10. Udalosti v grafickej ploche

video prezentácia

udalosti

Naučíme sa v našich programoch využívať tzv. **udalosti**, ktoré vznikajú v bežiacej grafickej aplikácii, buď aktivitami používateľa (klikanie myšou, stláčanie klávesov), alebo operačného systému (tikanie časovača). Na úvod si pripomeňme, čo už vieme o grafickej ploche. Pomocou metód grafickej plochy `Canvas` (definovanej v module `tkinter`) kreslíme grafické objekty:

- `canvas.create_line()` - kreslí úsečku alebo krivku z nadväzujúcich úsečiek
- `canvas.create_oval()` - kreslí elipsu
- `canvas.create_rectangle()` - kreslí obdĺžnik
- `canvas.create_text()` - vypíše text
- `canvas.create_polygon()` - kreslí vyfarbený útvar zadaný bodmi na obvode
- `canvas.create_image()` - kreslí obrázok (prečítaný zo súboru .gif alebo .png)

Ďalšie pomocné metódy manipulujú s už nakreslenými objektmi:

- `canvas.delete()` - zruší objekt
- `canvas.move()` - posunie objekt
- `canvas.coords()` - zmení súradnice objektu
- `canvas.itemconfig()` - zmení ďalšie parametre objektu (napríklad farba, hrúbka, text, obrázok, …)

Ďalšie metódy umožňujú postupne zobrazovať vytváranú kresbu:

- `canvas.update()` - zobrazí nové zmeny v grafickej ploche
- `canvas.after()` - pozdrží beh programu o zadaný počet milisekúnd

Udalosť

Udalosťou (**event**) voláme akciu, ktorá vznikne mimo behu programu a program môže na túto situáciu reagovať. Najčastejšie sú to udalosti od pohybu a klikania myši, od stláčania klávesov, od časovača (vnútorných hodín OS), od rôznych zariadení, … V našom programe potom môžeme nastaviť, čo sa má udiať pri ktorej udalosti. Tomuto sa zvykne hovoriť **udalosťami riadené programovanie** (event-driven programming).

Používa sa na to mechanizmus obsluhy udalosti (ovládač udalosti, **event handler**), čo je funkcia v našej aplikácii, ktorá má na starosti spracovávanie príslušnej udalostí. Napríklad, ak budeme potrebovať v našom programe spracovávať udalosť kliknutie myšou do grafickej plochy, napíšeme ovládač udalosti (obyčajnú pythonovskú funkciu) a systému oznámime, aby ju zavolal vždy, keď vznikne táto udalosť. Takémuto nastaveniu (event handler) funkcie k nejakej udalosti budeme hovoriť zviazanie (**binding**).

Naučíme sa, ako v našich grafických programoch reagovať na udalosti od myši a klávesnice.

Aby grafická plocha reagovala na klikania myšou, musíme ju zviazať (**bind**) s príslušnou udalosťou (**event**).

metóda `bind()`

Táto metóda grafickej plochy slúži na zviazanie niektorej konkrétnej udalosti s nejakou funkciou, ktorá sa bude v programe starať o spracovanie tejto udalosti (event handler). Jej formát je:

```
canvas.bind(meno_udalosti, funkcia)
```

kde `meno_udalosti` je znakový reťazec s popisom udalosti (napríklad pre kliknutie tlačidlom myši) a `funkcia` je **referencia na funkciu**, ktorá by sa mala spustiť pri vzniku tejto udalosti. Táto funkcia, musí byť definovaná s práve jedným parametrom, v ktorom nám systém prezradí detaily vzniknutej udalosti.

Klikanie a ťahanie myšou

Postupne ukážeme tieto tri „myšacie“ udalosti:

- **kliknutie** (zatlačenie tlačidla myši) - reťazec `'<ButtonPress>'`
- **ťahanie** (posúvanie myšou so zatlačeným tlačidlom alebo bez zatlačeného tlačidla) - reťazec `'<Motion>'`
- **pustenie myši** - reťazec `'<ButtonRelease>'`

Klikanie myšou

Kliknutie myšou do grafickej plochy vyvolá udalosť s menom `'<ButtonPress>'`. Ukážme ako vyzerá samotné zviazanie funkcie:

```
import tkinter

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

canvas.bind('<ButtonPress>', print)

tkinter.mainloop()
```

Druhý parameter metódy `bind()` musí byť **referencia na funkciu**, ale nie na hocijakú. Musí to byť funkcia, ktorá má jeden parameter. Tu sme použili štandardnú funkciu `print` a keďže do `bind()` treba poslať referenciu na túto funkciu, nesmieme za identifikátor `print` písať zátvorky `()`.

Keď teraz tento malý testovací program spustíte, objaví sa prázdne grafické okno a program čaká, čo sa bude diať. Keďže sme grafickej ploche udalosť kliknutie myšou **zviazali** s funkciou `print()`, každé kliknutie do plochy automaticky vyvolá práve túto funkciu. Teda, pri klikaní dostávame takýto výpis (pri každom kliknutí do plochy sa vypíše jeden riadok):

```
<ButtonPress event state=Mod1 num=1 x=194 y=117>
<ButtonPress event state=Mod1 num=1 x=194 y=173>
<ButtonPress event state=Mod1 num=2 x=328 y=31>
<ButtonPress event state=Mod1 num=3 x=17 y=9>
```

Zviazanie udalosti s nejakou funkciou teda znamená, že každé vyvolanie udalosti (kliknutie tlačidlom myši do grafickej plochy) automaticky zavolá zviazanú funkciu. To, čo nám pritom vypisuje `print()`, je ten jeden parameter, ktorý `tkinter` posiela pri každom jeho zavolaní.

Vytvorme si teraz vlastnú funkciu, ktorú zviažeme s udalosťou kliknutia:

```
import tkinter

def klik(parameter):
    print('klik')

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

Vytvorili sme funkciu `klik()`, ktorá sa bude automaticky volať pri každom kliknutí do plochy. Nezabudli sme do hlavičky funkcie pridať jeden formálny parameter, inak by Python pri vzniku udalosti protestoval, že našu funkciu `klik()` chcel zavolať s jedným parametrom a my sme ho nezadeklarovali. Ak by sme tento program spustili, pri každom kliknutí do plochy by sa do textovej plochy shellu mal vypísať text `'klik'`.

Teraz k samotnému parametru v našej funkcii `klik()`: tento parameter slúži na to, aby nám `tkinter` mohol nejakým spôsobom posielať informácie o detailoch udalosti. My vieme, že funkcia `klik()` sa zavolá vždy, keď sa niekam klikne, ale nevieme, kde presne do plochy sa kliklo. Práve na toto slúži tento parameter: z neho vieme vytiahnuť, napríklad x-ovú a y-ovú súradnicu kliknutého miesta. V nasledovnom príklade vidíme, ako sa to robí. Ešte sme tento `parameter` premenovali na `event` (t.j. „udalosť“ po anglicky), aby sme lepšie rozlíšili to, že s týmto parametrom prišla udalosť. Preto tieto súradnice získame ako `event.x` a `event.y`:

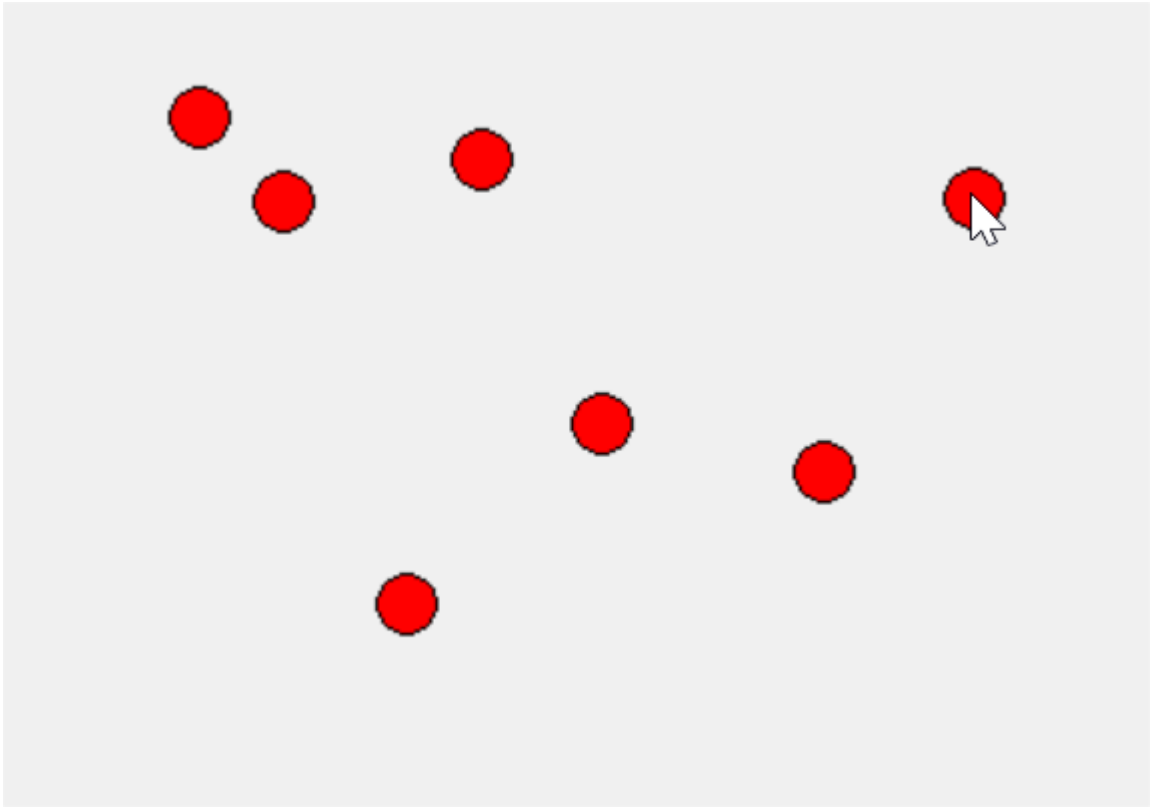
```
import tkinter

def klik(event):
    print('klik', event.x, event.y)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

V tomto programe sa pri každom kliknutí vypíšu do shellu aj súradnice kliknutého miesta.

V ďalšom príklade ukážeme, ako využijeme súradnice kliknutého bodu v ploche:

```
import tkinter

def klik(event):
    x, y = event.x, event.y
    canvas.create_oval(x-10, y-10, x+10, y+10, fill='red')

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

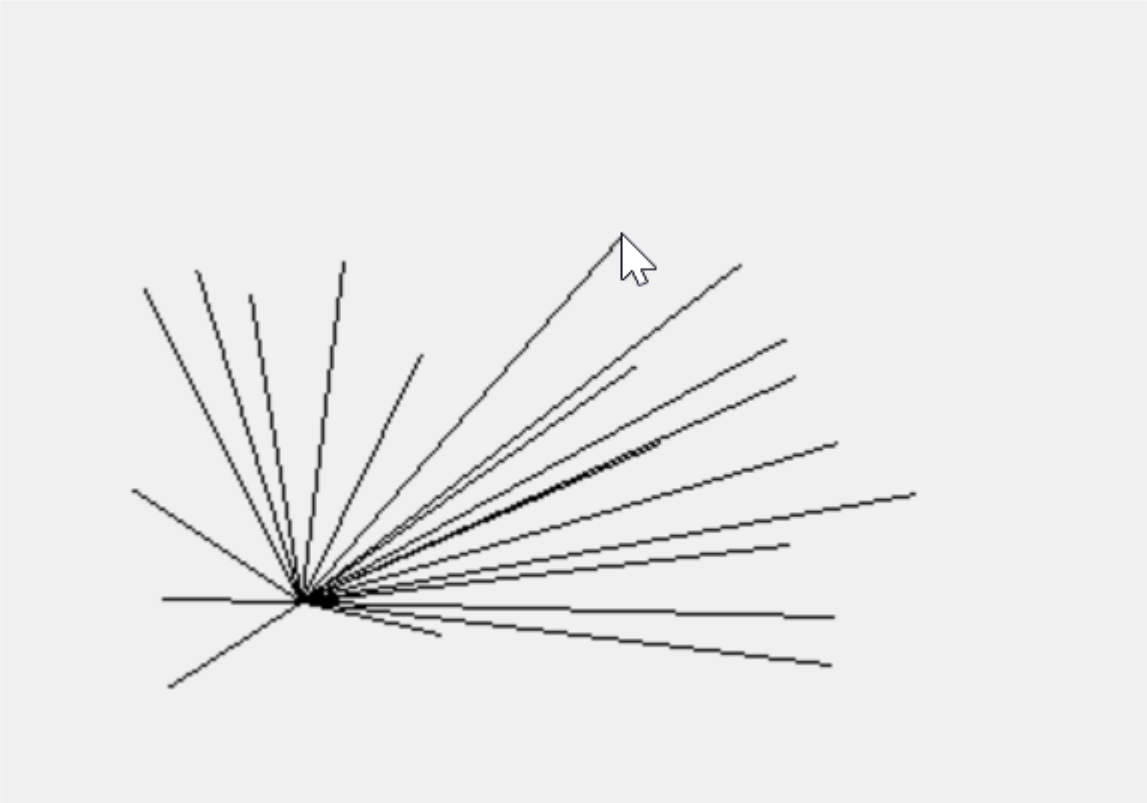
Teraz sa pri kliknutí nakreslí červený kruh a využijú sa pritom súradnice kliknutého miesta: stred kruhu je kliknuté miesto. Napríklad:



Akcia, ktorá sa vykoná pri kliknutí môže byť aj takto veľmi jednoduchá - spájanie kliknutého bodu s nejakým bodom grafickej plochy:

```
import tkinter

def klik(event):
    canvas.create_line(100, 200, event.x, event.y)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

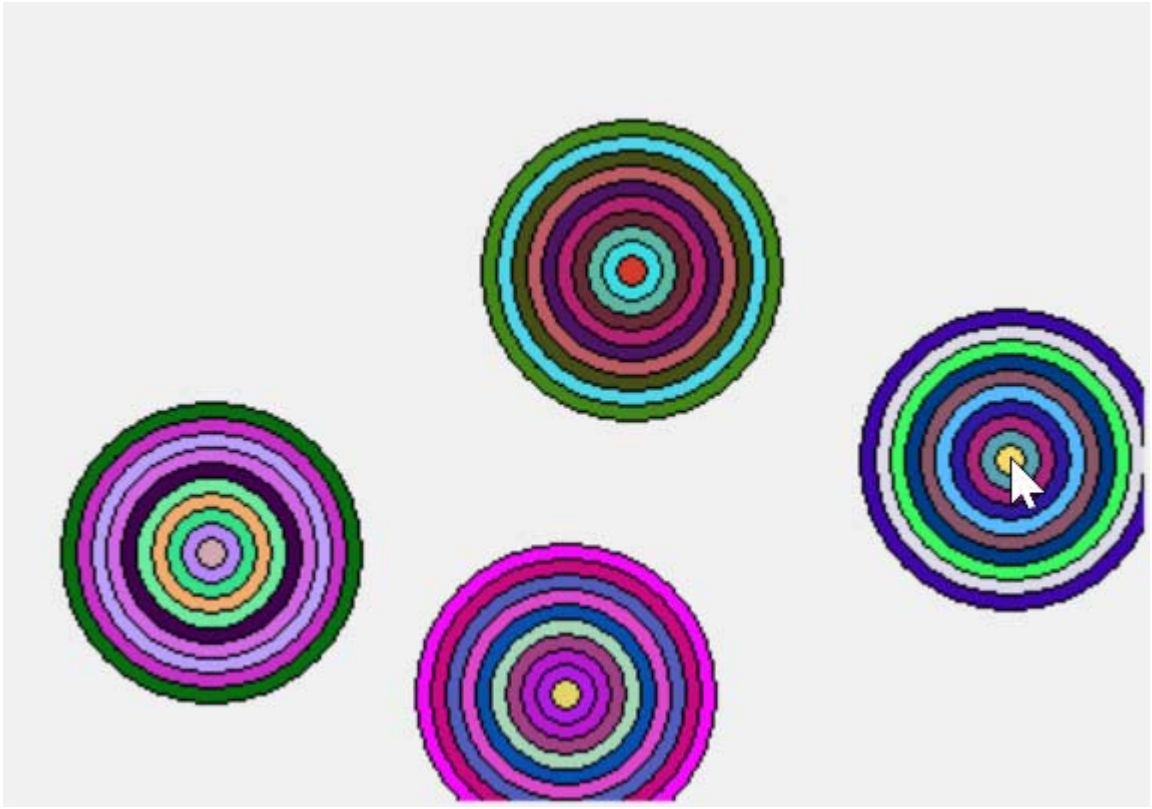
Napríklad:



Ale môžu sa nakresliť aj komplexnejšie kresby, napríklad 10 sústredných farebných kruhov:

```
import tkinter
import random

def klik(event):
    x, y = event.x, event.y
    for r in range(50, 0, -5):
        farba = f'#{random.randrange(256**3):06x}'
        canvas.create_oval(x-r, y-r, x+r, y+r, fill=farba)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

Napríklad:



Vráťme sa k príkladu, v ktorom sme kreslili malé krúžky:

```
import tkinter

def klik(event):
    x, y = event.x, event.y
    canvas.create_oval(x-5, y-5, x+5, y+5, fill='red')

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

Do tohto programu chceme pridať takéto správanie: tieto kliknuté body (červené krúžky) sa budú postupne spájať úsečkami (zrejme sa bude úsečka kresliť až od druhého kliknutia). Pridáme dve globálne premenné `xx` a `yy`, v ktorých si budeme pamätať predchádzajúci kliknutý bod. Pred prvým kliknutím sme do `xx` priradili `None`, čo bude označovať, že predchádzajúci vrchol ešte nebol kliknutý:

```
import tkinter

xx = yy = None

def klik(event):
    x, y = event.x, event.y
    canvas.create_oval(x-5, y-5, x+5, y+5, fill='red')
    if xx != None:
        canvas.create_line(xx, yy, x, y)
    xx, yy = x, y

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

Žiaľ to nefunguje: po spustení a kliknutí sa dozvieme:

```
...
File ..., line 8, in klik
    if xx != None:
UnboundLocalError: local variable 'xx' referenced before assignment
```

Problémom sú tu **globálne premenné**. Používať globálne premenné vo vnútri funkcii môžeme, len dovtedy, kým ich **nemeníme**. Priraďovací príkaz vo funkcii totiž znamená, že vytvárame novú **lokálnu premennú** (v mennom priestore funkcie `klik()`). Takže Python to v tejto funkcii pochopil takto: do premenných `xx` a `yy` sa vo vnútri funkcie priraďuje nejaká hodnota, takže obe sú lokálne premenné. Keď ale príde vykonávanie funkcie na podmienený príkaz `if  xx != None:`, Python už vie, že `xx` je lokálna premenná, ktorá nemá zatiaľ priradenú žiadnu hodnotu. A preto nám oznámil túto chybovú správu: `UnboundLocalError: local variable 'xx' referenced before assignment` (chceme používať lokálnu premennú ‚xx' skôr ako sme do nej niečo priradili).

Takže s globálnymi premennými vo funkcii sa bude musieť pracovať nejako inak. Zrejme, kým do takejto premennej nepotrebujeme vo funkcii nič priradzovať, iba ju používať, problémy nie sú. Problém nastáva vtedy, keď chceme (pomocou priraďovacieho príkazu) meniť obsah globálnej premennej.

Toto nám pomôže vyriešiť nový príkaz `global`:

príkaz `global`

príkaz má tvar:

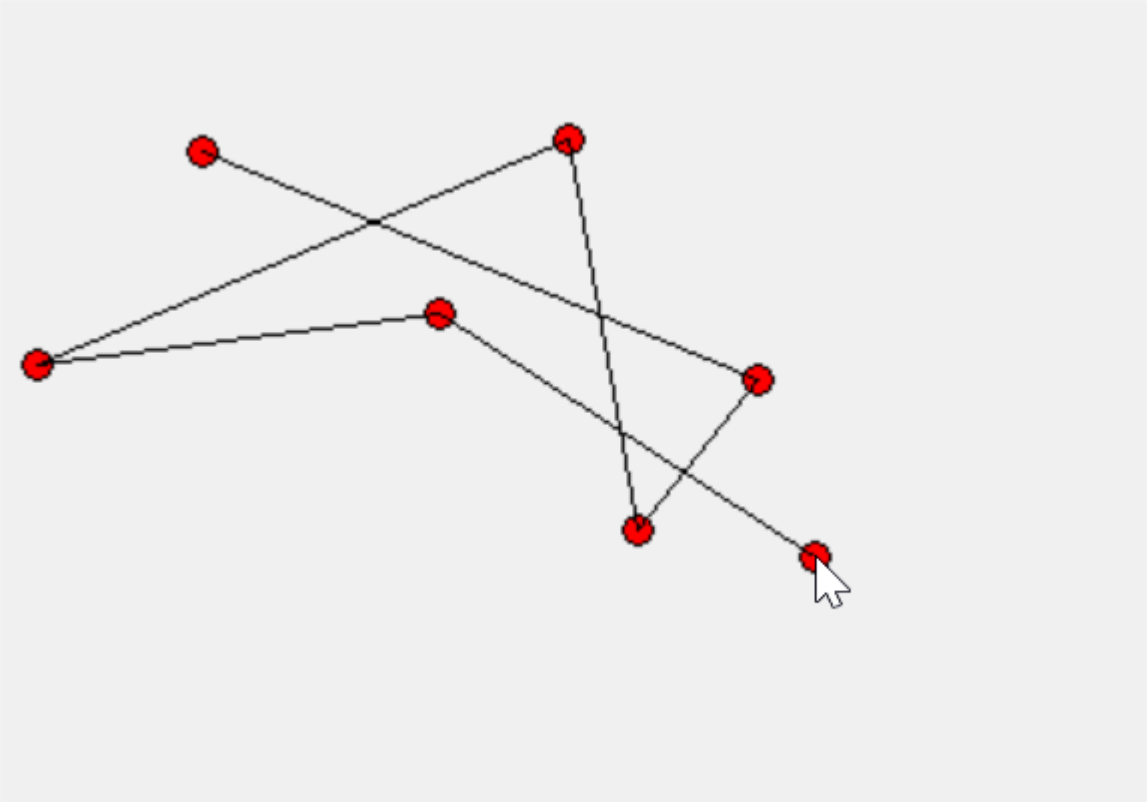
```
global premenná
global premenná, premenná, premenná, ...
```

Príkaz sa používa vo funkcii vtedy, keď v nej chceme pracovať s globálnou premennou (alebo aj s viac premennými), ale nechceme, aby ju Python vytvoril aj v lokálnom mennom priestore, ale ponechal len v globálnom.

Po doplnení tohto príkazu do predchádzajúceho príkladu všetko funguje tak, ako má:

```
import tkinter

xx = yy = None

def klik(event):
    global xx, yy
    x, y = event.x, event.y
    canvas.create_oval(x-5, y-5, x+5, y+5, fill='red')
    if xx != None:
        canvas.create_line(xx, yy, x, y)
    xx, yy = x, y

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

Po spustení dostávame takýto obrázok:



Nebezpečné

Príkaz **global** umožňuje modifikovať globálne premenné vo funkciách, teda vlastne robiť tzv. **vedľajší účinok** (*side effect*) na globálnych premenných. Toto je ale veľmi **nesprávny spôsob programovania** (*bad programming practice*) a väčšinou svedčí o programátorovi začiatočníkovi, amatérovi.

Aj testovač domácich zadaní a riešení skúšok väčšinou tento príkaz vo vašich projektoch zakazuje.

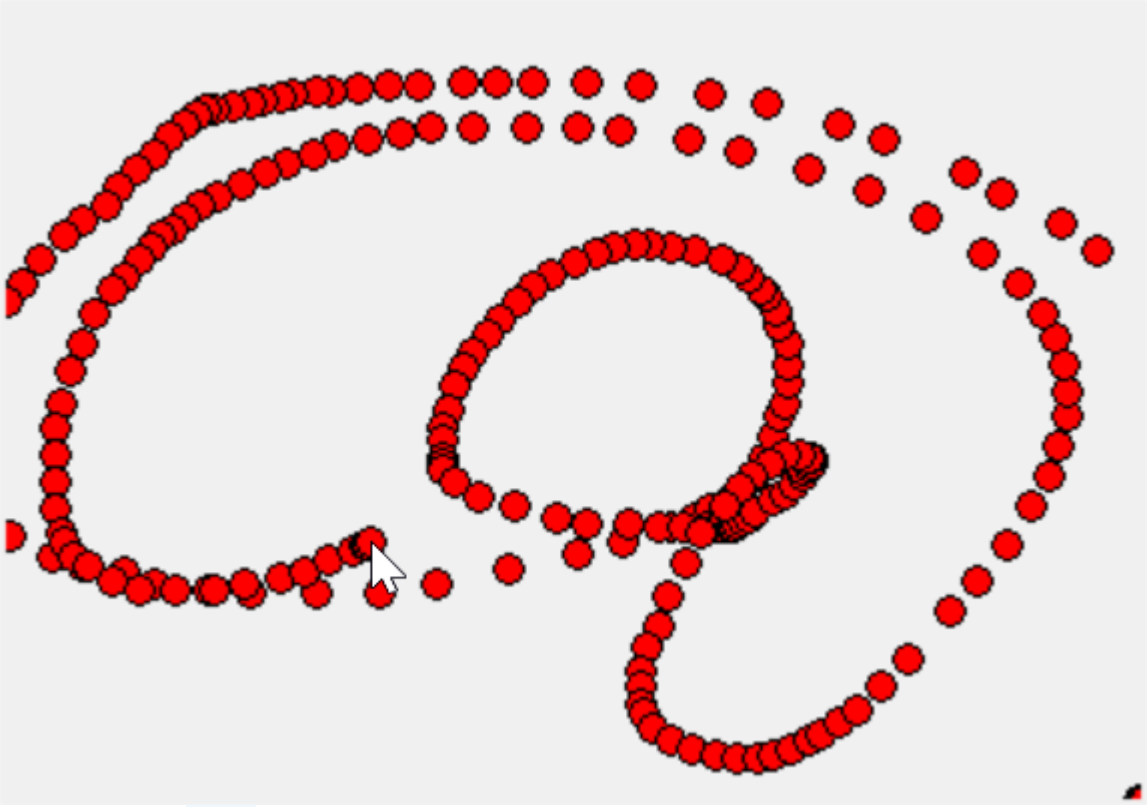
Kým sa nenaučíme, ako to obísť, budeme to používať, ale veľmi opatrne. Neskôr to využijeme veľmi výnimočne, najmä pri ladení. Správne sa takéto problémy riešia definovaním vlastných tried a použitím atribútov tried.

Ťahanie myšou

Obsluha udalosti ťahanie myšou (pohyb myši **bez zatlačeného** alebo **so zatlačeným** tlačidlom) je veľmi podobná klikaniu. Udalosť má meno `'<Motion>'`. Pozrime, čo sa zmení, keď kliknutie `'<ButtonPress>'` nahradíme ťahaním `'<Motion>'`:

```
import tkinter

def tahaj(event):
    x, y = event.x, event.y
    canvas.create_oval(x-5, y-5, x+5, y+5, fill='red')

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<Motion>', tahaj)      # '<Motion>' namiesto '<ButtonPress>'

tkinter.mainloop()
```

Funguje to veľmi dobre: pri ťahaní sa na pozícii myši kreslia červené kruhy. Pri pomalom ťahaní sú kruhy nakreslené veľmi nahusto.



Ak vo funkcii `tahaj` pri ťahaní myšou zakaždým zmažeme grafickú plochu (zrušíme všetky nakreslené objekty), v ploche zostanú nakreslené len objekty, ktoré sa kreslili až po tomto zmazaní. Napríklad:

```
import tkinter

def tahaj(event):
    x, y = event.x, event.y
    canvas.delete('all')
    canvas.create_oval(x-5, y-5, x+5, y+5, fill='red')

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<Motion>', tahaj)

tkinter.mainloop()
```

Tento program nakreslí červený krúžok, ktorý bude „prilepený“ na kurzor myši. Vyskúšajte namiesto krúžku vypisovať text:

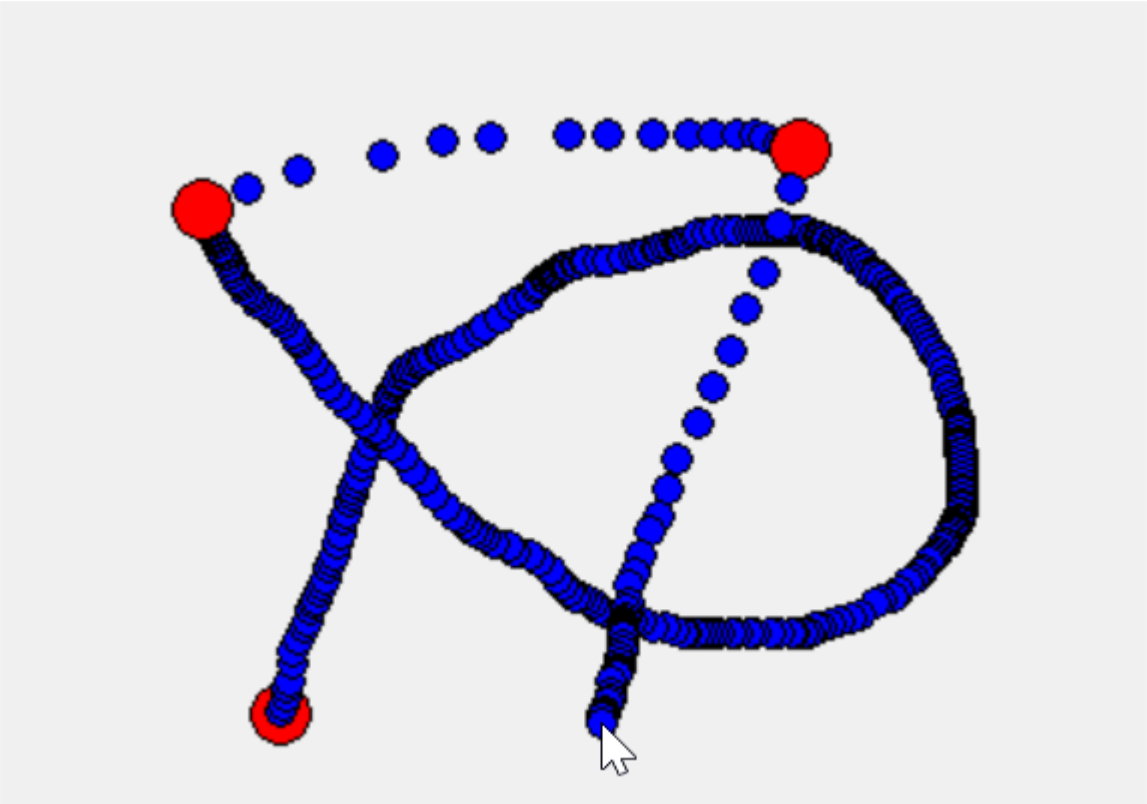
```
canvas.create_text(x, y, text=(x, y))
```

Často budeme v našich programoch spracovávať obe udalosti: kliknutie aj ťahanie. Niekedy sa o to bude starať tá istá funkcia, inokedy budú rôzne a preto je dobre ich pomenovať zodpovedajúcimi názvami, napríklad:

```
import tkinter

def klik(event):
    x, y = event.x, event.y
    canvas.create_oval(x-10, y-10, x+10, y+10, fill='red')

def tahaj(event):
    x, y = event.x, event.y
    canvas.create_oval(x-5, y-5, x+5, y+5, fill='blue')

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

```
canvas.bind('<ButtonPress>', klik)
canvas.bind('<Motion>', tahaj)

tkinter.mainloop()
```

Pri posúvaní myši sa kreslia malé modré kruhy, pri kliknutí sa nakreslí jeden väčší červený kruh:



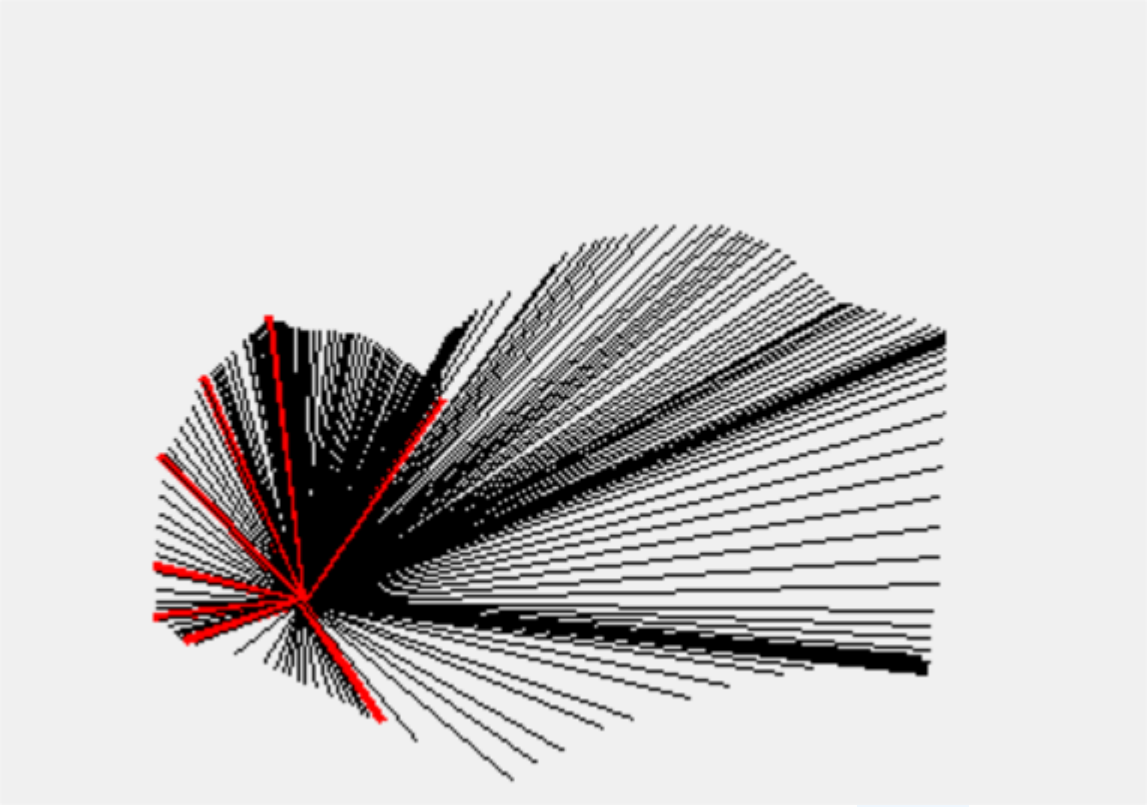
Všimnite si, že ťahanie (kreslenie modrých kruhov) funguje aj so zatlačeným tlačidlom myši.

Veľmi často budeme potrebovať, aby sa funkcia na ťahanie (event handler `tahaj`) zavolala len v prípade, že je súčasne s ťahaním zatlačené aj tlačidlo myši. Vtedy do mena udalosti `'<Motion>'` na začiatok pripíšeme `B1-`, čo bude označovať, že funkcia `tahaj` si bude všímať len ťahania so zatlačeným ľavým tlačidlom myši (`B1` je pre ľavé tlačídlo myši a `B3` pre pravé). Otestujte, ako sa bude predchádzajúci program správať, keď namiesto `canvas.bind('<Motion>', tahaj)` zapíšeme `canvas.bind('<B1-Motion>', tahaj)`. Napríklad:

Na podobnom princípe môžeme upraviť aj kreslenie lúčov z bodu `(100, 200)` do momentálnej pozície myši:

```
import tkinter

def klik(event):
    canvas.create_line(100, 200, event.x, event.y, fill='red', width=3)

def tahaj(event):
    canvas.create_line(100, 200, event.x, event.y)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)
canvas.bind('<Motion>', tahaj)

tkinter.mainloop()
```

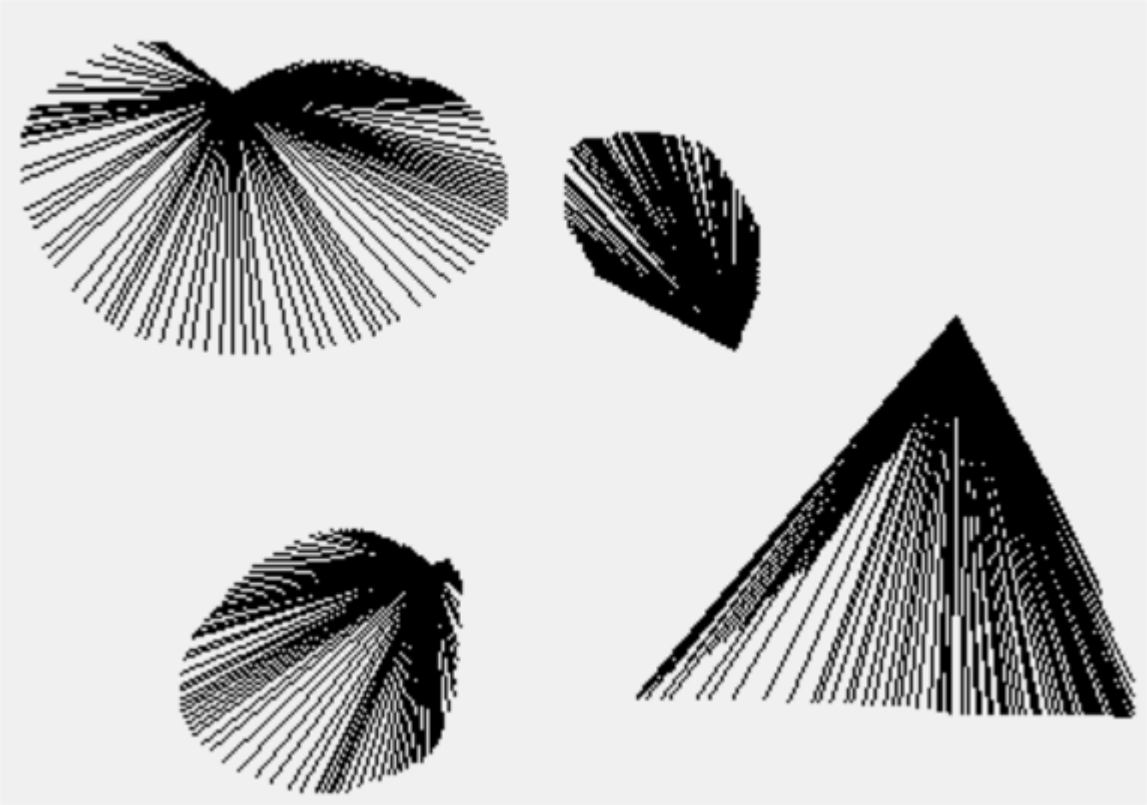
Pri posúvaní myši alebo ťahaní sa nakreslia čierne úsečky, pri každom kliknutí sa nakreslí červená úsečka:



Alebo malou zmenou kliknutím definujeme pozíciu (globálny bod `(xx, yy)`), z ktorého sa budú kresliť lúče:

```
import tkinter

def klik(event):
    global xx, yy
    xx, yy = event.x, event.y

def tahaj(event):
    canvas.create_line(xx, yy, event.x, event.y)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)
canvas.bind('<B1-Motion>', tahaj)

tkinter.mainloop()
```

Všimnite si, že sme tu použili `'<B1-Motion>'`. Zamyslite sa, prečo by tu pri `'<Motion>'` bez `B1-` vznikla chyba. Po spustení dostávame takúto kresbu:



Ďalej nadviažeme na program, v ktorom sme postupne spájali kliknuté body. Pri tomto programe sme využili nový príkaz `global`, aby sme sa dostali ku globálnym premenným. Tento nie najvhodnejší príkaz môžeme obísť, keď využijeme meniteľný (**mutable**) typ zoznam.

Budeme ťahať (ťahaním myši so zatlačeným tlačidlom) jednu dlhú lomenú čiaru, pričom si budeme ukladať súradnice prijaté z udalosti do zoznamu čísel:

```
import tkinter

zoznam = []

def klik(event):
    zoznam[:] = [event.x, event.y]

def tahaj(event):
    zoznam.extend([event.x, event.y])
    canvas.coords(ciara, zoznam)

canvas = tkinter.Canvas()
canvas.pack()
ciara = canvas.create_line(0, 0, 0, 0)
canvas.bind('<ButtonPress>', klik)
canvas.bind('<B1-Motion>', tahaj)

tkinter.mainloop()
```

Všimnite si, že v týchto dvoch funkciách používame 3 globálne premenné (okrem funkcií):

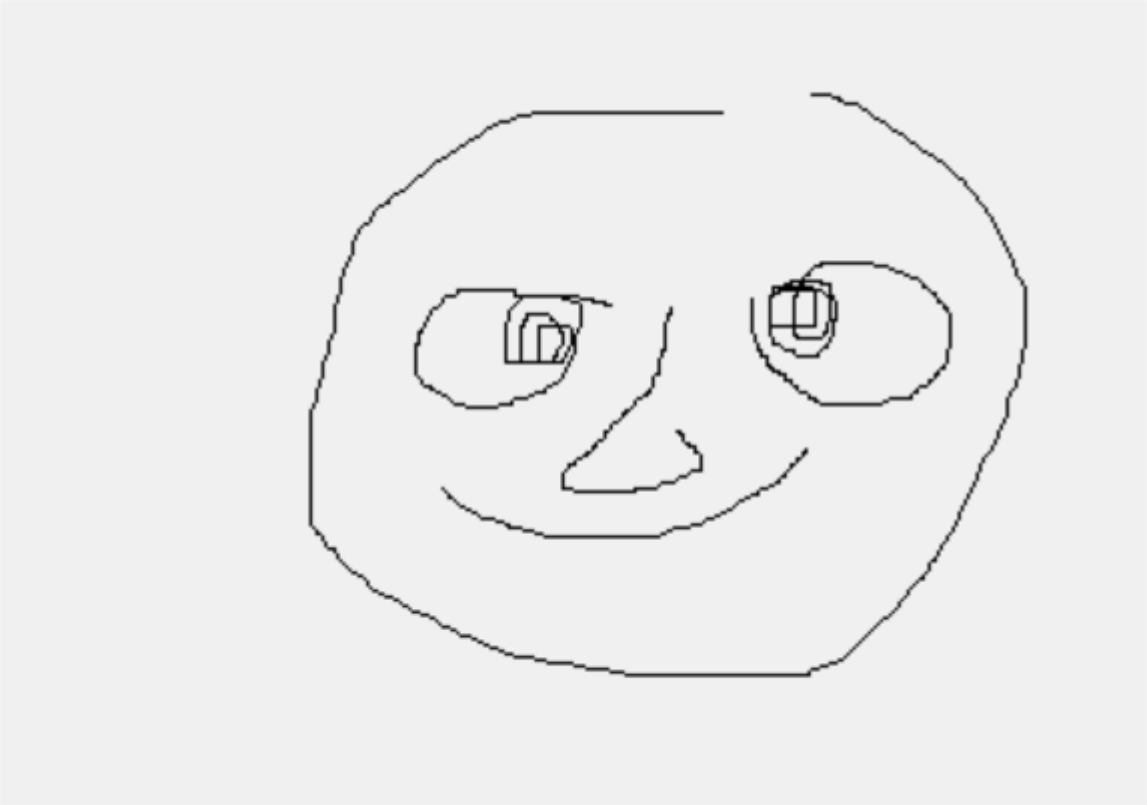
- `canvas` - referencia na grafickú plochu
- `ciara` - identifikátor objektu čiara, potrebujeme ho pre neskoršie menenie postupnosti súradníc príkazom `coords()`
- `zoznam` - zoznam súradníc je meniteľný objekt, teda môžeme meniť obsah zoznamu bez toho, aby sme do premennej `zoznam` priraďovali; v našich funkciách buď priraďujeme do rezu alebo voláme metódu `extend()` (táto prilepí nejakú postupnosť na koniec zoznamu)
    - aj modifikovanie zoznamu vo funkcii, v ktorej tento zoznam nie je parametrom funkcie, nie je najvhodnejším spôsobom programovania; aj v tomto prípade je to nevhodný **vedľajší**

účinok podobne ako príkaz `global`, zatiaľ to inak robiť nevieme, tak je to dočasne akceptovateľné

Ťahanie čiary v predchádzajúcom príklade žiaľ kreslí jedinú čiaru: každé ďalšie kliknutie a ťahanie začne kresliť novú čiaru, pričom stará čiara zmizne. Vyriešime to tak, že nová lomená čiara vznikne až pri kliknutí a ťahaním myši sa táto čiara stále predlžuje. Stará čiara pritom ostane bez zmeny:

```
import tkinter

zoznam = []

def klik(event):
    global ciara
    zoznam[:] = [event.x, event.y]
    ciara = canvas.create_line(0, 0, 0, 0)

def tahaj(event):
    zoznam.extend([event.x, event.y])
    canvas.coords(ciara, zoznam)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)
canvas.bind('<B1-Motion>', tahaj)

tkinter.mainloop()
```

Poexperimentujeme:



Malou zmenou dosiahneme veľmi zaujímavý efekt. Vyskúšajte a poštudujte:

```
import tkinter
import random

zoznam = []

def klik(event):
    global poly
    zoznam[:] = [event.x, event.y]
```

```
    farba = f'#{random.randrange(256**3):06x}'
    poly = canvas.create_polygon(0, 0, 0, 0, fill=farba)

def tahaj(event):
    zoznam.extend([event.x, event.y])
    canvas.coords(poly, zoznam)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)
canvas.bind('<B1-Motion>', tahaj)

tkinter.mainloop()
```

Udalosti od klávesnice

Aj každé zatlačenie nejakého klávesu na klávesnici môže vyvolať udalosť. Základnou univerzálnou udalosťou je `'<KeyPress>'`, ktorá sa vyvolá pri každom zatlačení nejakého klávesu. Môžeme otestovať:

```
import tkinter

def test(event):
    print(event.keysym)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind_all('<KeyPress>', test)

tkinter.mainloop()
```

Všimnite si, že sme museli zapísať `bind_all()` namiesto `bind()`. Každé zatlačenie nejakého klávesu vypíše jeho reťazcovú reprezentáciu, napríklad:

```
a
Shift_L
A
Left
Right
Up
Down
Next
Escape
Return
F1
```

Pritom každý jeden kláves môže vyvolať aj samostatnú udalosť. Ako meno udalosti treba uviesť meno klávesu (jeho reťazcovú reprezentáciu) v tvare `'<KeyPress-...>'` alebo len ako `'<...>'` (bez mena udalosti `KeyPress`) alebo väčšinou bude fungovať len samostatný znak, napríklad:

```
import tkinter

def test_vlavo(event):
    print('šípka vľavo')

def test_a(event):
    print('stlačil si kláves a')

def test_F1(event):
    print('F1')

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind_all('a', test_a)
canvas.bind_all('<Left>', test_vlavo)
canvas.bind_all('<KeyPress-F1>', test_F1)

tkinter.mainloop()
```

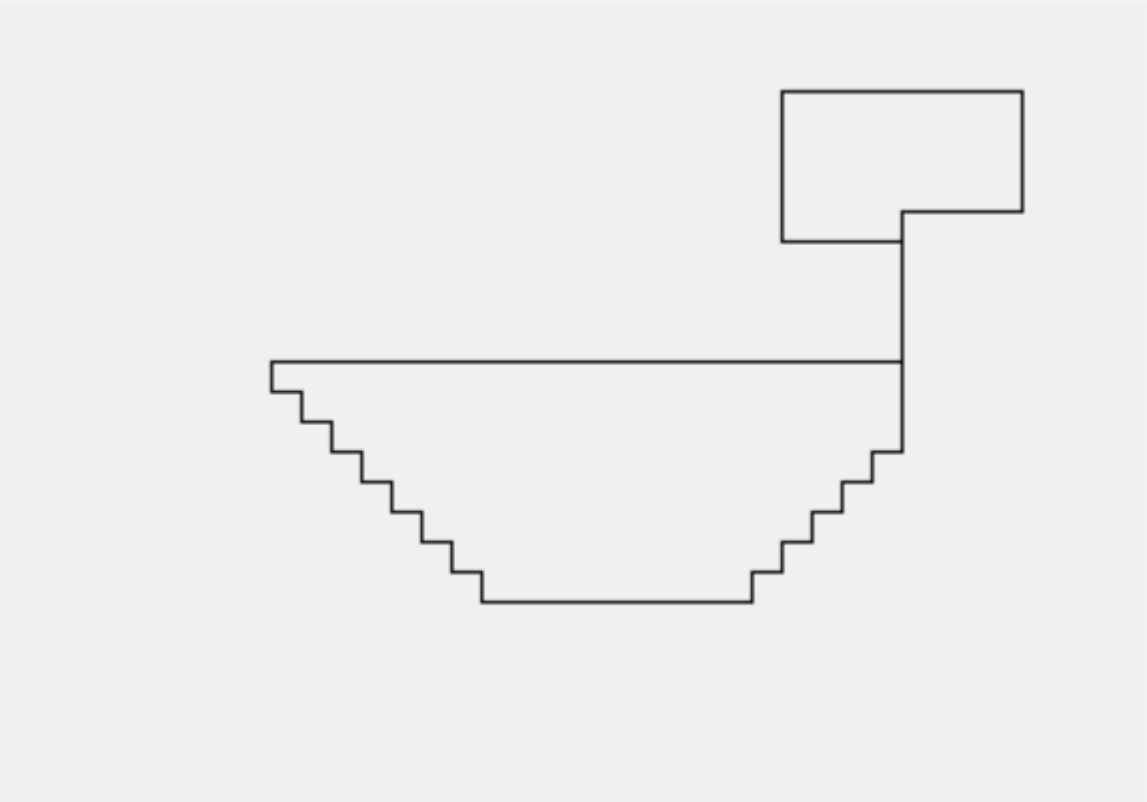
Tento program reaguje len na stláčanie klávesu `'a'`, šípky vľavo a klávesu `F1`. Všetky ostatné klávesy ignoruje.

Často sa samostatné udalosti pre jednotlivé šípky použijú podobne, ako v tomto príklade:

```
import tkinter

x, y = 200, 200
zoznam = [x, y]

def kresli(dx, dy):
    global x, y
    x += dx
    y += dy
    zoznam.extend((x, y))
    canvas.coords(ciara, zoznam)

def udalost_vlavo(event):
    kresli(-10, 0)

def udalost_vpravo(event):
    kresli(10, 0)

def udalost_hore(event):
    kresli(0, -10)

def udalost_dolu(event):
    kresli(0, 10)
```

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

ciara = canvas.create_line(0, 0, 0, 0)        # zatiaľ prázdna čiara
canvas.bind_all('<Left>', udalost_vlavo)
canvas.bind_all('<Right>', udalost_vpravo)
canvas.bind_all('<Up>', udalost_hore)
canvas.bind_all('<Down>', udalost_dolu)

tkinter.mainloop()
```

Kreslenie pomocou tohto programu môže pripomínať detskú hračku „magnetická tabuľka“:



Časovač

Pripomeňme si, ako sme sa doteraz naučili v grafickej ploche kresliť krúžky na náhodné pozície s nejakým časovým pozdržaním (napríklad 100 ms):

```
import tkinter
import random

def kresli():
    while True:
        x = random.randint(10, 370)
        y = random.randint(10, 250)
        canvas.create_oval(x-10, y-10, x+10, y+10, fill='red')
        canvas.update()
        canvas.after(100)

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

kresli()
print('hotovo')
```

```
tkinter.mainloop()
```

Použili sme tu **nekonečný cyklus** a preto sa príkaz `print('hotovo')` za volaním `kresli()` s nekonečným while-cyklom už nikdy nevykoná.

Metóda grafickej plochy `after()`, ktorá pozdrží výpočet o nejaký počet milisekúnd, je oveľa všestrannejšia: môžeme pomocou nej štartovať, tzv. **časovač**:

metóda `after()`

Metóda `after()` grafickej plochy môže mať jeden z týchto tvarov:

```
canvas.after(milisekundy)
canvas.after(milisekundy, funkcia)
```

Prvý parameter `milisekundy` už poznáme: výpočet sa pozdrží o príslušný počet milisekúnd. Lenže, ak je metóda zavolaná aj s druhým parametrom `funkcia`, výpočet sa v skutočnosti nepozdrží, ale pozdrží sa vyvolanie zadanej funkcie (parameter `funkcia` musí byť **referencia na funkciu**, teda väčšinou bez okrúhlych zátvoriek). Táto vyvolaná funkcia musí byť definovaná bez parametrov.

S týmto druhým parametrom metóda `after()` **naplánuje** (niekedy do budúcnosti) spustenie nejakej funkcie a pritom výpočet pokračuje normálne ďalej na ďalšom príkaze za `after()` (bez pozdržania).

Tomuto mechanizmu hovoríme **časovač** (naplánovanie spustenia nejakej akcie), po anglicky **timer**. Najčastejšie sa používa takto:

```
def casovac():
    # príkazy
    canvas.after(cas, casovac)
```

V tomto prípade funkcia naplánuje spustenie samej seba po nejakom čase. Môžete si to predstaviť tak, že v počítači tikajú nejaké hodiny s udanou frekvenciou v milisekundách a pri každom tiknutí sa vykonajú príkazy v tele funkcie.

Najprv jednoduchý test:

```
import tkinter

def casovac():
    print('tik')
    canvas.after(1000, casovac)

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

casovac()

tkinter.mainloop()
```

Časovač každú sekundu vypíše do textovej plochy reťazec `'tik'`.

Predchádzajúci program s náhodnými červenými krúžkami teraz prepíšeme s použitím časovača (`canvas.update()` treba z funkcie časovača vyhodiť):

```
import tkinter
import random

def kresli():
    x = random.randint(10, 370)
    y = random.randint(10, 250)
```

```
    canvas.create_oval(x-10, y-10, x+10, y+10, fill='red')
    #canvas.update()               # v časovači by sa nemalo volať
    canvas.after(100, kresli)

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

kresli()             # naštartovanie časovača
print('hotovo')

tkinter.mainloop()
```

Po spustení funkcie `kresli()` (tá nakreslí jeden kruh a zavolá `after()`, t. j. naplánuje ďalšie kreslenie) sa pokračuje ďalším príkazom, t. j. vypíše sa `print('hotovo')`. Sem by sme mohli zapísať ďalšie pythonovské príkazy.

V nasledovnom príklade sme pridali zviazanie s klavesom <Enter>, pomocou ktorého sa zmaže celá grafická plocha (pomocou `canvas.delete('all')`):

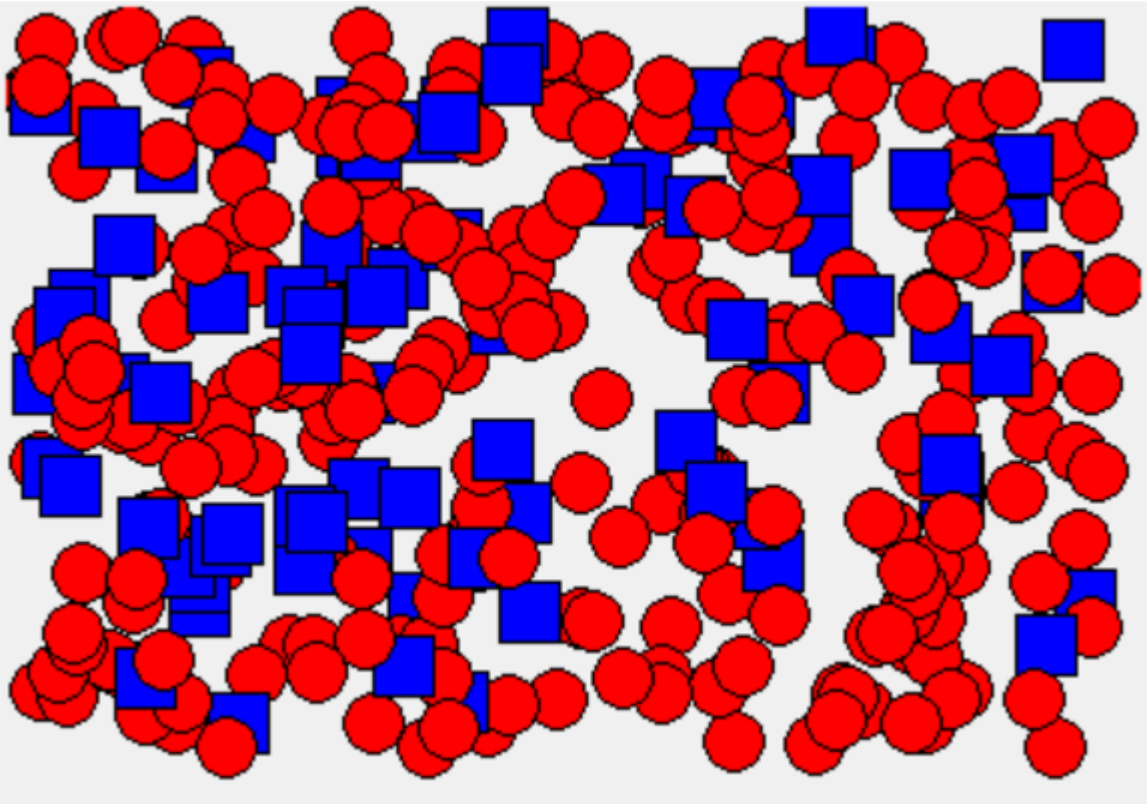
```
import tkinter
import random

def kresli():
    x = random.randint(10, 370)
    y = random.randint(10, 250)
    canvas.create_oval(x-10, y-10, x+10, y+10, fill='red')
    canvas.after(100, kresli)

def zmaz(event):
    canvas.delete('all')

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

kresli()             # naštartovanie časovača
canvas.bind_all('<Return>', zmaz)

tkinter.mainloop()
```

Keďže počas behu časovača môže program vykonávať ďalšie akcie, môže spustiť hoci aj ďalší časovač. Zapíšme:

```
import tkinter
import random

def kresli():
    x = random.randint(10, 370)
    y = random.randint(10, 250)
    canvas.create_oval(x-10, y-10, x+10, y+10, fill='red')
    canvas.after(100, kresli)

def kresli1():
    x = random.randint(10, 370)
    y = random.randint(10, 250)
    canvas.create_rectangle(x-10, y-10, x+10, y+10, fill='blue')
    canvas.after(300, kresli1)

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

kresli()
kresli1()
```

```
tkinter.mainloop()
```

Program teraz spustí oba časovače: kreslia sa červené krúžky a modré štvorčeky. Keďže druhý časovač má svoj interval `300` milisekúnd, teda „tiká“ 3-krát pomalšie ako prvý, kreslí 3-krát menej modrých štvorčekov ako prvý časovač červených krúžkov, napríklad:



Zastavovanie časovača

Na zastavenie časovača nemáme žiaden príkaz. Časovač môžeme zastaviť len tak, že on sám v svojom tele na konci nezavolá metódu `canvas.after()` a tým aj skončí. Upravíme predchádzajúci príklad tak, že zadefinujme dve globálne premenné, ktoré budú slúžiť pre oba časovače na zastavovanie. Aby sme mohli tieto časovače zastavovať, resp. opäť rozbiehať, pridáme dve funkcie na spracovanie udalosti od klávesov ``<a>`` a `<b>`. Každá z týchto funkcií sa bude starať o svoj časovač:

```
import tkinter
import random

bezi = bezi1 = True

def kresli():
    x = random.randint(10, 370)
    y = random.randint(10, 250)
    canvas.create_oval(x-10, y-10, x+10, y+10, fill='red')
    if bezi:
        canvas.after(100, kresli)

def kresli1():
    x = random.randint(10, 370)
    y = random.randint(10, 250)
    canvas.create_rectangle(x-10, y-10, x+10, y+10, fill='blue')
    if bezi1:
        canvas.after(300, kresli1)
```

```
def prvy(event):
    global bezi
    bezi = not bezi
    if bezi:
        kresli()

def druhy(event):
    global bezi1
    bezi1 = not bezi1
    if bezi1:
        kresli1()

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind_all('<a>', prvy)
canvas.bind_all('<b>', druhy)

kresli()
kresli1()

tkinter.mainloop()
```

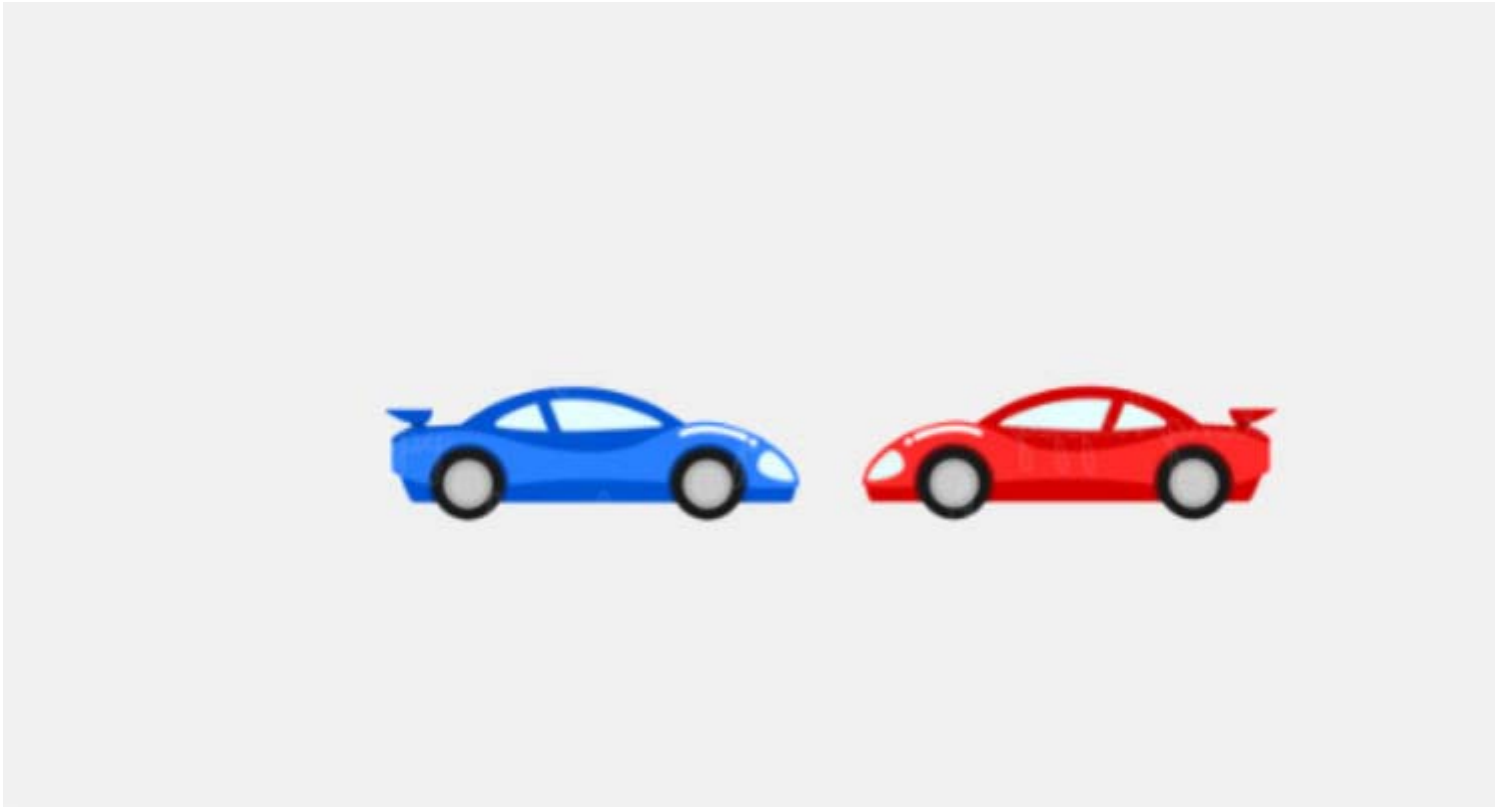
Teraz bežia oba časovače, ale stačí priradiť stláčať klávesy `<a>` alebo `<b>` a časovače sa budú zastavovať alebo opäť spúšťať.

Globálne premenné môžeme využiť aj na iné účely: môžeme nimi meniť „parametre“ bežiacich príkazov. Napríklad farbu krúžkov, ale aj interval tikania časovača, teda zrýchlovať alebo spomaľovať bežiace časovače.

Ďalšia ukážka bude hýbať dvoma obrázkami autíčok (napríklad auto1.png a auto2.png) rôznou rýchlosťou:

```
import tkinter

def pohyb():
    canvas.move(auto1, 4, 0)
    canvas.move(auto2, -5, 0)
    canvas.after(30, pohyb)

canvas = tkinter.Canvas(width=600)
canvas.pack()

obr_auto1 = tkinter.PhotoImage(file='auto1.png')
obr_auto2 = tkinter.PhotoImage(file='auto2.png')

auto1 = canvas.create_image(0, 150, image=obr_auto1)
auto2 = canvas.create_image(600, 150, image=obr_auto2)

pohyb()

tkinter.mainloop()
```

Program rozbehne dve autíčka proti sebe, ale nekontroluje, či sa zrazia:



Aby sa autá nerozbehli už pri štarte programu, ale až po kliknutí do plochy, zapíšeme:

```
import tkinter

def start(event):
    canvas.coords(auto1, 0, 150)
    canvas.coords(auto2, 600, 150)
    pohyb()

def pohyb():
    canvas.move(auto1, 4, 0)
    canvas.move(auto2, -5, 0)
    canvas.after(30, pohyb)

canvas = tkinter.Canvas(width=600)
canvas.pack()

obr_auto1 = tkinter.PhotoImage(file='auto1.png')
obr_auto2 = tkinter.PhotoImage(file='auto2.png')

auto1 = canvas.create_image(0, 150, image=obr_auto1)
auto2 = canvas.create_image(600, 150, image=obr_auto2)

canvas.bind('<ButtonPress>', start)

tkinter.mainloop()
```

Teraz chceme pridať kontrolu, či sa už obe autá zrazili. Ak by sme zastavili časovač (museli by sme doprogramovať), mohli by sme vypísať informácie o polohe oboch áut. Keďže si nikde neuchovávame ich momentálnu pozíciu, využijeme metódu `coords()`, ktorá okrem zmeny súradníc grafického objektu, dokáže zistiť momentálne jeho súradnice. Napríklad, by sme mohli dostať takéto výpisy:

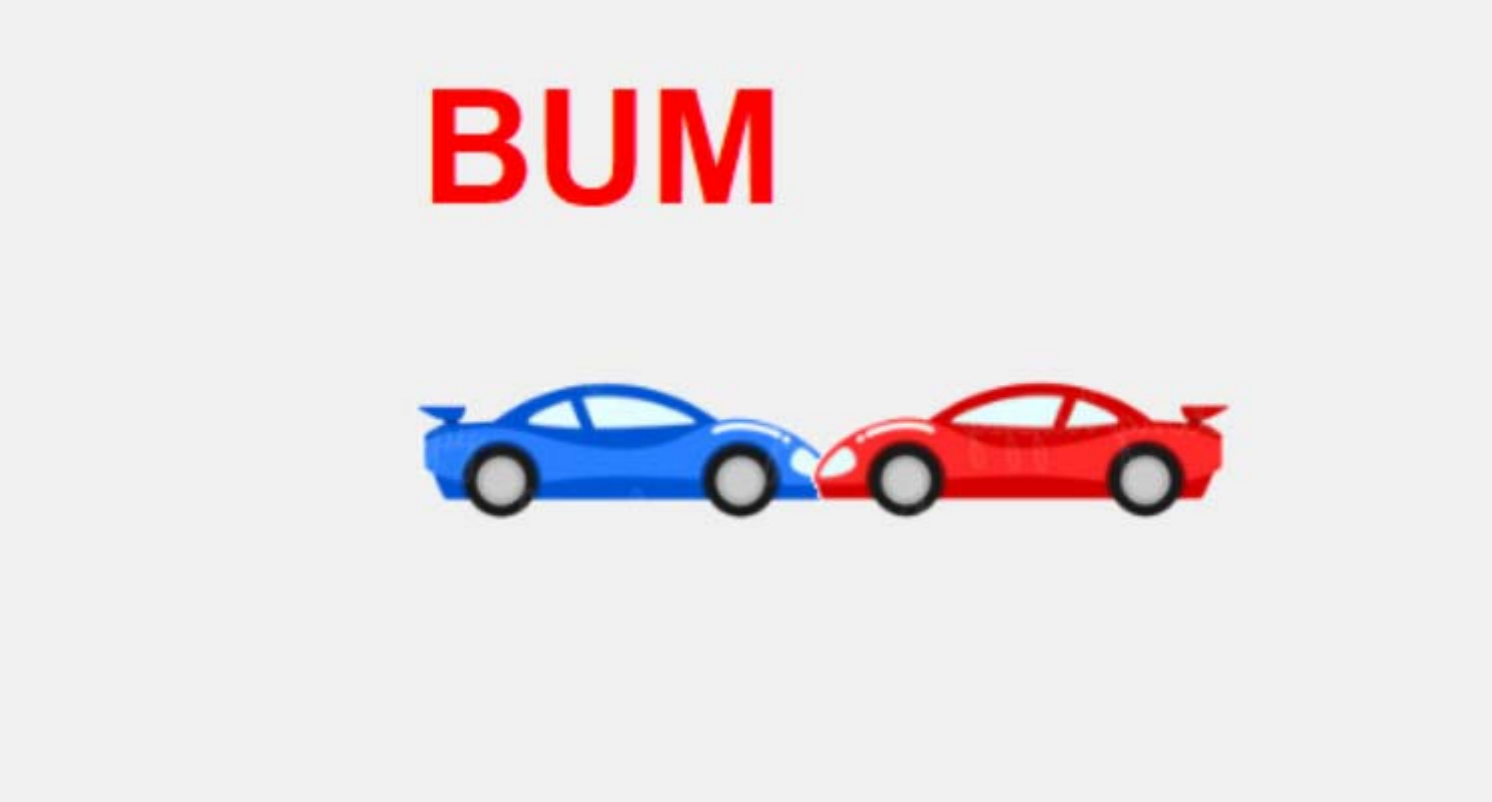
```
print('auto1 =', canvas.coords(auto1))
print('auto2 =', canvas.coords(auto2))
```

```
auto1 = [164.0, 150.0]
auto2 = [395.0, 150.0]
```

Vidíme, že táto funkcia nám vracia polohu autíčka (súradnice stredu obrázka), preto môžeme zastaviť časovač testovaním x-ových súradníc autíčok:

```
import tkinter

def start(event):
    canvas.coords(auto1, 0, 150)
    canvas.coords(auto2, 600, 150)
    pohyb()

def pohyb():
    canvas.move(auto1, 4, 0)
    canvas.move(auto2, -5, 0)
    x_auto1 = canvas.coords(auto1)[0]
    x_auto2 = canvas.coords(auto2)[0]
    if x_auto1 > x_auto2 - 140:
        canvas.create_text(200, 50, text='BUM', fill='red', font='arial 40 bold')
    else:
        canvas.after(30, pohyb)

canvas = tkinter.Canvas(width=600)
canvas.pack()

obr_auto1 = tkinter.PhotoImage(file='auto1.png')
obr_auto2 = tkinter.PhotoImage(file='auto2.png')

auto1 = canvas.create_image(0, 150, image=obr_auto1)
auto2 = canvas.create_image(600, 150, image=obr_auto2)

canvas.bind('<ButtonPress>', start)

tkinter.mainloop()
```

Pri stretnutí autíčok dostávame:



Hoci tento program funguje dobre, má niekoľko malých nedostatkov:

- ak počas pohybu autíčok (beží časovač `pohyb()`) znovu klikneme do plochy (vyvoláme udalosť `start()`), autíčka skočia do svojich štartových pozícií a znovu sa vyvolá časovač `pohyb()`;

teraz to ale vyzerá, že autíčka idú dvojnásobnou rýchlosťou - totiž teraz bežia naraz dva časovače `pohyb()` aj `pohyb()`, ktoré oba pohnú oboma autíčkami - hoci je to zaujímavé, budeme sa snažiť tomuto zabrániť
- ak autíčka do seba nabúrajú, vypíše sa text `'BUM'`, ktorý tam bude svietiť aj po opätovnom naštartovaní autíčok

Vylepšíme funkciu `start()` takto:

- keďže táto štartuje časovač `pohyb()`, zablokujeme opätovné kliknutie tým, že zrušíme zviazanie udalosti klik s funkciou `start()`
- ak svieti text `'BUM'`, tak ho vymažeme

Využijeme nový príkaz `unbind()`, pomocou ktorého vieme **rozviazať** nejakú konkrétnu udalosť s funkciou:

metóda `unbind()`

Metóda zruší zviazanie príslušnej udalosti:

```
canvas.unbind(meno_udalosti)
```

V časovači (vo funkcii) `pohyb()` pri výpise správy `'BUM'`, keďže zastavujeme časovač, opätovne zviažeme kliknutie do plochy s udalosťou `start()`:

```
import tkinter

text = None

def start(event):
    canvas.unbind('<ButtonPress>')              # zruší klikaciu udalosť
    canvas.coords(auto1, 0, 150)
    canvas.coords(auto2, 600, 150)
    canvas.delete(text)
    pohyb()

def pohyb():
    global text
    canvas.move(auto1, 4, 0)
    canvas.move(auto2, -5, 0)
    x_auto1 = canvas.coords(auto1)[0]
    x_auto2 = canvas.coords(auto2)[0]
    if x_auto1 > x_auto2 - 140:
        text = canvas.create_text(200, 50, text='BUM', fill='red', font='arial 40 bold')
        canvas.bind('<ButtonPress>', start)     # obnoví klikaciu udalosť
    else:
        canvas.after(30, pohyb)

canvas = tkinter.Canvas(width=600)
canvas.pack()

obr_auto1 = tkinter.PhotoImage(file='auto1.png')
obr_auto2 = tkinter.PhotoImage(file='auto2.png')

auto1 = canvas.create_image(0, 150, image=obr_auto1)
auto2 = canvas.create_image(600, 150, image=obr_auto2)

canvas.bind('<ButtonPress>', start)

tkinter.mainloop()
```

Zhrnutie udalostí od myši

Udalosť `'<ButtonPress>'` reprezentuje kliknutie ľubovoľným tlačidlom myši. Väčšinou má každá myš tri tlačidlá: ľavé, stredné a pravé. Priamo v názve udalosti môžeme určiť, aby sa udalosť vyvolala len pri konkrétnom tlačidle. Vtedy bude názov udalosti takýto:

- `'<ButtonPress-1>'` pre zatlačenie ľavého tlačidla
- `'<ButtonPress-2>'` pre zatlačenie stredného tlačidla
- `'<ButtonPress-3>'` pre zatlačenie pravého tlačidla

Môžeme zapísať napríklad:

```
import tkinter

def klik_lavy(event):
    canvas.create_text(event.x, event.y, text=1, font='Arial 30', fill='blue')

def klik_stredny(event):
    canvas.create_text(event.x, event.y, text=2, font='Arial 30', fill='green')

def klik_pravy(event):
    canvas.create_text(event.x, event.y, text=3, font='Arial 30', fill='red')

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress-1>', klik_lavy)
canvas.bind('<ButtonPress-2>', klik_stredny)
canvas.bind('<ButtonPress-3>', klik_pravy)

tkinter.mainloop()
```

Každá funkcia sa stará o kliknutie svojho tlačidla. V parametri `event` okrem súradníc `event.x` a `event.y` dostávame aj poradové číslo tlačidla `event.num`, ktorým sme práve klikli. Predchádzajúci program môžeme zapísať aj takto:

```
import tkinter

def klik(event):
    farba = ['blue', 'green', 'red'][event.num-1]
    canvas.create_text(event.x, event.y, text=event.num, font='Arial 30', fill=farba)

canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)

tkinter.mainloop()
```

Je to na programátorovi, ktorú z týchto možností preferuje.

Meno udalosti `'<ButtonPress-1>'` existuje aj v skrátenej forme a to buď `'<Button-1>'` alebo dokonca `'<1>'` (zrejme to funguje aj s `2` aj `3`). Opäť je na programátorovi, ktorý zápis preferuje a pritom neznižuje čitateľnosť kódu.

Už poznáme meno udalosti pre ťahanie myši `'<Motion>'`. Táto pomenovaná udalosť zavolá príslušnú funkciu so zatlačeným aj bez zatlačeného tlačidla. Poznáme aj variant mena udalosti `'<B1-Motion>'`, vďaka čomu spracovávame len ťahanie so zatlačeným ľavým tlačidlom myši. Podobne ako pri klikaní môžeme špecifikovať zatlačené tlačidlo číslom od 1 do 3, bude to fungovať aj pri ťahaní, preto:

- `'<B1-Motion>'` ťahanie so zatlačeným ľavým tlačidlom
- `'<B2-Motion>'` ťahanie so zatlačeným stredným tlačidlom
- `'<B3-Motion>'` ťahanie so zatlačeným pravým tlačidlom

Neskôr uvidíme využitie udalosti aj od pustenia tlačidla myši `'<ButtonRelease>'`. Aj táto udalosť funguje pre varianty s konkrétnym tlačidlom myši, napríklad `'<ButtonRelease-1>'` spracováva len pustenie ľavého tlačidla myši.

Cvičenia

L.I.S.T.

- riešenia **aspoň 12 úloh** odovzdaj na úlohový server https://list.fmph.uniba.sk/
- pozri si Riešenie úloh 10. cvičenia

1. Napíš program, ktorý zabezpečí ťahanie myši: pri ťahaní so zatlačeným ľavým tlačidlom sa kreslia vodorovné úsečky (z kliknutej pozície dĺžky 100). Pravým klikom sa obrazovka zmaže. V programe zabezpeč zviazanie týchto dvoch ovládačov:

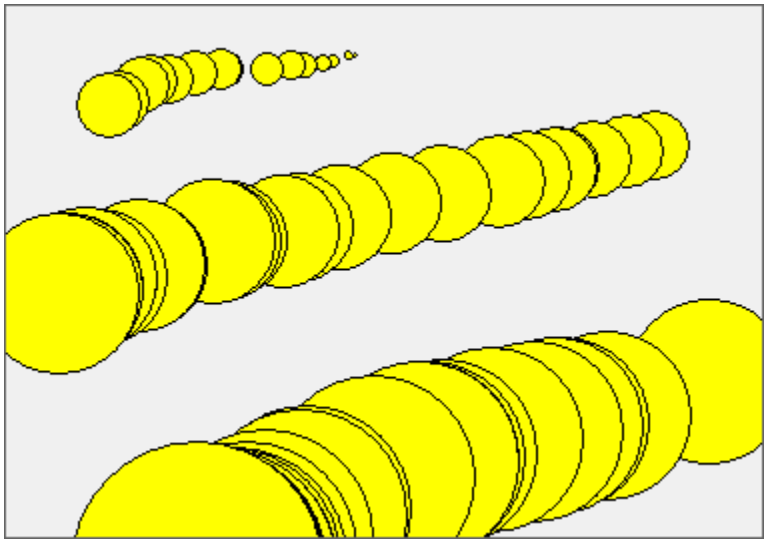
```
2. canvas.bind('<B1-Motion>', kresli)
3. canvas.bind('<ButtonPress-3>', zmaz)
```

   Po spustení a ťahaní môžeš dostať, napríklad:

2. Program počas ťahania myši zabezpečí kreslenie žltých krúžkov, prvý s polomerom `1`, každý ďalší je o `1` väčší. Pravým klikom sa obrazovka zmaže a nastaví sa kreslenie od najmenšieho krúžku (s polomerom `1`). V programe zabezpeč zviazanie dvoch ovládačov:

```
3. canvas.bind('<B1-Motion>', kresli)
4. canvas.bind('<ButtonPress-3>', zmaz)
```

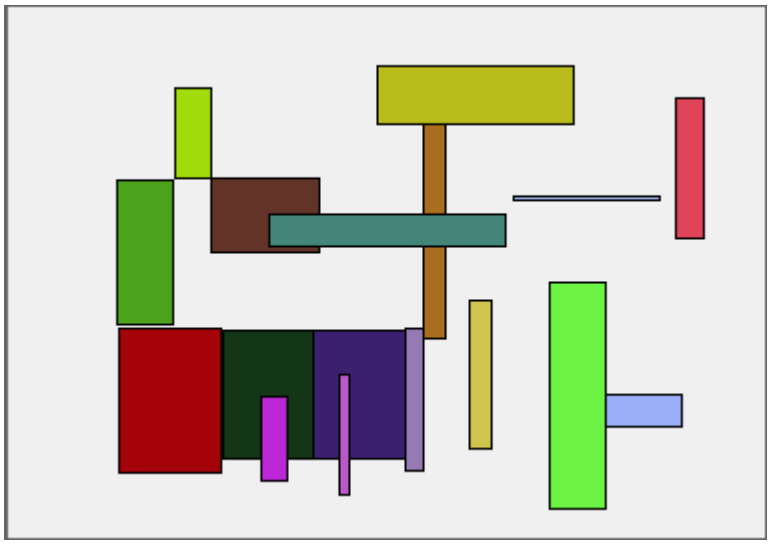
   Po spustení a ťahaní môžeš dostať, napríklad:



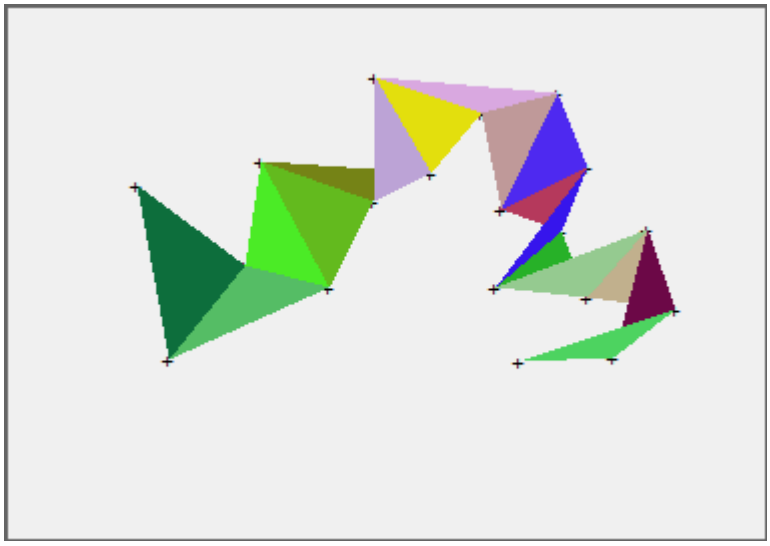
3. Program zabezpečí klikanie myšou: prvé kliknutie si zapamätá súradnice, druhé kliknutie nakreslí obdĺžnik (`create_rectangle`), v ktorom jeden vrchol je zapamätaný a druhý vrchol je práve kliknutý. Obdĺžnik je zafarbený náhodnou farbou. Toto sa opakuje pri ďalších klikaniach.

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

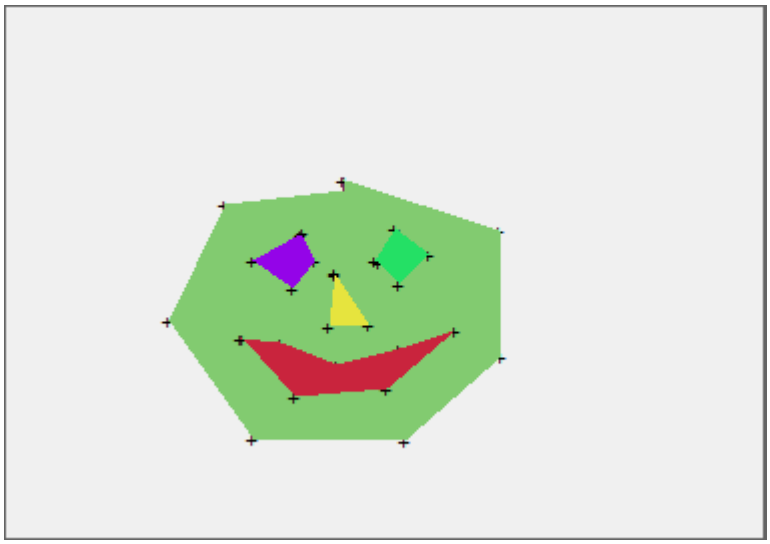
Napríklad:



4. Podobne ako v (3) úlohe: pri každom kliknutí sa na danej pozícii nakreslí `'+'` (`create_text`), pritom pri prvých dvoch kliknutiach sa tieto body zapamätajú. Tretie z týchto troch kliknutí nakreslí náhodne zafarbený trojuholník (`create_polygon`). Zoznam zapamätaných bodov má momentálne 3 vrcholy. Teraz sa z tohto zapamätaného zoznamu vyhodí prvý prvok a pokračuje sa ďalej s dvomi vrcholmi: ďalšie kliknutie pridá do zoznamu 3. vrchol, nakreslí trojuholník a prvý vrchol sa opäť vyhodí. Môžeš dostať, napríklad:



5. Podobne ako v (4) úlohe: klikanie do plochy kreslí `'+'` a zapamätáva tieto body. Ak mám aspoň 3 zapamätané body a pritom prvý a posledný majú vzdialenosť menšiu ako 5, ukončí sa zapamätávanie bodov, nakreslí sa z nich náhodne zafarbený polygon a všetko sa začne od začiatku. Môžeš dostať, napríklad:



6. Vylepši riešenie (5) úlohy: pri kreslení polygónu sa automaticky všetky `'+'` z plochy vymažú (`canvas.delete(...)`). Každému nakreslenému `'+'` môžeš pridať rovnaký `tag` a potom ich naraz zrušíš. Pozri, napríklad 3. prednášku.

7. Najprv zadefinuj štyri premenné (napríklad `x1, y1, x2, y2 = 100, 50, 200, 100`) a pomocou nich nakresli farebný obdĺžnik. Potom program pri každom kliknutí, ale len do vnútra obdĺžnika, mu zmení farbu výplne. Každé kliknutie do vnútra obdĺžnika cyklicky strieda jednu zo 4 farieb, napríklad `farby = ['blue', 'red', 'green', 'yellow']`. Nepoužívaj `global`.

8. Napíš program, ktorý pri každom kliknutí do grafickej plochy vypíše na toto miesto poradové číslo kliknutia, teda postupne čísla 1, 2, 3, 4, … Nepoužívaj `global` (môžeš využiť globálnu premennú s nejakým zoznamom).

9. Predstav si, že celá grafická plocha je štvorcová sieť, ktorej štvorčeky majú veľkosť 50x50. Napíš program, ktorý pri každom kliknutí do grafickej plochy nakreslí náhodne zafarbený príslušný štvorček tejto siete (pomocou `create_rectangle()`). Nepoužívaj `global`.

10. Najprv nakresli kruh so stredom (`x0`, `y0`) a polomerom `r` (napríklad pre `r, x0, y0 = 120, 150, 130`). Potom každé kliknutie do vnútra kruhu zmení farbu výplne na odtieň šedej - čím bližšie do stredu tým tmavšie (v strede kruhu čierne), ku okraju svetlejšie (na obvode biele). Zrejme pri kliknutí vypočítaš vzdialenosť od stredu kruhu a toto číslo potom prepočítaš na celé číslo od `0` do `255` (napríklad `f = int(255 * vzd / r)`). Z tohto čísla vyrobíš šedý odtieň pre farbu kruhu (zrejme `rgb(f, f, f)`). Nepoužívaj `global`. Namiesto klikania môžeš otestovať ťahanie myšou.

11. Napíš program, ktorý bude robiť efekt spreja: ťahanie myšou so zatlačeným ľavým tlačidlom nakreslí 20 farebných bodiek (farba podľa globálnej premennej, napríklad `farba = 'blue'`) na náhodných pozíciách. Tieto náhodné bodky budú mať od kliknutého miesta takúto vzdialenosť: x-ová súradnica bude z intervalu `<x-30, x+30>` a y-ová z `<y-30, y+30>`. Najlepšie je ich kresliť ako kruhy s polomerom `2` bez obrysu (`width=0`). Do programu pridaj aj spracovanie ľavého kliknutia myši: vtedy sa nastaví premenná `farba` na náhodnú farbu. Vďaka tomuto každé ťahanie myšou bude sprejovať inou farbou.

12. Gumená úsečka: kliknutie naštartuje vytváranie úsečky (najprv prvý aj druhý bod úsečky je samotný kliknutý bod, teda jej veľkosť je 0), ťahanie aktualizuje jej druhý bod (použi metódu `canvas.coords()`). Každé ďalšie kliknutie a ťahanie vytvára ďalšiu úsečku.

13. Gumený obdĺžnik: kliknutie naštartuje vytváranie obdĺžnika (jeden vrchol je kliknutý bod a veľkosť je zatiaľ 0x0), ťahanie aktualizuje jeho veľkosť, t.j. protiľahlý vrchol.
    - prvé kliknutie nastaví náhodnú farbu výplne tohto obdĺžnika
    - každé ďalšie kliknutie a ťahanie vytvára ďalší obdĺžnik
    - otestuj tento program s „gumenou elipsou“:- namiesto `create_rectangle()` daj `create_oval()`

14. V ploche sa nachádza jeden červený štvorček veľkosti 50x50. Keď klikneme do jeho vnútra (počíta sa aj obvod), môžeme ho ťahať, teda posúvať po ploche pomocou pohybov myši (inak ťahanie s kliknutím mimo štvorček nerobí nič).
    - daj pozor, aby aj malé posunutie myši počas ťahania neurobilo jeho neúmerne veľký skok (môžeme ho chytiť a ťahať napríklad aj za ľubovoľný vrchol)
    - pri kliknutí do vnútra štvorčeka si môžeš niekde zapamätať posunutie kliknutého bodu od ľavého horného rohu štvorčeka a toto posunutie využiješ pri prekreslení štvorčeka na nových pozíciách (pomocou `canvas.coords()` alebo `canvas.move()`)

15. Riešenie predchádzajúcej (14) úlohy uprav tak, aby fungoval aj pre 2 rôzne veľké štvorce: jeden červený veľkosti 50x50, druhý modrý veľkosti 100x100. Vedel by si tento program upraviť tak, aby fungoval pre ľubovoľný počet rôzne veľkých štvorcov v ploche?

16. Stláčaním malých a veľkých písmen abecedy sa tieto vypisujú nejakým väčším fontom vedľa seba (Napríklad `'arial 30'`). Využi jeden grafický objekt pre text (`create_text`) a tomuto budeš pri stláčaní písmen pridávať vypisovaný text (pomocou `canvas.itemconfig()`). Program by mal akceptovať aj stláčanie medzery a Enteru (do textu vloží `'\n'`). Použi metódu `bind_all('<KeyPress>', ...)` pričom vo viazanej funkcii pracuj s hodnotou `event.char`.

17. Do grafickej plochy nakresli kružnicu (napríklad pre `r, x0, y0 = 100, 150, 120`). Potom naprogramuj časovač, ktorý bude rovnomerne posúvať červenú bodku (kruh s polomerom 5) na

obvode tejto kružnice (po každom tiknutí časovača posunie jeho pozíciu na kružnici o uhol `10` stupňov). Posúvanie kruhu budeš robiť pomocou `canvas.coords()`.

18. Do riešenia predchádzajúcej (17) úlohy pridaj ďalší pohybujúci sa modrý kruh, ktorého uhol posunu bude iný, napríklad `15` stupňov.
    - experimentuj s rôznymi rýchlosťami pohybu oboch kruhov po kružnici (prípadne sa jeden z nich pohybuje opačným smerom)
    - ak by sa farebné kruhy pohybovali po rôznych kružniciach, mohli by sme simulovať pohyb planét okolo slnka

19. Nechaj bežať na obrazovke veľké digitálne hodiny: čas je zobrazený v tvare `'9:22:34.5'` a mení sa každú 0.1 sekundy. Použi jeden textový objekt (`create_text()`), ktorému pomocou `itemconfig()` meníš zobrazovanú hodnotu.
    - pri štarte programu môžeš využiť `h, m, s = time.localtime()[3:6]`

20. Naprogramuj takúto hru na postreh:
    - každých `interval` milisekúnd sa farebný kruh s polomerom `r` presunie na náhodnú pozíciu v ploche
    - keď klikneme do plochy a trafíme do vnútra kruhu, ku nášmu skóre sa pripočíta 10
    - keď klikneme do plochy, ale netrafíme do kruhu, skóre sa zníži o 1
    - aktuálne skóre sa vypisuje niekde v rohu obrazovky (ako grafický objekt `create_text()`)
    - `interval` a `r` sú nejaké globálne premenné, napríklad s hodnotami `1000` a `20`; po niekoľkých klikaniach sa tieto hodnoty môžu automaticky meniť

21. Na mieste kliknutia myšou sa nakreslí kruh s polomerom 50 a s náhodnou farbou výplne, ďalších 50 tiknutí časovača sa jej polomer zmenšuje o `1` s časovým intervalom `0.1` sekundy. Keď bude polomer `0`, časovač túto kružnicu (grafický objekt kruh) zruší. Zabezpeč, aby aj viac kliknutí tesne za sebou na rôzne miesta plochy vytvorilo viac kruhov a aby sa všetky postupne zmenšovali o `1`.

5. Týždenný projekt

L.I.S.T.

- riešenie odovzdaj na úlohový server https://list.fmph.uniba.sk/

Napíš pythonovský skript, ktorý bude definovať tieto funkcie a jednu globálnu premennú:

```
tab = []

def citaj(meno_suboru):
    ...

def pocet_vyskytov(slovo):
    ...
    return 0

def najcastejsie():
    ...
    return [...]

def s_poctom(n):
    ...
    return [...]
```

```
def najdlhsie():
    ...
    return [...]

def najkratsie():
    ...
    return [...]

def s_dlzkou(n):
    ...
    return [...]
```

Funkcia `citaj()` dostáva ako parameter meno textového súboru. Funkcia tento súbor prečíta a do globálnej premennej `tab` si nejako zaznačí počet výskytov každého slova v súbore (najlepšie pre každé slovo ako dvojicu: reťazec a počet výskytov). Premenná `tab` musí byť typu zoznam (`list`). Slová vo vstupnom súbore sú oddelené aspoň jednou medzerou alebo znakom konca riadkov. Nepoužívaj ďalšie globálne premenné ani ďalšie funkcie.

Naledovné funkcie, ktoré treba naprogramovať, spracujú informácie v premennej `tab` a vrátia zoznam nejakých slov (možno aj prázdny):

- funkcia `pocet_vyskytov(slovo)` vráti počet, koľkokrát sa dané slovo objavilo v texte
- funkcia `najcastejsie()` vráti nticu slov (`tuple`), ktoré boli v texte najčastejšie - buď je to ntica s jedným slovom, alebo je to ntica viacerých slov, ak mali rovnaký počet výskytov (zrejme maximálny)
- funkcia `s_poctom(n)` vráti nticu slov (`tuple`), ktoré sa v texte vyskytli presne `n`-krát - táto ntica môže byť aj prázdna, keď v súbore nebolo ani jedno slovo s týmto počtom výskytov
- funkcia `najdlhsie()` vráti nticu slov (`tuple`), ktoré boli v texte najdlhšie - buď je to ntica s jedným slovom, alebo je to ntica viacerých slov, ak majú rovnakú maximálnu dĺžku
- funkcia `najkratsie()` vráti nticu slov (`tuple`), ktoré boli v texte najkratšie - buď je to ntica s jedným slovom, alebo je to ntica viacerých slov, ak majú rovnakú minimálnu dĺžku
- funkcia `s_dlzkou(n)` vráti nticu slov (`tuple`), ktoré majú dĺžku presne `n` - táto ntica môže byť aj prázdna, keď v súbore nebolo ani jedno slovo s touto dĺžkou

Žiadna z týchto funkcií nič nevypisuje, len vráti (`return`) nejakú nticu slov, respektíve celé číslo.
Funkcia `citaj()` nič nevracia.

Napríklad `"text1.txt"` sa skladá z týchto riadkov:

```
i see it i deduce it how do i know that you have been getting yourself very wet lately and that
you have a most clumsy and careless servant girl
my dear holmes said i this is too much you would certainly have been burned had you lived a few
centuries ago it is true that i had a country walk on thursday and came home in a dreadful mess
but as i have changed my clothes i can't imagine how you deduce it as to mary jane she is incor
rigible and my wife has given her notice but there again i fail to see how you work it out
he chuckled to himself and rubbed his long nervous hands together
it is simplicity itself said he my eyes tell me that on the inside of your left shoe just where
the firelight strikes it the leather is scored by six almost parallel cuts obviously they have
been caused by someone who has very carelessly scraped round the edges of the sole in order to
remove crusted mud from it hence you see my double deduction that you had been out in vile weat
her and that you had a particularly malignant bootslitting specimen of the london slavey as to
your practice if a gentleman walks into my rooms smelling of iodoform with a black mark of nitr
ate of silver upon his right forefinger and a bulge on the right side of his tophat to show whe
re he has secreted his stethoscope i must be dull indeed if i do not pronounce him to be an act
ive member of the medical profession
```

Tento súbor obsahuje 271 slov, pričom niektoré sa v texte opakujú. Po spustení takéhoto testu:

```
if __name__ == '__main__':
    citaj('text1.txt')
```

```
    print('pocet vyskytov "the":', pocet_vyskytov('the'))
    print('najcastejsie:', najcastejsie())
    print('najdlhsie:', najdlhsie())
    print('najkratsie:', najkratsie())
    print('len s poctom 5:', s_poctom(5))
    print('len s poctom 10:', s_poctom(10))
    print('len s dlzkou 10:', s_dlzkou(10))
    print('pocet roznych slov =', len(tab))
```

program vypíše:

```
pocet vyskytov "the": 8
najcastejsie: ('i',)
najdlhsie: ('incorrigible', 'particularly', 'bootslitting')
najkratsie: ('i', 'a')
len s poctom 5: ('have', 'is')
len s poctom 10: ('i',)
len s dlzkou 10: ('simplicity', 'carelessly', 'forefinger', 'profession')
pocet roznych slov = 161
```

Vypisované ntice môžu byť aj v inom poradí.

Prvý riadok tohto testu `if __name__ == '__main__':` označuje, že telo tohto príkazu (keď podmienka je pravdivá) sa vykoná jedine v prípade, že program spúšťaš pomocou **Run** (**F5**). Ak tvoj program bude spúšťať testovač, tieto príkazy sa nevykonajú (testovač neobľubuje príkaz `print`).

Tvoj odovzdaný program s menom `riesenie.py` musí začínať tromi riadkami komentárov:

```
# 5. zadanie: najcastejsie
# autor: Janko Hraško
# datum: 1.11.2021
```

Projekt `riesenie.py` odovzdaj na úlohový server https://list.fmph.uniba.sk/ najneskôr do 23:00 **5. novembra**, kde ho môžeš nechať otestovať. Testovač bude spúšťať tvoje funkcie s rôznymi textovými súbormi, ktoré si môžeš stiahnuť z L.I.S.T.u. Odovzdať projekt aj ho testovať môžeš ľubovoľný počet krát. Môžeš zaň získať **5 bodov**.