Základy grafiky

Aby sme mohli pomocou Pythonu kresliť, potrebujeme použiť externú knižnicu, v našom prípade `tkinter`. Vďaka nej vieme vytvoriť kresliace plátno (canvas), na ktoré potom vieme kresliť. Nasledovné príkazy by teda mal obsahovať každý náš program.

```
import tkinter

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

Tieto príkazy postupne vytvoria plátno na kreslenie a priradia ho premennej `canvas` a vložia toto plátno do okna (príkaz `pack()`).

Výsledné okno nám však nemusí vyhovovať, napríklad rozmermi, či farbou, preto by bolo fajn vedieť si ho upraviť.

```
import tkinter

canvas = tkinter.Canvas(width=100, height=100, bg='red')
canvas.pack()
```

`bg` je skratka od background a môžeme tam vložiť ľubovoľnú farbu. Ľavý horný roh tohto okna má súradnice $(0, 0)$. Následne x-ová súradnica rastie zľava doprava a y-ová rastie zhora nadol. Pravý dolný roh by mal mať teda súradnicu $(width, height)$

Kreslenie objektov

Na canvas vieme kresliť rôzne útvary.

```
canvas.create_text(x, y, text='retazec', font='arial 30')
canvas.create_rectange(x1, y1, x2, y2, outline='blue')
canvas.create_oval(x1, y1, x2, y2, fill='red')
canvas.create_line(x1, y1, x2, y2, width=5)
```

Ako vidíme, vykresľované útvary vieme rôzne modifikovať. Napríklad im meniť farbu obrysu (`outline`), vyfarbenie (`fill`), hrúbku čiary (`width`). Väčšina z týchto vecí sa dá použiť vo všetkých spomenutých príkladoch, nemusíme ich však použiť ak nechceme. Vyskúšajte si to.

Podprogram

Občas, keď chceme mať program pekne štrukturovaný, poprípade chceme tú istú časť príkazov používať stále dookola, chceme použiť podprogramy. Podprogram, teda funkciu, definujeme pomocou príkazu `def`, za ktorým nasleduje názov funkcie.

```
def strom():
    canvas.create_rectangle(50,80,70,150,fill='brown')
    canvas.create_oval(30,40,90,100,fill='green')

strom()
```

Táto funkcia nakreslí obrázok stromu na zadaných súradniciach. Čo ak však chceme funkciu modifikovať, napríklad, chceme stromy kresliť na rôznych pozíciách? Stačí do funkcie pridať parametre. Hodnoty, ktoré sa do funkcie vložia pri jej volaní. Všimnite si, ako sa zmení definícia funkcie.

```
def strom(x,y):
    canvas.create_rectangle(x,y,x+20,y+70,fill='brown')
    canvas.create_oval(x-20,y-40,x+40,y+20,fill='green')

strom(10, 10)
strom(50, 50)
```

Takýto program nakreslí dva stromy – jeden na pozícii $(10, 10)$ a druhý na pozícii $(50, 50)$.

Reakcia na stlačenie myši

Zišlo by sa nám, keby náš program vedel reagovať na stlačenie tlačidiel na myši. A to sa skutočne dá pomocou príkazu `canvas.bind('udalost', funkcia)`. Použitie takéhoto príkazu spôsobí, že keď nastane `udalosť` (čo môže byť stlačenie myši, pohyb myši a podobne), spustí sa `funkcia`.

```
def stvorec(suradnice):
    x = suradnice.x
    y = suradnice.y
    canvas.create_rectangle(x,y,x+30,y+30)

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

Takýto kus kódu vykreslí štvorec na mieste, kde stlačíme ľavé tlačítko myši (`<Button-1>`). Všimnite si, že funkcia `stvorec()` dostane parameter súradnice. To budú presne súradnice canvasu, na ktorých bola myška stlačená. Takisto si všimnite, že aby sme sa dostali k skutočnej x-ovej súradnici, musíme použiť `suradnice.x`.

Globálne premenné

Občas by sme chceli, aby funkcia štvorec videla nejakú premennú mimo nej. Napríklad farbu, ktorou má kresliť.

```
def stvorec(suradnice):
    x = suradnice.x
    y = suradnice.y
    canvas.create_rectangle(x,y,x+30,y+30, fill=farba)

farba = 'red'
canvas.bind('<Button-1>',stvorec)
```

Nanešťastie, takéto niečo Python nechápe. Preňho je premenná `farba` vo vnútri funkcie iná, ako premenná mimo nej. Našťastie, môžeme použiť príkaz `global`, ktorý funkcii `stvorec()` povie, že má použiť premennú, ktorá sa používa mimo nej. Správne to teda bude:

```
def stvorec(suradnice):
    global farba
    x = suradnice.x
    y = suradnice.y
    canvas.create_rectangle(x,y,x+30,y+30, fill=farba)

farba = 'red'
canvas.bind('<Button-1>',stvorec)
```

Prekresľovanie

Ak chceme, aby naše vykresľovanie vyzeralo „animovane“. Preto možno chceme veci prekresľovať a nechceli by sme, aby sa všetko vykreslilo naraz. Aby sme donútili program pred vykreslením počkať, môžeme použiť príkaz `canvas.after(milisekundy)`, ktorý ho donúti počkať príslušný počet milisekúnd. Následne potom však musíme zavolať aj `canvas.update()`, aby sa canvas znova obnovil.