Události v grafické ploše

vznikají v běžící grafické aplikaci, buď aktivitami uživatele (klepání myší, mačkání kláves), nebo operačního systému (tikání časovače).

Pomocí metod grafické plochy *Canvas* (definované v modulu *tkinter*) kreslíme grafické objekty:

- *canvas.create_line()* - kreslí úsečku nebo křivku z navazujících úseček
- *canvas.create_oval()* - kreslí elipsu
- *canvas.create_rectangle()* - kreslí obdélník
- *canvas.create_text()* - vypíše text
- *canvas.create_polygon()* - kreslí vybarvený útvar zadaný body na obvodu
- *canvas.create_image()* - kreslí obrázek (přečtený ze souboru .gif nebo .png)

Další pomocné metody manipulují s již nakreslenými objekty:

- *canvas.delete()* - zruší objekt
- *canvas.move()* - posune objekt
- *canvas.coords()* - změní souřadnice objektu
- *canvas.itemconfig()* - změní další parametry objektu (například barva, tloušťka, text, obrázek, …)

Další metody umožňují postupně zobrazovat vytvářenou kresbu:

- *canvas.update()* - zobrazí nové změny v grafické ploše
- *canvas.after()* - pozdrží běh programu o zadaný počet milisekund

Událost

Událostí (*event*) voláme akci, která vznikne mimo běh programu a program může na tuto situaci reagovat. Nejčastěji jsou to události od pohybu a klikání myši, od mačkání kláves, od časovače (vnitřních hodin OS), od různých zařízení, … V našem programu pak můžeme nastavit, co se má udát při které události. Tomuto se zvykne ***říkat událostmi řízené programování (event-driven programming).***

Používá se k tomu mechanismus obsluhy události (***ovladač události, event handler***), což je funkce v naší aplikaci, která má na starosti zpracovávání příslušné události. Například, budeme-li potřebovat v našem programu zpracovávat událost kliknutí myší do grafické plochy, napíšeme ovladač události (obyčejnou pythonovskou funkci) a systému oznámíme, aby ji zavolal vždy, když vznikne tato událost. Takovému nastavení (***event handler***) funkce k nějaké události budeme říkat ***svázání (binding).***

metoda bind()

Aby grafická plocha reagovala na klikání myší, musíme ji ***svázat (bind)*** s příslušnou ***událostí (event).*** Tato metoda grafické plochy slouží ke svázání některé konkrétní události s nějakou funkcí, která se bude v programu starat o zpracování této ***události (event handler***).

Její formát je: ***canvas.bind(jméno_události, funkce)***

kde ***jméno_události*** je znakový řetězec s popisem události (například pro kliknutí tlačítkem myši) a funkce je reference na funkci, která by se měla spustit při vzniku této události. Tato funkce musí být definována s právě jedním parametrem, ve kterém nám systém prozradí detaily vzniklé události.

Klikání a tahání myší

• kliknutí (zatlačení tlačítka myši) - řetězec ***'<ButtonPress>'***
• tahání (posouvání myší se zatlačeným tlačítkem nebo bez zatlačeného tlačítka) - řetězec ***'<Motion>'***
• puštění myši - řetězec ***'<ButtonRelease>'***

Klikání myší

Klepnutí myší do grafické plochy vyvolá událost se jménem ***'<ButtonPress>'.***

```
import tkinter
def klik(event):
   print('klik', event.x, event.y)
canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)
tkinter.mainloop()
```

Druhý parametr metody ***bind()*** musí být reference na funkci, která má jeden parametr. Zde jsme použili standardní funkci ***print*** a jelikož do ***bind()*** je třeba poslat referenci na tuto funkci, nesmíme za identifikátor ***print*** psát závorky ***()***.

Svázání události s nějakou funkcí tedy znamená, že každé vyvolání události (kliknutí tlačítkem myši do grafické plochy) automaticky zavolá svázanou funkci.

Parametr slouží k tomu, aby nám *tkinter* mohl nějakým způsobem posílat informace o detailech události. Z něj umíme vytáhnout, například *x-ovou a y-ovou* souřadnici kliknutého místa.
Je dobré tento ***parametr*** přejmenovat ***event*** (tj. „událost“ anglicky), abychom lépe rozlišili to, že s tímto parametrem přišla událost.

Souřadnice získáme jako event.x a event.y:

V dalším příkladu ukážeme, jak využijeme souřadnice kliknutého bodu v ploše:

```
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()
```

Nyní se při kliknutí nakreslí červený kruh a využijí se přitom souřadnice kliknutého místa: střed kruhu je kliknuté místo.
Akce, která se provede při kliknutí může být i takto velmi jednoduchá - spojování kliknutého bodu s nějakým bodem grafické plochy, ale mohou se nakreslit i komplexnější kresby (například 10 soustředných barevných kruhů).

Globální proměnné uvnitř funkce

Používat globální proměnné uvnitř funkce můžeme, jen dokud je neměníme.
Přiřazovací příkaz ve funkci totiž znamená, že vytváříme novou lokální proměnnou.
Problém nastává tehdy, když chceme (pomocí přiřazovacího příkazu) měnit obsah globální proměnné. To nám pomůže vyřešit nový příkaz ***global***.

příkaz global

příkaz má tvar:

global proměnná
global proměnná, proměnná, proměnná, ...

Příkaz se používá ve funkci tehdy, když v ní chceme pracovat s globální proměnnou (nebo s více proměnnými), ale nechceme, aby ji Python vytvořil v lokálním jmenném prostoru, ale ponechal jen v globálním.

Nebezpečné!
Příkaz global umožňuje modifikovat globální proměnné ve funkcích, tedy vlastně dělat tzv. vedlejší účinek (side effect) na globálních proměnných. Toto je ale velmi nesprávný způsob programování (bad programming practice) a většinou svědčí o programátorovi začátečníkovi, amatérovi.

Dokud se nenaučíme, jak to obejít, budeme to používat, ale velmi opatrně. Později toho využijeme velmi výjimečně, zejména při ladění. Správně se takové problémy řeší definováním vlastních tříd a použitím atributů tříd.

Tento ne nejvhodnější příkaz můžeme obejít, když využijeme ***měnitelný (mutable) typ seznam.***

```
import tkinter
seznam = []
def klik(event):
   global ciara
   seznam[:] = [event.x, event.y]
   čára = canvas.create_line(0, 0, 0, 0)
def tahaj(event):
   seznam.extend([event.x, event.y])
   canvas.coords(čára, seznam)
canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)
canvas.bind('<B1-Motion>', táhni)
tkinter.mainloop()
```

Všimněte si, že v těchto dvou funkcích používáme 3 globální proměnné (kromě funkcí):
• **canvas** - reference na grafickou plochu
• **čára** - identifikátor objektu čára, potřebujeme ho pro pozdější měnění posloupnosti souřadnic příkazem ***coords()***
• **seznam** - seznam souřadnic je měnitelný objekt, tedy můžeme měnit obsah seznamu, aniž bychom do proměnné seznam přiřazovali; v našich funkcích buď přiřazujeme do řezu nebo voláme metodu ***extend()*** (tato přilepí nějakou posloupnost na konec seznamu)

Modifikování seznamu ve funkci, ve které tento seznam není parametrem funkce, není vhodný způsobem programování; zatím to jinak dělat nevíme, tak je to dočasně akceptovatelné

Tahání myší

Obsluha události tažení myší (pohyb myši bez zatlačeného nebo se zatlačeným tlačítkem) je velmi podobná klikání. Událost má jméno ***'<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>', táhni) # '<Motion>' místo '<ButtonPress>'
tkinter.mainloop()
```

Při tažení se na pozici myši kreslí červené kruhy.

Pokud ve funkci ***táhni*** při tažení myší pokaždé smažeme grafickou plochu (zrušíme všechny nakreslené objekty), v ploše zůstanou nakresleny pouze objekty, které se kreslily až po tomto smazání. Tento program nakreslí červený kroužek, který bude „přilepen“ na kurzor myši:

```
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>', táhni)
tkinter.mainloop()
```

Často budeme v našich programech zpracovávat obě události: kliknutí i tahání. Někdy se o to bude starat tatáž funkce, jindy budou různé a proto je dobře je pojmenovat odpovídajícími názvy, například:

```
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>', táhni)
tkinter.mainloop()
```

Při posouvání myši se kreslí malé modré kruhy, při klepnutí se nakreslí jeden větší červený kruh.

Velmi často budeme potřebovat, aby se funkce pro tahání (***event handler tahej***) zavolala jen v případě, že je současně s tažením zatlačeno i tlačítko myši. Tehdy do jména události ***'<Motion>'*** na začátek připíšeme ***B1-,*** což bude označovat, že funkce tahej si bude všímat jen tahání se zatlačeným levým tlačítkem myši (***B1*** je pro levé tlačítko myši a ***B3*** pro pravé).

Události od klávesnice

I každé zatlačení nějaké klávesy na klávesnici může vyvolat událost.
Základní univerzální událostí je ***'<KeyPress>',*** která se vyvolá při každém zatlačení nějaké klávesy.

```
import tkinter
test def(event):
   print(event.keysym)
canvas = tkinter.Canvas()
canvas.pack()
canvas.bind_all('<KeyPress>', test)
tkinter.mainloop()
```

Všimněte si, že jsme museli zapsat ***bind_all()*** místo ***bind()***.
Každé zatlačení nějaké klávesy vypíše jeho řetězcovou reprezentaci.

Přitom každá jedna klávesa může vyvolat i samostatnou událost. Jako jméno události je třeba uvést jméno klávesy (jeho řetězcovou reprezentaci) ve tvaru ***'<KeyPress-...>'*** nebo jen jako ***'<...>'*** *(bez jména události KeyPress)* nebo většinou bude fungovat jen samostatný znak.

Často se jednotlivé události pro jednotlivé šipky použijí podobně, jako v tomto příkladu:

```
import tkinter
x, y = 200, 200
seznam = [x, y]
def kresli(dx, dy):
   global x, y
   x += dx
   y += dy
   seznam.extend((x, y))
   canvas.coords(čára, seznam)
def událost_vlavo(event):
   kresli(-10, 0)
def událost_vpravo(event):
   kresli(10, 0)
def událost_nahoru(event):
   kresli(0, -10)
def událost_dolu(event):
   kresli(0, 10)
canvas = tkinter.Canvas()
canvas.pack()
ciara = canvas.create_line(0, 0, 0, 0) # zatím prázdná čára
canvas.bind_all('<Left>', událost_vlevo)
canvas.bind_all('<Right>', událost_vpravo)
canvas.bind_all('<Up>', událost_nahoru)
canvas.bind_all('<Down>', událost_dolů)
tkinter.mainloop()
```

Kreslení pomocí tohoto programu může připomínat dětskou hračku „magnetická tabulka“:

Časovač

metoda after()

Metoda grafické plochy ***after(),*** která pozdrží výpočet o nějaký počet milisekund, je všestrannější: můžeme pomocí ní startovat časovač a může mít jeden z těchto tvarů:

canvas.after(milisekundy)
canvas.after(milisekundy, funkce)

První parametr *milisekundy* již známe: výpočet se pozdrží o příslušný počet milisekund.
Jenže, je-li metoda zavolána i s druhým parametrem funkce, výpočet se ve skutečnosti nezdrží, ale pozdrží se vyvolání zadané funkce (parametr funkce musí být reference na funkci, tedy většinou bez kulatých závorek). Tato vyvolaná funkce musí být definována bez parametrů.

S tímto druhým parametrem metoda ***after()*** naplánuje (někdy do budoucna) spuštění nějaké funkce a přitom výpočet pokračuje normálně dále na dalším příkazu za ***after()*** (bez pozdržení).
Tomuto mechanismu říkáme časovač (naplánování spuštění nějaké akce), anglicky *timer*.
Funkce naplánuje spuštění sebe sama po nějakém čase. Můžete si to představit tak, že v počítači tikají nějaké hodiny s udanou frekvencí v milisekundách a při každém tiknutí se provedou příkazy v těle funkce.

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

Časovač každou sekundu vypíše do textové plochy řetězec 'tik'.

V následujícím příkladu jsme přidali svázání s klávesou <*Enter*>, pomocí kterého se smaže celá grafická plocha (pomocí *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 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()
```

Protože během běhu časovače může program provádět další akce, může spustit třeba i další časovač. Program nyní spustí oba časovače: kreslí se červené kroužky a modré čtverečky. Jelikož druhý časovač má svůj interval 300 milisekund, tedy „tiká“ 3krát pomaleji než první, kreslí 3krát méně modrých čtverečků než první časovač červených kroužků.

Zastavování časovače

Na zastavení časovače nemáme žádný příkaz. Časovač můžeme zastavit jen tak, že on sám ve svém těle na konci nezavolá metodu ***canvas.after()*** a tím i skončí.

Upravíme předchozí příklad tak, že definujeme dvě globální proměnné, které budou sloužit pro oba časovače k zastavování. Abychom mohli tyto časovače zastavovat, resp. opět rozbíhat, přidáme dvě funkce pro zpracování události od kláves ***¨<a>¨ a <b>.*** Každá z těchto funkcí se bude starat o svůj 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 první(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>', první)
canvas.bind_all('<b>', druhy)
kresli()
kresli1()
tkinter.mainloop()
```

Nyní běží oba časovače, ale stačí přiřadit mačkat klávesy <a> nebo <b> a časovače se budou zastavovat nebo opět spouštět.

Bourající autíčka

Další ukázka bude hýbat dvěma obrázky autíček různou rychlostí.
Přestože tento program funguje dobře, má několik malých nedostatků:
• pokud během pohybu autíček (běží časovač pohyb()) znovu klikneme do plochy (vyvoláme událost start()), autíčka skočí do svých startovních pozic a znovu se vyvolá časovač pohyb(); teď to ale vypadá, že autíčka jedou dvojnásobnou rychlostí - totiž teď běží najednou dva časovače pohyb() i pohyb(), které oba pohnou oběma autíčky - ačkoli je to zajímavé, budeme se snažit tomuto zabránit
• pokud autíčka do sebe nabourají, vypíše se text 'BUM', který tam bude svítit i po opětovném nastartování autíček

Vylepšíme funkci ***start()*** takto:
• jelikož tato startuje časovač ***pohyb(),*** zablokujeme opětovné kliknutí tím, že zrušíme svázání události klik s funkcí ***start()***
• pokud svítí text ***'BUM'***, tak jej vymažeme

Využijeme nový příkaz ***unbind()***, pomocí kterého umíme rozvázat nějakou konkrétní událost s funkcí:

metoda unbind()
Metoda zruší svázání příslušné události: *canvas.unbind(jméno_události)*

V časovači (ve funkci) pohyb() při výpisu zprávy 'BUM', jelikož zastavujeme časovač, opětovně svážeme kliknutí do plochy s událostí start():

```
import tkinter
text = None
def start(event):
    canvas.unbind('<ButtonPress>') # zruší klikací událost
    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 klikací událost
    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()
```

Shrnutí událostí od myši

Událost ***'<ButtonPress>'*** reprezentuje kliknutí libovolným tlačítkem myši. Většinou má každá myš tři tlačítka: levá, střední a pravá. Přímo v názvu události můžeme určit, aby se událost vyvolala jen u konkrétního tlačítka. Tehdy bude název události takový:

- ***'<ButtonPress-1>'*** pro zatlačení levého tlačítka
- ***'<ButtonPress-2>'*** pro zatlačení středního tlačítka
- ***'<ButtonPress-3>'*** pro zatlačení pravého tlačítka

Můžeme zapsat například:

```
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_střední)
canvas.bind('<ButtonPress-3>', klik_pravy)
tkinter.mainloop()
```

Každá funkce se stará o kliknutí svého tlačítka. V parametru ***event*** kromě souřadnic ***event.x a event.y*** dostáváme také pořadové číslo tlačítka ***event.num***, kterým jsme právě klikli.
Předchozí program můžeme zapsat i takto:

```
import tkinter
def klik(event):
   barva = ['blue', 'green', 'red'][event.num-1]
   canvas.create_text(event.x, event.y, text=event.num, font='Arial 30', fill=barva)
canvas = tkinter.Canvas()
canvas.pack()
canvas.bind('<ButtonPress>', klik)
tkinter.mainloop()
```

Je to na programátorovi, kterou z těchto možností preferuje.
Jméno události ***'<ButtonPress-1>'*** existuje také ve zkrácené formě a to buď ***'<Button-1>'*** nebo dokonce ***'<1>'*** (zřejmě to funguje is 2 i 3).

Již známe jméno události pro tažení myši ***'<Motion>'.*** Tato pojmenovaná událost zavolá příslušnou funkci se zatlačeným i bez zatištěného tlačítka. Známe i variantu jména události *'<B1-Motion>'*, díky čemuž zpracováváme jen tahání se zatlačeným levým tlačítkem myši. Podobně jako při klikání můžeme specifikovat zatlačené tlačítko číslem od 1 do 3, bude to fungovat i při tažení, proto:

- ***'<B1-Motion>'*** tahání se zatlačeným levým tlačítkem
- ***'<B2-Motion>'*** tahání se zatlačeným středním tlačítkem
- ***'<B3-Motion>'*** tahání se zatlačeným pravým tlačítkem

Později uvidíme využití události i od puštění tlačítka myši **'<ButtonRelease>'.** I tato událost funguje pro varianty s konkrétním tlačítkem myši, například ***'<ButtonRelease-1>'*** zpracovává jen puštění levého tlačítka myši.