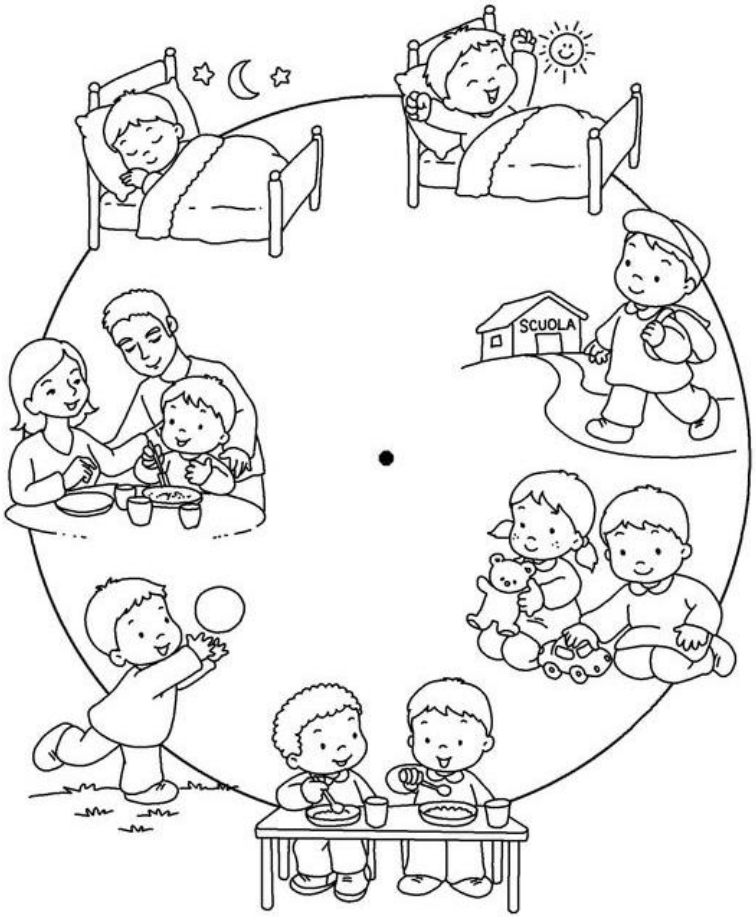
PYTHON

09o. opakovanie

Vypracovala: Ing. Eva Gabonayová
Predmet: Informatika
Vzdelávacia oblasť: Matematika a práca s informáciami
Dátum spracovania: 10. 9. 2018

Cyklus



Cyklus

- Programová konštrukcia **cyklus** nám poskytuje prostriedok umožňujúci opakovať činnosť alebo činnosti.
- Pri jeho použití je potrebné vedieť, **čo** sa má opakovať a **dokedy** sa to má opakovať.
- Činnosť, ktorá sa opakuje, označujeme ako **telo cyklu**, podmienku, ktorá určuje dokedy sa bude telo cyklu opakovať, nazývame **podmienka cyklu**.

Cyklus FOR

- cyklus so známym počtom opakovaní
- vieme presne, koľkokrát telo cyklu prebehne

Príklad

- Ak potrebujeme 5-krát vypísať ten istý text, môžeme to zapísať napr. takto:

```
print('programujem v Pythone')
print('programujem v Pythone')
print('programujem v Pythone')
print('programujem v Pythone')
print('programujem v Pythone')
```

... alebo

- Ak potrebujeme 5-krát vypísať ten istý text, môžeme to zapísať napr. aj takto:

```
for i in 1, 2, 3, 4, 5:
    print('programujem v Pythone')
```

i – riadiaca premenná cyklu

... ako to funguje

```
for i in 1, 2, 3, 4, 5:
    print('programujem v Pythone')
```

- do premennej i sa bude postupne priraďovať nasledovná hodnota zo zoznamu hodnôt; začíname s prvou hodnotou, teda v tomto prípade 1
- pre každú hodnotu so zoznamu sa vykonajú príkazy, ktoré sú v **tele cyklu**, t.j. tie príkazy, ktoré sú odsadené
- v našom príklade sa päťkrát vypíše rovnaký text, hodnota premennej i nemá na tento výpis žiadny vplyv
- všimnite si znak ‚:‘ na konci riadka s for - ten je tu povinne, bez neho by to nefungovalo

... ako to funguje

```
for i in 1, 2, 3, 4, 5:
    print('programujem v Pythone')
    print('tešíme sa')
```

Telo cyklu

- tvoria príkazy, ktoré sa majú opakovať; definujú sa **odsadením** príslušných riadkov
- odsadenie je povinné a musí byť minimálne 1 medzera, odporúča sa odsadzovať vždy o **4 medzery**
- ak telo cyklu obsahuje viac príkazov, všetky **musia** byť odsadené o rovnaký počet medzier
- telo cyklu nesmie byť prázdne, musí obsahovať aspoň jeden príkaz

Generovanie **postupnosti** čísel – funkcia **range()**

```
for i in range (1,6):
    print('programujem v Pythone')
```

- **`range (1,6)`**
- dva parametre, ktoré určujú interval sprava otvorený – postupnosť čísel: 1,2,3,4,5

… ďalšie príklady

range(10)	0,1,2,3,4,5,6,7,8,9
range(0, 10)	0,1,2,3,4,5,6,7,8,9
range(0, 10, 1)	0,1,2,3,4,5,6,7,8,9
range(3, 10)	3,4,5,6,7,8,9
range(3, 10, 2)	3,5,7,9
range(10, 100, 10)	10,20,30,40,50,60,70,90
range(10, 1, -1)	10,9,8,7,6,5,4,3,2

Zhrnutie

- for-cyklus pomocou **premennej cyklu** vykoná zadaný počet-krát **telo cyklu**

```
for i in range (1,6):
    print('programujem v Pythone')
    print('tešíme sa')
```

- funkcia range()vytvorí postupnosť celočíselných hodnôt, najčastejšie pre for-cyklus

Časté chyby: chýba dvojbodka, zlé odsanenie príkazov tela cyklu

Spomalenie vykresľovania v cykle For

```
canvas.update() # zabezpečí, aby sa medzi čakaniami aktualizovalo grafické plátno
canvas.after(1000) # podrží vykonávanie programu na čas uvedený v zátvorkách(čas v milisekundách)
1000 ms = 1 s
```

Podprogram - funkcia

už sme pracovali s funkciami, napr.

- výstup **print()**
- generovanie postupnosti čísel pre for-cyklus **range()**
- funkcia **random.randint()** či **random.randrange()**

Podprogram - funkcia

- menší **kúsok programu, ktorý pomenujeme**
- keď ho chceme použiť, len napíšeme v programe jeho meno
- je to ako keby sme si vytvorili **vlastný príkaz**
- najprv definujeme funkciu (vytvoríme ju), potom ju v programe voláme
- použitím podprogramov je program **prehľadnejší**, pretože obsahuje **menej príkazov**
- **opakujúce sa vzory** v programe (napr. v obrázku) dáme do funkcie (podprogramu)

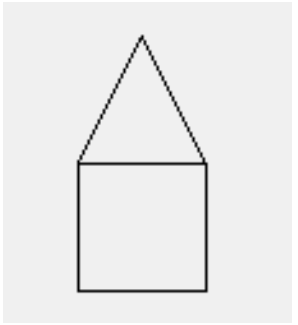
Definícia funkcie

- funkcia je **pomenovaný blok príkazov** (niekedy sa tomu hovorí aj podprogram).
- popisujeme (**definujeme**) ju špeciálnou konštrukciou:

```
def meno_funkcie():
    prikaz
    prikaz
    ...
```

 # zapamätaj si blok príkazov (telo funkcie) ako nový príkaz

Príklad funkcie dom



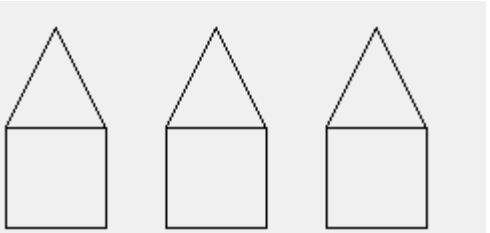
- Program nakreslí dom z obdĺžnika a lomenej čiary

```
def dom():
    canvas.create_rectangle(100,200,150,250)

    canvas.create_line(100,200,125,150,150,200)

dom()
```

Voláme funkciu dom () - funkcia bez parametrov

Príklad funkcie dom s parametrami

- Program nakreslí tri domy



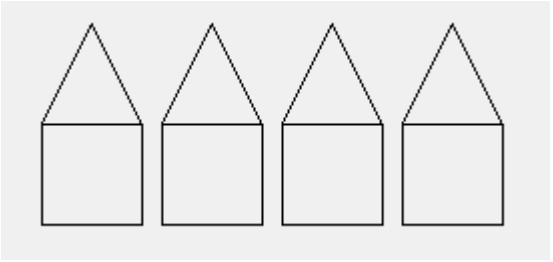
```
def dom(x,y):
    canvas.create_rectangle(x,y,x+50,y+50)
    canvas.create_line(x,y,x+25,y-50,x+50,y)

dom(100,100)
dom(180,100)
dom(260,100)
```

3x voláme funkciu dom (x,y) - funkcia s dvoma parametrami

Časté chyby: chýba dvojbodka, zlé odsanenie príkazov tela funkcie

Príklad funkcie dom v tele cyklu

- Program nakreslí štyri domy



```
def dom(x,y):
    canvas.create_rectangle(x,y,x+50,y+50)
    canvas.create_line(x,y,x+25,y-50,x+50,y)

for i in range(1,5):
    dom (i*60, 100)
```

Časté chyby: chýba dvojbodka, zlé odsanenie príkazov tela funkcie

Udalosť

- je akcia, 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 programe potom môžeme nastaviť, čo sa má udiať pri ktorej udalosti
- tomuto sa zvykne hovoriť **udalosťami riadené programovanie** (event-driven programming)

Metóda **bind()**

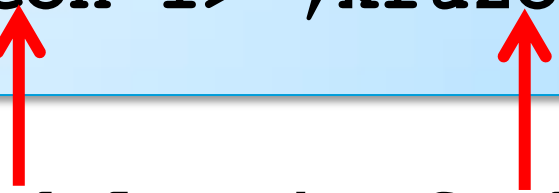
- metóda grafickej plochy ktorá slúži na **zviazanie** niektorej konkrétnej **udalosti** s nejakou **funkciou**, ktorá sa bude v programe starať o spracovanie tejto udalosti. Jej formát je:

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

- **meno_udalosti** je znakový reťazec s popisom udalosti (napr. pre kliknutie tlačidlom myši)
- **funkcia** sa spustí pri vzniku tejto udalosti

Reakcia na ľavé tlačidlo myši

```
def kruzok1(suradnice): #súradnice myši x,y
    x = suradnice.x
    y = suradnice.y
    canvas.create_oval(x-5,y-5,x+5,y+5,fill=,red')
canvas.bind('<Button-1>',kruzok1)
```

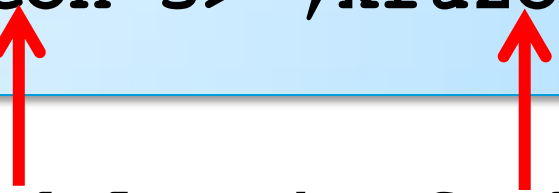


```
canvas.bind(meno_udalosti, funkcia)
```

na mieste kliknutia ĽTM sa nakreslí červený krúžok

Reakcia na pravé tlačidlo myši

```
def kruzok2(suradnice): #súradnice myši x,y
    x = suradnice.x
    y = suradnice.y
    canvas.create_oval(x-5,y-5,x+5,y+5,fill='blue')
canvas.bind('<Button-3>',kruzok2)
```



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

na mieste kliknutia PTM sa nakreslí modrý krúžok

“Myšacie“ udalosti

- <Button-1> kliknutie ĽTM
- <Button-3> kliknutie PTM
- <Double-Button-1> dvojklik ĽTM
- <ButtonRelease-1> uvoľnenie ĽTM
- <Motion> pohyb myši bez zatlačeného TM
- <B1-Motion> pohyb myši so zatlačeným ĽTM
- <B3-Motion> pohyb myši so zatlačeným PTM

ĽTM – ľavé tlačidlo myši, PTM – pravé tlačidlo myši, TM – tlačidlo myši

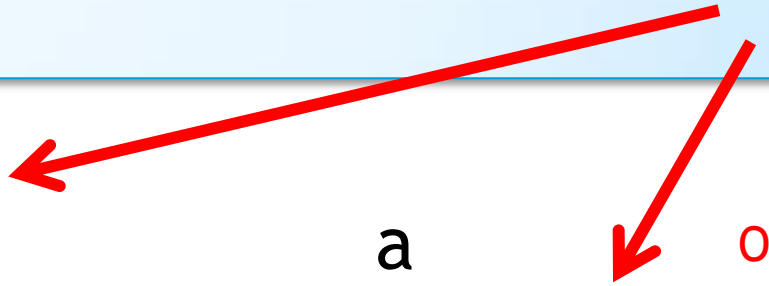
Udalosť

- je akcia, ktorá vznikne mimo behu programu a program môže na túto situáciu reagovať
- v programe potom môžeme nastaviť, čo sa má udiať pri udalosti
- na udalosti stlačenia klávesu reagujeme príkazom

```
canvas.bind_all('klaves', funkcia)
```

Meno klávesu

```
canvas.bind_all('klaves', funkcia)
```



- 'a' a
- '<up>' šípka hore
- '<down>' šípka dole
- '<right>' šípka vpravo
- '<left>' šípka vľavo
- '<space>' medzera
- '<return>' enter

Ovládacie klávesy

Parameter funkcia

```
canvas.bind_all('klaves', funkcia)
```

- funkcia – názov funkcie, ktorá sa vykoná stlačením klávesu
- musí mať jeden parameter napr. event (udalosť)
- v parametri **event** sa funkcii pošlú podrobnejšie informácie o udalosti
- aj keď s týmto parametrom nepracujeme, musíme ho vo funkcii zadať

Príklad 1.

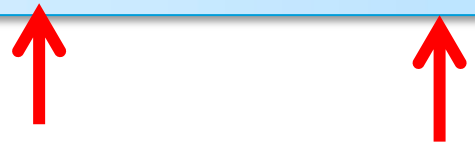
```
def stvorec(event):
    x = randrange(300)
    y = randrange(250)
    canvas.create_rectangle(x, y, x+50, y+50)
canvas.bind_all('s', stvorec)
```

kláves, funkcia)

po stlačení klávesu **s** sa nakreslí na náhodnej pozícii štvorec

Príklad 2.

```
def sipka_hore(event):
    x = randrange(300)
    y = randrange(250)
    canvas.create_line(x-10, y+20, x, y, x+10, y+20)
    canvas.create_line(x, y, x, y+40)

canvas.bind_all('<Up>', sipka_hore)
```



kláves, funkcia)

po stlačení klávesu **šípka hore** sa nakreslí na náhodnej pozícii šípka hore

Udalosť na stlačenie ktoréhokoľvek klávesu '<Key>'

```
canvas.bind_all('<Key>', funkcia)
```

- vo funkcii, ktorá reaguje na túto udalosť, môžeme cez vstupný parameter a jeho **zložku (atribút)** zistiť napríklad:
- event.**keysym** symbol klávesu
- event.**char** samotný znak klávesu
- event.**keycode** číselný kód klávesu (Ascii kód)

Príklad 3.

```
def pis(event):
    print(event.keysym)
    x = randrange(300)
    y = randrange(300)
    canvas.create_text(x,y,text=event.keysym)

canvas.bind_all('<Key>', pis)
```

Rozhodovanie

Vetvenie

Podmienený príkaz

- pri programovaní často riešime situácie, keď sa program má na základe nejakej **podmienky** rozhodnúť medzi **viacerými možnosťami**
- podmienka je **logický výraz**, ktorého hodnota je pravda (true) alebo nepravda (false)

```
if podmienka:      #ak podmienka platí vykonaj príkazy
    prikaz
    ...
else:              #ak podmienka neplatí vykonaj príkazy
    prikaz
    ...
```

Podmienený príkaz - neúplný

- pri neplatnej podmienke sa nič nevykoná
- chýba vetva else

```
if podmienka:      #ak podmienka platí vykonaj príkazy
    prikaz
    ...
                   #ak podmienka neplatí nevykonaj nič
```

Podmienky

- podmienky zapisujeme podobne ako v matematike

`body < 90`	je menšie ako
`body <= 50`	je menšie alebo rovné
`body == 50`	rovná sa
`body != 77`	nerovná sa
`body > 100`	je väčšie ako
`body >= 90`	je väčšie alebo rovné
`40 < body <= 50`	je väčšie ako … a zároveň menšie alebo rovné …
`a < b < c`	`a` je menšie ako `b` a zároveň je `b` menšie ako `c`

Logické operátory and, or

- v podmienenom príkaze môžeme zadávať aj viacero podmienok a spájame ich logickým operátorom **and** a **or**.
- keď chceme mať splnené **všetky podmienky súčasne**, použijeme operátor **and**
- keď stačí splnenie **aspoň jednej** z podmienok, použijeme operátor **or**

elif - konštrukcia, ktorá uľahčuje vnorenú sériu if-ov

```
if podmienka_1:      # ak podmienka_1 platí, vykonaj 1. skupinu príkazov
    prikaz
    ...
elif podmienka_2:    # ak podmienka_1 neplatí, ale platí podmienka_2, ...
    prikaz
    ...
elif podmienka_3:    # ak ani podmienka_1 ani podmienka_2 neplatia, ale platí podmienka_3, ...
    prikaz
    ...
else:                # ak žiadna z podmienok neplatí, ...
    prikaz
    ...
```

Časovač (timer)

- mechanizmus, pomocou ktorého môžeme v programe nejakú akciu nechať stále vykonávať v určitých časových intervaloch
- 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

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

Globálne a lokálne premenné

- lokálna premenná – platí len v podprograme (funkcii)

- globálna premenná – platí aj v hlavnom programe

```
global x, y, a
```

Ďalšie príkazy

```
canvas.delete ( all)
```

- zmaže všetko, čo je nakreslené na plátne

Spustenie časovača

```
def lopticka():
    canvas.delete('all')
    canvas.create_oval(x-5, y-5, x+5, y+5)
    global y
    y = y+5
    if y<200:
        canvas.after(100, lopticka)
x = 200
y = 5
lopticka()
```

Zastavenie časovača

```
def stop(suradnice):
    global pokracovat
    if pokracovat == 1:
        pokracovat = 0
    else:
        pokracovat = 1
        lopticka()
pokracovat = 1
lopticka()
canvas.bind('<Button-1>',stop)
```

Použitie časovača

- animácie
- jednoduché hry
- šetrič obrazovky
- nekonečný cyklus
- cyklus s podmienkou

Vytvorenie tlačidla

- vytvoríme tlačidlo a nastavíme mu, **čo má robiť po kliknutí**

```
button1 =
tkinter.Button(text='kresli',


command=button1_klik)
button1.pack()
```

zabezpečí zobrazenie tlačidla

Funkcia button1_klik

- funkcia je nastavaná ako príkaz pre tlačidlo
- v tejto funkcii zapíšeme, čo všetko sa má diať po stlačení tlačidla

```
button1 =
tkinter.Button(text='kresli',
command=button1_klik)
button1.pack()
```

Čo robí nasledujúca funkcia?

```
def button1_klik():
    x=randrange(300)
    y=randrange(200)
    info=randrange(100)
    canvas.create_rectangle(x-10, y-10, x+10, y+10)
    canvas.create_text(x, y, text=info)
```

Vstupné pole

- hovorí sa mu aj textové pole, editovací riadok alebo editovacie políčko
- slúži na **zadávanie vstupu** do programu


```
entry1 = tkinter.Entry()
entry1.pack()
```

zabezpečí zobrazenie tlačidla

Vstupné pole

- ak chceme zistiť či použiť aktuálne údaje v našom vstupnom poli (entry1), použijeme funkciu **`entry1.get()`**
- to, čo zadáme do vstupného poľa je reťazec znakov – text (**str**)
- **int** – z reťazca urobí celé číslo
- **float** - z reťazca urobí desatinné číslo

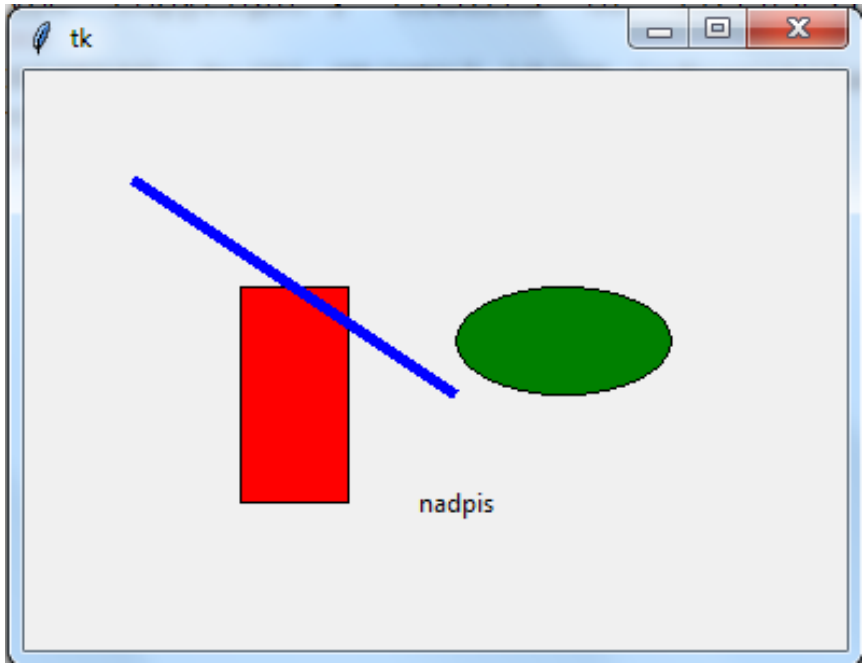
Čo robí nasledujúca funkcia?

```
def button1_klik():
    od = int(entry1.get())
    do = int(entry2.get())
    for i in range (od, do):
        stvorcek(i*20, 100, i)

def stvorcek(x, y, info):
    canvas.create_rectangle(x-10, y-10, x+10, y+10)
    canvas.create_text(x, y, text=info)
```

Objekty (útvary) canvasu

V Pythone je každý vytvorený útvar (canvas.create_) objektom, ktorý sa dá :

- meniť
- posúvať
- modifikovať



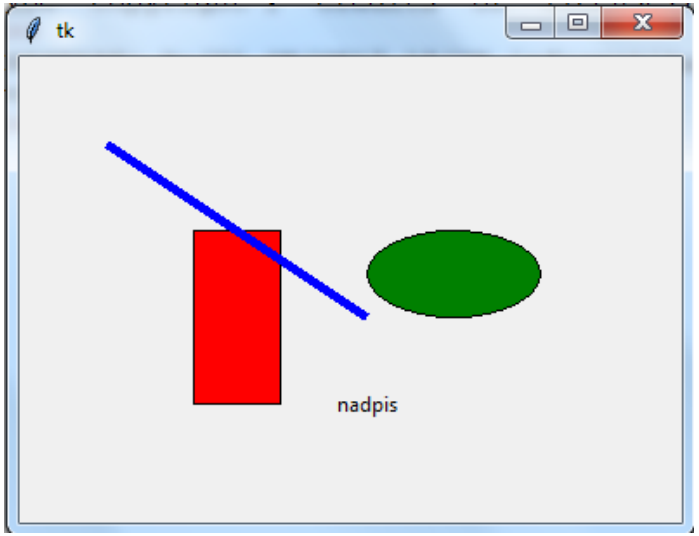
Nie je to len zafarbený bod v grafickom plátne.

Objekty (útvary) canvasu

canvas.create_

- text
- rectangle
- line
- oval

- 1
- 2
- 3
- 4



čísla objektov môžeme použiť na rôzne zmeny objektov:

- posúvať canvas.move(**3**,5,10)
- canvas.move(**' all'**,5,10)

Objekty (útvary) canvasu

- text
- rectangle
- line
- oval

- 1
- 2
- 3
- 4

- nadpis
- obdlznik
- ciara
- elipsa

- čísla vytvorených objektov si môžeme zapamätať do premennej
- canvas.move(**1**,5,10)
- canvas.move(**nadpis**,5,10)

Objekty (útvary) canvasu

- text
- rectangle
- line
- oval

- 1
- 2
- 3
- 4

- nadpis
- obdlznik
- ciara
- elipsa

- ak obrázok pozostáva z viacerých objektov, všetky časti obrázku označíme spoločnou značkou **tags = 'robot'**
- canvas.move(' **robot'**,5,10)

Práca s textom

- ľubovoľný reťazec znakov v apostrofoch
  - reťazec znakov
  - textový reťazec
  - string
- Spájanie reťazcov operáciou +

Zdroje:

- e-učebnica: Peter Kučera: **Programujeme v Pythone**, učebnica informatiky pre SŠ
- www.python.org
- Python
- Interaktivny Python

Ďakujem za pozornosť!