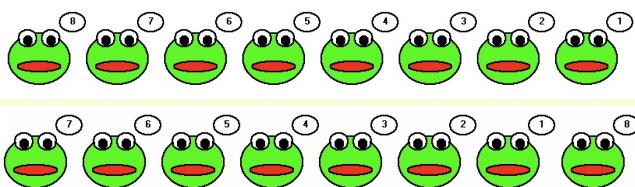
Program skončí, keď všetky autička dosiahnú pravý okraj obrazovky.

Na obrazovke sa vypisujú čísla autičok v poradí, v akom prišli do cieľa.

**12**

Vytvorte program tanec žabiek. V programe je tlačidlo, ktoré zapne / vypne tanec žabiek. Choreografia tanca je nasledovná:

- prvá žabka vľavo sa posunie na koniec úplne vpravo,
- všetky ostatné žabky sa posunú o jednu pozíciu vľavo,
- žabky si takto menia svoje pozície v pravidelných časových intervaloch, až kým nevypneme tanec tlačidlom.

**13**

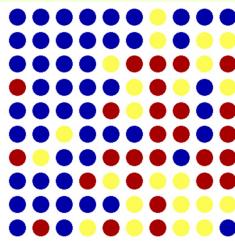
Počítač si myslí náhodné číslo. Používateľ zadáva svoje tipy a dostáva odozvu, či číslo uhádol, alebo je to nižšie alebo vyššie číslo. Počítač si pamäta tipy hráča. Ak hráč zadá rovnaký tip ako v minulosti, počítač ho na to upozorní. Nakoniec počítač vyhodnotí, na koľko tipov sa hráčovi podarilo číslo uhádnuť a ktoré čísla zadal opakovane. Priebeh hry zapíše do textového súboru.

**14**

Používateľ si myslí číslo. Počítač zadá dolnú a hornú hranicu intervalu, v ktorom sa mysené číslo nachádza. Úlohou počítača je uhádnuť číslo najmenším možným počtom krokov. Keď vytvoríte tento program, porovajte vaše programy vytvorené vo vašej skupine zadaním rovnakého vstupu (myslite si rovnaké číslo) a vyhodnoťte, ktorý z vytvorených programov uhádol mysené číslo najmenším počtom pokusov.

**15**

Vytvorte hru kruhy. V hracej ploche sa vykreslí  $10 \times 10$  farebných kruhov. Kruhy sú zafarbené náhodne jednou z troch farieb (červená, modrá, žltá). Úlohou hráča je zafarbiť všetky kruhy rovnakou farbou. Hra sa ovláda ľavým a pravým tlačidlom myši. Kliknutím ľavého tlačidla myši sa zmení farba celého stĺpca, nad ktorým je kurzor myši. Kliknutím pravého tlačidla myši sa zmení farba celého riadku, nad ktorým je kurzor myši. Po stlačení klávesu + sa zmení farba jednej diagonály a po stlačení klávesu - sa zmení farba druhej diagonály. Pri zmene farieb (stĺpca, riadku alebo diagonály) sa zmení farba každého kruhu (danej oblasti) tak, že vždy sa mení cyklicky v poradí farieb červená, modrá, žltá, červená. Program má dané obmedzenie - je potrebné využiť si pamätanie informácií pomocou vlastností útvarov v canvase.

**16**

Upravte program kruhy tak, že priebeh hry uloží do textového súboru a pomocou iného vytvoreného programu vieme priebeh hry prehrať. Navrhnite štruktúru textového súboru optimálne vzhľadom na veľkosť zapísaných údajov. O svojich návrhoch a ich efektívnosti diskutujte v triede.

**17**

Máme dva textové súbory `x.txt` a `y.txt`. V každom z nich sú uložené informácie o naklikaných bodoch v canvase - ich príslušnej zložke (`x` alebo `y`). Vytvorte program Prekvapenie, ktorý bude čítať súradnice `x` a `y` a podľa týchto súradníc postupne (animáciou) vykresli na zafarbené pozadie malé biele kruhy.

**18**

Vytvorte program, ktorý zo súborov `x.txt` a `y.txt` vytvorí nový textový súbor `súradnice.txt`, kde spojí súradnicu `x` (z príslušného riadku) so súradnicou `y` (z príslušného riadku) a v súbore ich zapíše v tvare `x; y` (čiže spoločne na jeden riadok).

19

Upravte program Prekvapenie tak, že postupnosť bodov  $x$  a  $y$  zo súborov  $x.txt$  a  $y.txt$  si uloží do zloženého dátového typu. Používateľ pomocou ovládacieho prvku nastaví koeficient zmenšenia / zväčšenia týchto súradníc a kliknutím myši do canvasu sa všetky body nakreslia v zmenšenej mierke (podľa nastavenia) s posunutím vzhľadom na miesto kliknutia (miesto kliknutia je ľavým horným bodom umiestnenia celej postupnosti bodov). Viacnásobným kliknutím myši pečiatkujeme pripravenú postupnosť bodov do canvasu v nastavenej mierke.

20

Vytvorte program, v ktorom používateľ bude ovládať kresliaceho robota. Na začiatku programu bude robot v strede grafickej plochy a je natočený smerom na sever. Samotného robota nezobrazujte, vidieť bude iba jeho kresby. Robot poslúcha na zadane príkazy a podľa nich kreslí obrázky. Príkazy zadávame do widgetu `entry` a potvrdzujeme ich tlačidlom vykonáť. Robot pozná tieto príkazy:

`ciara <dĺzka>` - nakreslí čiaru zadanej dĺžky v smere, v ktorom je natočený (napr. `ciara 50`),  
`vlavo` – otočí sa o 90 stupňov vľavo (relatívne od aktuálneho natočenia),  
`vpravo` – otočí sa o 90 stupňov vpravo (relatívne od aktuálneho natočenia).

- a) Program umožňuje opakované zadanie (vstupom z `entry`) jedného z textových príkazov `ciara`, `vpravo`, `vlavo`. Zadaný príkaz robot ihneď vykoná.
- b) Používateľ môže napísť naraz viac príkazov (akoby program pre robota) a až po stlačení spúšťacieho tlačidla sa príkazy vykonajú.
- c) Sekvenciu príkazov pre robota pripravte do textového súboru. Program po zadaní názvu pripraveného súboru bude vedieť príkazy vykonať a nakresliť tak výsledný obrazec.
- d) Doprogramujte pre robota aj príkazy nastavenia vzhľadu čiary – napr. `farbaper blue`, `hrubkapera 5`.
- e) Program pre robota v textovom súbore umožňuje zadanie príkazu `opakuj`, ktorý zopakuje  $N$ -krát zadané príkazy. Syntax príkazu `opakuj` si zvoľte sami. Napr.

```
opakuj 4
ciara 70
vlavo
koniecopakuj
```

- f) Upravte správanie sa príkazov `vlavo` a `vpravo`, aby ešte vyžadovali vstupný parameter, o koľko stupňov sa robot otočí, takže pomocou príkazu `ciara` bude schopný kresliť aj šikmé čiary.
- h) Pridajte do programu tlačidlo na uloženie kresby do súboru.

# Príkazovník

## Textové reťazce

```
+, *, int(reťazec), str(číslo)           reťazec.find(podreťazec)
type(premenná)                            reťazec.lower()
premenná = input('Zadajte ...')            reťazec.upper()
for znak in reťazec:                      reťazec.replace(podreťazec1, reťazec2)
len(reťazec)                             reťazec_formátu.format(parameter1, parameter2, ...)
reťazec[i], reťazec[odkial:pokial+1:krok] 'Ahoj {}'.format(meno)
ord(znak), chr(kód_znaku)                  '#{:02x}{:02x}{:02x}'.format(r, g, b)
podreťazec in reťazec
True, False
```

## N-tice

```
ntica = ()      ntica = (1,)          +, *, in
for prvok in ntica:                         ntica[i], ntica[odkial:pokial+1:krok]
len(ntica)                                a, b = 10, 20      a, b = b, a
```

## Textové súbory

```
subor = open('meno_súboru', 'w')  #alebo 'a'           repr(reťazec)
subor.write(reťazec+'\n')          reťazec.strip()
subor.close()                     while podmienky:
subor = open('meno_suboru', 'r', encoding = 'kódovanie')    prikaz1
subor.readline()                 ....prikaz2
subor.read()                     :
for riadok in subor:           with open(...) as premenná:
    print(riadok)                čítanie, alebo zápis
                                 príkazy
```

## Funkcie s návratovou hodnotou

```
def meno_funkcie(parametre):
    príkazy...
    return návratová_hodnota  #alebo hodnoty           print(meno_funkcie(parametre))
                           premenná = meno_funkcie(parametre)
                           //, %, **
```

## Práca s viacerými údajmi (zoznam)

```
zoznam = []      zoznam = [0] * 5      zoznam = [None] * 10      +, *, in
zoznam[i] = nová_hodnota                      len(zoznam)
zoznam[i], zoznam[odkial:pokial+1:krok]       for prvok in zoznam:
zoznam.append(hodnota)                        canvas['height']  #width, bg
zoznam.insert(index, hodnota)                  list()
zoznam.pop()        zoznam.pop(0)          tuple()
zoznam.index(hodnota)                        reťazec.split('oddelovač')
zoznam.count(hodnota)                        sum()   max()   min()
zoznam.sort()                                abs()
nový_zoznam = zoznam[:]
```

### Obrázky

```
premenná = tkinter.PhotoImage(file = 'meno_súboru')
canvas.create_image(x, y, image = premenná, anchor = 'nw')
obrázok.width()    obrázok.height()
canvas.place(x = 200, y = 100)
entry1.insert(0, reťazec)
lambda parametre: výraz
```

### Matematické výpočty a geometria

```
import math
math.sqrt(číslo)
abs(číslo)
round(číslo, počet_miest)
math.radians(uhol_v_stupňoch)
math.degrees(uhol_v_radiánoch)
math.cos(uhol)
math.sin(uhol)
math.pi
```

### Asociatívne polia (slovnik - dictionary)

```
slovnik = {}
slovnik = {klúč1:hodnota1, klúč2:hodnota2, ...}
slovnik[klúč] = nová_hodnota
slovnik[klúč]
slovnik.get(klúč, náhradná_hodnota)
for klúč, hodnota in slovnik:
    sorted(slovnik, key = slovnik.get, reverse = True)
klúč in slovnik
len(slovnik)
slovnik.values()
slovnik.keys()
slovnik.items()
random.shuffle()
```

### Vlastnosti útvarov nakreslených v canvase

```
canvas.itemcget(ID, 'vlastnosť')
canvas.itemconfig(ID, vlastnosť = hodnota)
canvas.coords(ID)
canvas.coords(ID, postupnosť_súradníc)
canvas.find_overlapping(x1, y1, x2, y2)
canvas.find_withtag('značka')  #napr. 'current'
canvas.addtag('nová_značka', 'withtag', ID)
canvas.type(ID)
canvas.find_all()
canvas.lift(ID)
canvas.lower(ID)
canvas.gettags(ID)
canvas.dtag(ID, 'značka')
state = 'hidden'      #'normal'
```

## Použitá literatúra

- [1] Blaho, A.: <http://python.input.sk>, 2017
- [2] <http://effbot.org/tkinterbook/>, 2017
- [3] <https://docs.python.org/3/>, 2017
- [4] Summerfield, M.: Python 3 Výukový kurz. Brno: Computer Press, 2013, ISBN 978 - 80 - 251 - 2737 - 7
- [5] Hanulová, E., Kučera, P.: Maturita z informatiky. Tvorba zadanií. Bratislava: MPC, 2007, ISBN 978 - 80 - 7164 - 440 - 8
- [6] Kučera, P.: Programujeme v Pythone (učebnica informatiky pre stredné školy). Bratislava: Peter Kučera, 2016, ISBN 978 - 80 - 972320 - 4 - 7
- [7] <http://korpus.juls.savba.sk/>, 2017



[www.programujemevpython.sk](http://www.programujemevpython.sk)