Časovač

Opakovanie je dôležitá súčasť akéhokoľvek programovacieho jazyka. Jeden spôsob na opakovanie poznáme – `for` cykly. To však nie je vždy vhodné. Čo keď nevieme ako dlho sa má niečo opakovať? A ešte horšie, `for` cyklus je jeden príkaz, ktorý sa celý vykoná naraz. To znamená, že program akoby stojí na ňom a nič iné robiť nemôže. Tento problém sme mali, keď sme chceli rozpohybovať dve guličky. Ak sme ich pohyb dali do dvoch `for` cyklov, tak druhá gulička čakala, kým sa neposunula prvá toľkokrát koľko mala a až potom sa začala hýbať.

Všetky tieto problémy vieme jednoducho vyriešiť pomocou časovača. Myšlienka je jednoduchá – máme funkciu, ktorá niečo robí, napríklad posunie guličku o 5 pixelov dodola. Túto funkciu chceme opakovane volať, ale medzi týmito volaniami má byť časová medzera, povedzme 20 milisekúnd. Naviac nechceme, aby sa týchto 20 milisekúnd nič nerobilo (nech to znie akokoľvek zvláštne). Môžeme preto funkciu guličky načasovať.

```
def gulicka():
    global y
    y += 5
    canvas.delete('all')
    canvas.create_oval(200, y, 200 + 25, y + 25)
    canvas.after(20, gulicka)

y = 0
gulicka()
```

Nový príkaz je `canvas.after(20, gulicka)`, rozoberme si čo robí. V podstate ho môžeme prečítať ako: „O 20 milisekúnd spusti funkciu s menom gulicka().“. Čo sa teda bude diať? Funkcia gulicka() posunie guličku o 5 dodola a nastaví časovač, ktorý o 20 milisekúnd opäť spustí funkciu gulicka(). O 20 milisekúnd sa teda opäť spustí táto funkcia, ktorá opäť nastaví časovač... Až do nekonečna (alebo kým neukončíme program).

Nezabudnite, že aby sa toto celé začalo vykonávať, musíme funkciu gulicka() zavolať prvýkrát.

No a čo sa týka tých ušetrených 20 milisekúnd, to si všimnete v okamihu, keď takýchto časovačov nastavíte viacero. Jeden časovač totiž neblokuje vykonávanie ostatných a v tých 20 voľných milisekundách sa vykonajú ostatné funkcie zvyšných časovačov. Nuž a ak vám to pomôže pre predstavu – za 20 milisekúnd stihne bežný počítač vykonať zhruba milión operácií (riadkov kódu). Takže 20 milisekúnd je naozaj veľa.

Ako posledné si dajte pozor, aby ste funkciu s časovačom nezavolali veľakrát. Keby sme napríklad pridali na koniec daného programu ďalší riadok `gulicka()`, tak naša gulička by bola zrazu dvakrát tak rýchla. Obe tieto funkcie by si totiž nastavili svoj časovač, takže naraz by čakali vždy dve volania funkcie `gulicka()`.

ID objektu a kde sa dá použiť

Vždy keď si pomocou `tkinter` niečo vykreslíte, Python sa vám pokúsi vrátiť ID (z anglického identifier) nakresleného obrázka. Predstavte si to ako jedinečnú značku (číslo), ktorá je pri ňom napísaná a podľa ktorej ho vie Python nájsť. Niečo podobné robíte ak používate `tag='znacka'`, akurát toto označenie nemusí byť jednoznačné. Toto ID si vieme pamätať a následne používať na to, aby sme Pythonu ukázali, s ktorým objektom chceme spraviť nejakú operáciu.

```
id = canvas.create_oval(200, 200, 225, 225)
print(id)    # toto vypise nejake cislo (pravdepodobne 1), cize hodnotu toho ID

canvas.delete(id)   # Pythonu sme povedali, aby zmazal iba obrazok, ktory ma dane ID
```

Zapamätať si toto ID vieme v ľubovoľnej premennej. V príklade vyššie dokonca vidíte využitie na vymazanie. Namiesto toho, aby sme si museli vytvárať `tag`, alebo mazať všetko `canvas.delete('all')` povieme Pythonu aby vymazal iba obrázok, ktorý má ID `id`. A keďže ID je jedinečné, tak sa vymaže iba ten jeden konkrétny krúžok.

Samozrejme, príkazov, ktoré sa odkazujú na ID je veľa, my si ukážeme niekoľko z nich.

- `canvas.delete(id)` – vymaže objekt s príslušným ID
- `canvas.move(id, x, y)` – posunie objekt s ID id o x v x-ovej súradnici a o y v y-ovej súradnici
- `canvas.coords(id, x1, y1, x2, y2)` – nastaví súradnice daného objektu id na zadané čísla. V podstate to za nás spraví `canvas.delete(id) id=canvas.create_oval(x1, y1, x2, y2)`
- `canvas.coords(id)` – príkaz `.coords()` vieme použiť aj iba s ID. Vtedy nám vráti list súradníc daného objektu. A v liste vieme pristupovať k hodnotám pomocou `[]`.

- `canvas.coords(id)[0]` – vráti x-ovú súradnicu objektu id
- `canvas.coords(id)[1]` – vráti y-ovú súradnicu objektu id

Čo sa týka pohybu guličku, v podstate sú dva hlavné prístupy. V prvom si držíme vlastnú x-ovú a y-ovú súradnicu v premenných a používame `.delete()` a `.create_...()` na vymazávanie a opätovné kreslenie objektov.

V druhom zase používame `.move()` na posúvanie a keď chceme vedieť súradnice, zavoláme funkciu `.coords()`. V tomto prípade si však potrebuje poriadne pamätať id objektu.

Tieto dva prístupy sa samozrejme dajú miešať, treba si však rozmyslieť, čo si na ne potrebujete pamätať. Teda či potrebuje id, x a y. A ak áno, tak si ich aj priebežne upravovať podľa toho ako sa objekt mení. Neočakávajte, že keď zavoláte `canvas.move(id, 0, 5)`, tak sa vaša premenná s menom y zväčší o 5. Nezväčší kým tak nespravíte pomocou `y += 5`.

Matematika

Informatika má v sebe ako súčasť slova matika a naozaj sa v nej plne využíva. Preto budete musieť vedieť aj nejaké základné veci z matematiky. Pri pohyboch útvarov proste musíte vedieť rozmýšľať nad vektormi, sínusmi a podobne. V podstate by mali ale stačiť nasledovné veci – výpočet sínusu/cosínusu, Pytagorova veta a podobnosť trojuholníkov. S tým by už malo ísť všetko. Hlavne si to treba nakresliť a popísať do obrázku čo už poznáme.

Užitočné príkazy:

- `import math` – nech vôbec máme potrebnú knižnicu
- `math.cos(x)` – vypočíta hodnotu cosínusu z x radiánov
- `math.sin(x)` – vypočíta hodnotu sínusu z x radiánov
- `math.radians(x)` – zmení x stupňov na radiány