11. Korytnačky (turtle)

video prezentácia

korytnačky

Už sme si zvykli na to, že grafika v Pythone (teda `tkinter`) funguje takto: do grafickej plochy môžeme klásť rôzne **grafické objekty** (obdĺžmik, oval, čiara, obrázok, …). Kladieme sme ich na **absolútne pozície** (súradnice), ale môžeme ich neskôr, napríklad posúvať, meniť ich parametre (hrúbky, farby), rušiť ich, a rôzne inak meniť. Napríklad:

```
import tkinter

canvas = tkinter.Create()                          # vytvor grafickú plochu
canvas.pack()                                      # zobraz ju do okna

i = g.create_oval(100, 50, 150, 80, fill='red')    # nakresli červenú elipsu
canvas.itemconfig(i, fill='blue')                  # zmeň farbu výplne elipsy

tkinter.mainloop()                                 # zabezpeč grafickú aplikáciu
```

Pri programovaní takýchto úloh sme počítali s tým, že:

- počiatočný bod `(0, 0)` je v ľavom hornom rohu grafickej plochy
- `x`-ová súradnica ide vpravo vodorovne po hornej hrane grafickej plochy
- `y`-ová súradnica ide nadol zvislo po ľavej hrane grafickej plochy: smerom nadol sú kladné `y`-ové hodnoty, smerom nahor idú záporné hodnoty súradníc

Teraz sa naučíme pracovať s grafikou, ktorá funguje na inom princípe: v grafickej ploche sa bude nachádzať **grafické pero**, ktoré budeme vedieť posúvať a tým budeme v ploche zanechávať farebnú stopu. Grafické pero (budeme mu hovoriť **korytnačka**, *turtle*) si musí stále pamätať svoju **pozíciu v ploche** (súradnice) ale aj momentálny **smer natočenia**. Okrem toho si pero pamätá aj svoju nastavenú farbu, hrúbku a tiež to, či pri pohybe zanecháva alebo nezanecháva čiaru (či je pero spustené alebo zdvihnuté). Uvidíme aj ďalšie atribúty grafického pera.

Korytnačia grafika

Korytnačka je grafické pero (grafický robot), ktoré si okrem pozície v grafickej ploche (`pos()`) pamätá aj smer natočenia (`heading()`). Korytnačku vytvárame podobne ako v `tkinter` grafickú plochu:

```
>>> import turtle
>>> t = turtle.Turtle()
>>> t.pos()
    (0.00,0.00)
>>> t.heading()
    0.0
```

Príkaz `t = turtle.Turtle()` vytvorí grafickú plochu a v jej strede korytnačku s natočením na východ. Volanie `t.pos()` vráti momentálnu pozíciu korytnačky `(0, 0)` a `t.heading()` vráti uhol `0`. Všimnite

si **bodkovú notáciu**: zápis `turtle.Turtle()` označuje, že z modulu `turtle` vyvoláme konštrukciu `Turtle()` (táto vytvára novú korytnačku). Zápis `t.pos()` označuje, že korytnačky `t` sa *pýtame*, akú má momentálne pozíciu (voláme jej **metódu** `pos()`). Výsledkom tejto metódy je dvojica desatinných čísel - súradnice korytnačky.

Pre grafickú plochu korytnačej grafiky platí:

- súradná sústava má počiatok v strede grafickej plochy
- `x`-ová súradnica ide vpravo vodorovne od počiatku
- `y`-ová súradnica ide nahor zvislo od počiatku: smerom nahor sú kladné `y`-ové hodnoty, smerom nadol idú záporné hodnoty súradníc
- smer natočenia určujeme v stupňoch (nie v radiánoch) a v protismere otáčania hodinových ručičiek:
    - na východ je to `0`
    - na sever je to `90`
    - na západ je to `180`
    - na juh je to `270`
- pozícia a smer korytnačky je vizualizovaná malým čiernym trojuholníkom - keď sa bude korytnačka hýbať alebo otáčať, bude sa hýbať tento trojuholník.

Základnými príkazmi sú `forward(dĺžka)`, ktorý posunie korytnačku v momentálnom smere o zadanú dĺžku a príkazy `right(uhol)` a `left(uhol)`, ktoré otočia korytnačku o zadaný uhol vpravo, resp. vľavo, napríklad:

```
>>> import turtle
>>> t = turtle.Turtle()
>>> t.forward(100)
>>> t.right(90)
>>> t.forward(100)
>>> t.right(90)
>>> t.forward(100)
>>> t.right(90)
>>> t.forward(100)
```

nakreslí štvorec so stranou 100. Častejšie budeme používať skrátené zápisy týchto príkazov: `fd()`, `rt()` a `lt()`.

mainloop()

Takýto spôsob postupného testovania korytnačích príkazov a programov s korytnačkami nemusí fungovať mimo **IDLE**. Niektorí z vás už máte skúsenosti z iných vývojových prostredí, že pri práci s grafikou musíte na záver programu pripísať `canvas.mainloop()`. Pri práci s korytnačou grafikou Python tiež využíva `tkinter`, preto aj v tomto prípade musíme v niektorých prostrediach na záver programu zadať, napríklad:

```
turtle.mainloop()         # to isté aj turtle.done()
```

Pripadne môžeme namiesto tohto zapísať

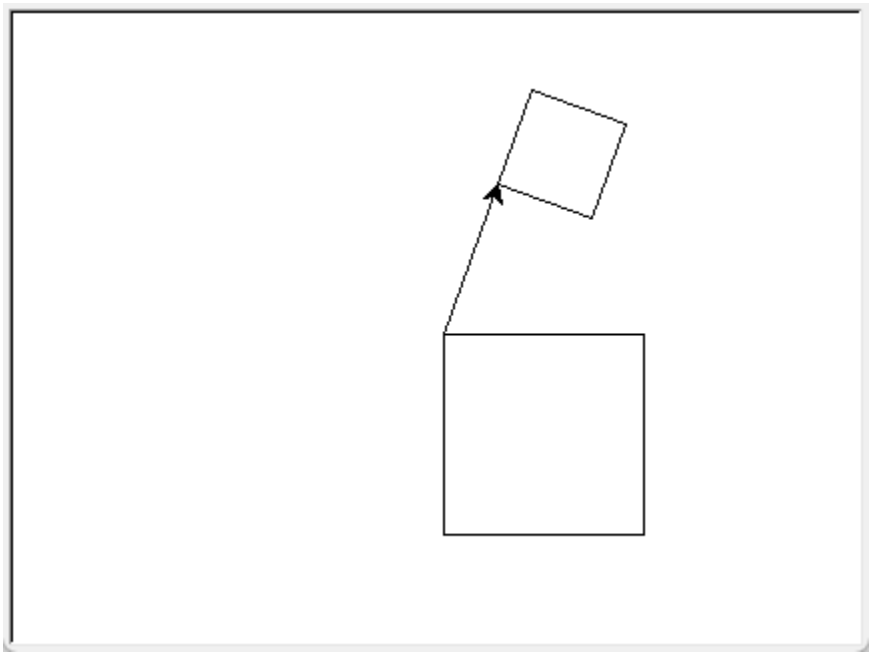
```
turtle.exitonclick()
```

čo bude označovať, že grafické okno sa automaticky zatvorí po kliknutí niekam do okna.

Zapíšme funkciu, pomocou ktorej korytnačka nakreslí štvorec:

```
import turtle

def stvorec(dlzka):
    for i in range(4):
        t.fd(dlzka)
        t.rt(90)

t = turtle.Turtle()
stvorec(100)
```

```
t.lt(70)
t.fd(80)
stvorec(50)
turtle.done()
```



Program nakreslí dva rôzne štvorce - druhý je posunutý a trochu otočený.

Aby sa nám ľahšie experimentovalo s korytnačkou, nemusíme stále reštartovať shell z programového režimu (napríklad klávesom **F5**). Keď máme vytvorenú korytnačku `t`, stačí zmazať kresbu pomocou:

```
>>> t.clear()     # zmaže grafickú plochu a korytnačku nechá tam, kde sa momentálne nachádza
```

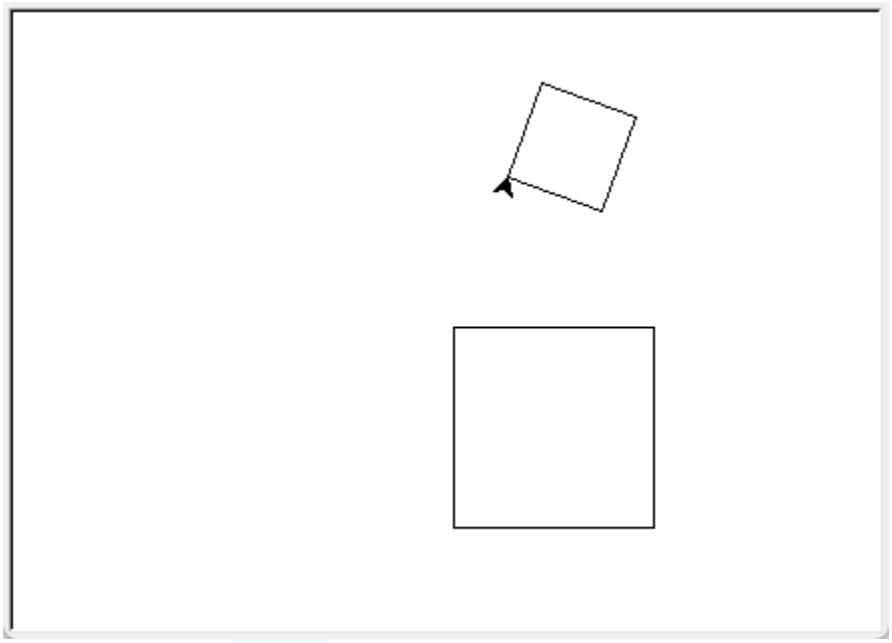
alebo pri zmazaní plochy aj inicializuje korytnačku v strede plochy otočenú na východ:

```
>>> t.reset()     # zmaže grafickú plochu a inicializuje korytnačku
```

a teraz znovu kreslíme už do prázdnej plochy.

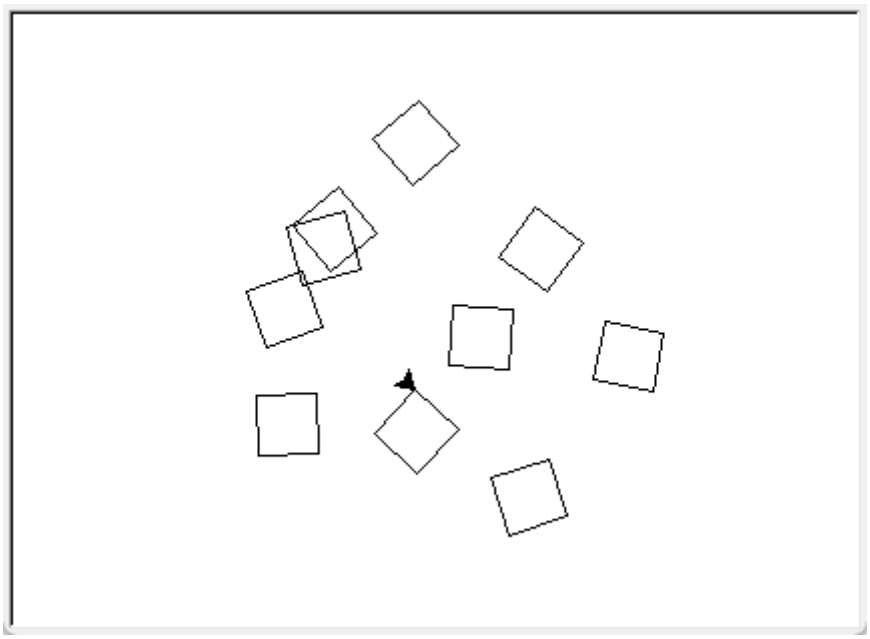
Korytnačka má pero, ktorým kreslí pri svojom pohybe po grafickej ploche. Toto pero môžeme zdvihnúť (`pen up`) - odteraz sa pohybuje bez kreslenia, alebo spustiť (`pen down`) - opäť bude pri pohybe kresliť. Na to máme dva príkazy `penup()` alebo `pendown()`, prípadne ich skratky `pu()` alebo `pd()`. Predchádzajúci príklad doplníme o dvíhanie pera:

```
import turtle

def stvorec(dlzka):
    for i in range(4):
        t.fd(dlzka)
        t.rt(90)

t = turtle.Turtle()
stvorec(100)
t.pu()
t.lt(70)
t.fd(80)
t.pd()
stvorec(50)
turtle.done()
```



Napíšme funkciu `posun()`, ktorá presunie korytnačku na náhodnú pozíciu v ploche a dá jej aj náhodný smer:

```
import turtle
import random

def stvorec(dlzka):
    for i in range(4):
        t.fd(dlzka)
        t.rt(90)

def posun():
    t.pu()
    t.setpos(random.randint(-100, 100), random.randint(-100, 100))
    t.seth(random.randint(0, 359))
    t.pd()

t = turtle.Turtle()
for i in range(10):
    posun()
    stvorec(30)
turtle.done()
```



- funkcia na náhodné pozície nakreslí 10 malých štvorcov

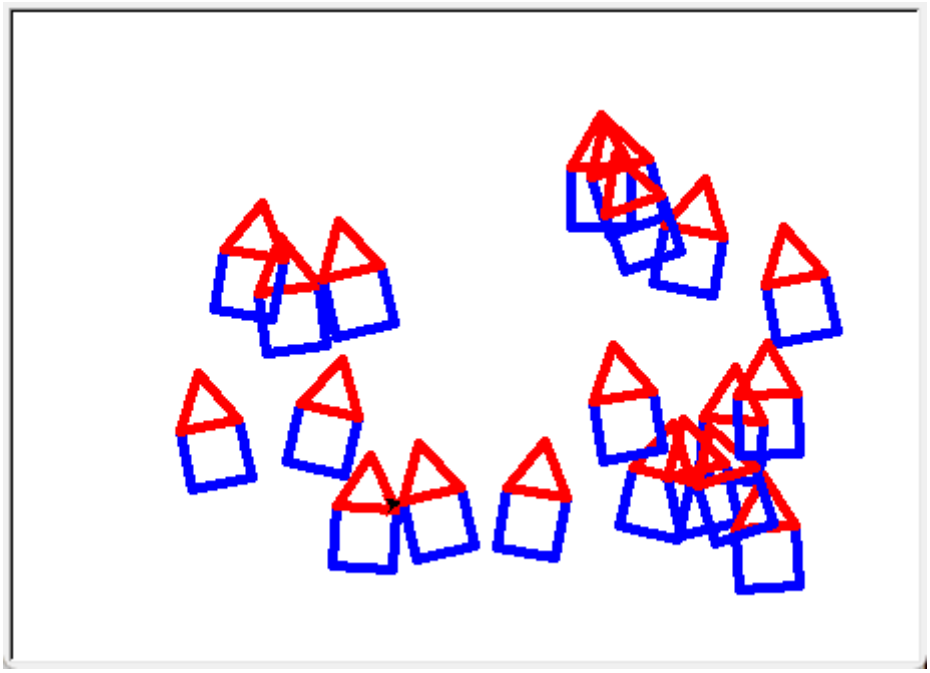
- použili sme tu dva nové príkazy: `setpos(x, y)`, ktorá presunie korytnačku na novú pozíciu a `seth(uhol)` (skratka z `setheading()`), ktorá otočí korytnačku do daného smeru

Grafickému peru korytnačky môžeme meniť hrúbku a farbu:

- príkaz `pencolor(farba)` zmení farbu pera - odteraz bude korytnačka všetko kresliť touto farbou, až kým ju opäť nezmeníme
- príkaz `pensize(hrúbka)` zmení hrúbku pera (celé kladné číslo) - odteraz bude korytnačka všetko kresliť touto hrúbkou, až kým ju opäť nezmeníme

V nasledovnom príklade uvidíme aj príkaz `turtle.delay()`, ktorým môžeme urýchliť (alebo spomaliť) pohyb korytnačky (rýchlosť `turtle.delay(0)` je najrýchlejšia, `turtle.delay(10)` je pomalšia - parameter hovorí počet milisekúnd, ktorým sa zdržuje každé kreslenie). V programe zadefinujeme aj funkciu na kreslenie trojuholníka a potom pomocou funkcií štvorec aj trojuholník nakreslíme dom:

```
import turtle
import random

def stvorec(dlzka):
    for i in range(4):
        t.fd(dlzka)
        t.rt(90)

def trojuholnik(dlzka):
    for i in range(3):
        t.fd(dlzka)
        t.rt(120)

def dom(d):
    t.pencolor('blue')
    stvorec(d)
    t.lt(60)
    t.pencolor('red')
    trojuholnik(d)
    t.rt(60)

def posun():
    t.pu()
    t.setpos(random.randint(-150, 150), random.randint(-100, 100))
    t.seth(random.randint(-30, 30))
    t.pd()

turtle.delay(0)
t = turtle.Turtle()
t.pensize(5)
for i in range(20):
    posun()
    dom(30)
turtle.done()
```



Program na náhodné pozície umiestni modré štvorce, na ktoré ešte položí červené trojuholníky. Zrejme korytnačka je v **globálnej premennej** `t` (v hlavnom mennom priestore) a teda ju vidia všetky naše funkcie.

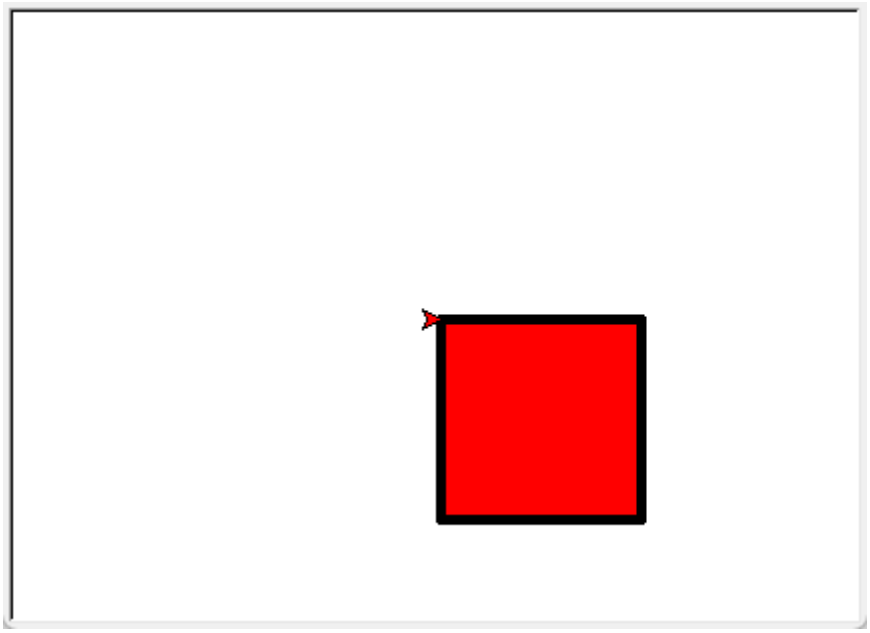
Vyfarbenie útvaru

Útvary, ktoré nakreslí korytnačka sa dajú aj vyfarbiť. Predpokladajme, že korytnačka nakreslí nejaký útvar (napríklad štvorec), a potom ho chceme vyfarbiť nejakou farbou výplne. Princíp fungovania vypĺňania nejakou farbou je takýto:

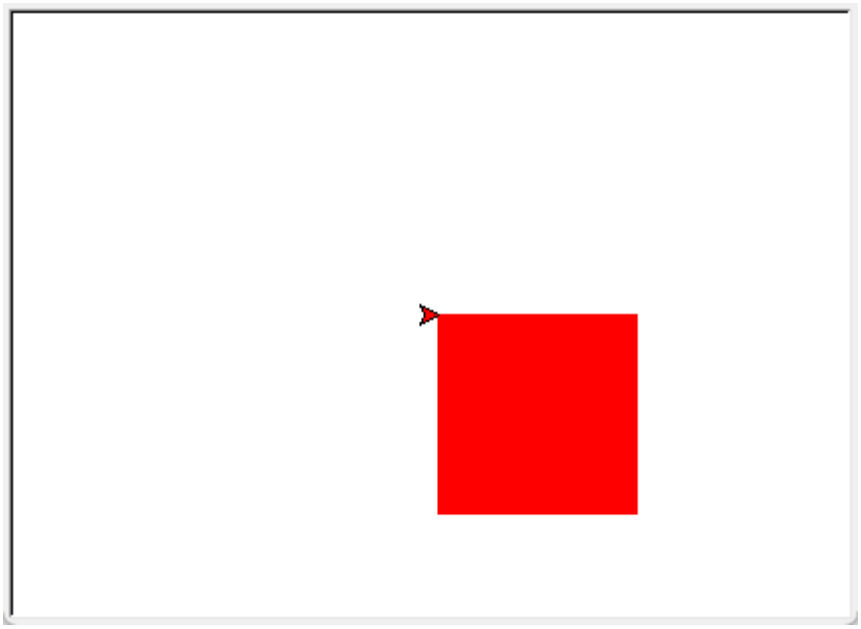
- na začiatok postupnosti korytnačích príkazov, ktoré definujú obrys útvaru, umiestnime príkaz `begin_fill()`
- na koniec tejto postupnosti dáme príkaz `end_fill()`, ktorý vyfarbí nakreslený útvar farbou výplne
- farbu výplne meníme príkazom `fillcolor(farba)` (na začiatku je nastavená čierna farba)
- ak nakreslíme krivku, ktorá netvorí uzavretý útvar, pri vypĺňaní sa táto uzavrie

Ak teraz zadáme:

```
import turtle

def stvorec(dlzka):
    for i in range(4):
        t.fd(dlzka)
        t.rt(90)

t = turtle.Turtle()
t.pensize(5)
t.fillcolor('red')
t.begin_fill()
stvorec(100)
t.end_fill()
turtle.done()
```



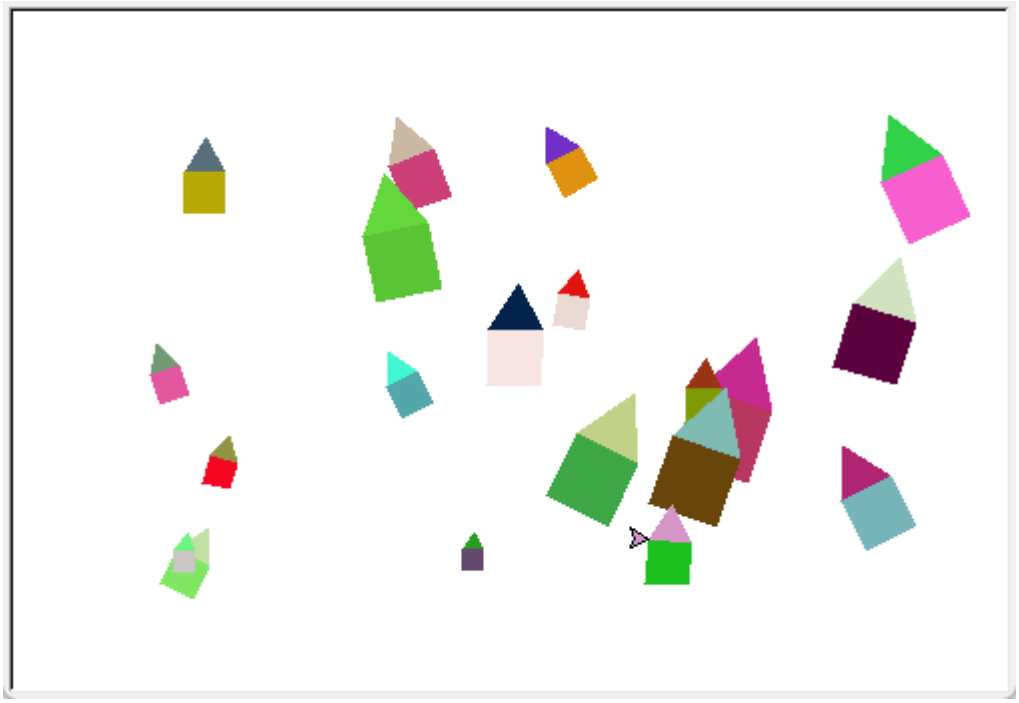
Nakreslí sa štvorec (s čiernym obrysom), ktorý sa vyplní červenou farbou. Všimnite si, že ak by sme štvorec kreslili so zdvihnutým perom, nenakreslil by sa čierny obrys, len by sme dostali štvorec vyfarbený červenou farbou:



Opravme program s kreslením domčekov tak, aby sa nakreslili domčeky s náhodnými farbami štvorca aj trojuholníka:

```
import turtle
import random

def stvorec(dlzka):
    t.fillcolor(f'#{random.randrange(256**3):06x}')
    t.begin_fill()
    for i in range(4):
        t.fd(dlzka)
        t.rt(90)
    t.end_fill()

def trojuholnik(dlzka):
    t.fillcolor(f'#{random.randrange(256**3):06x}')
    t.begin_fill()
    for i in range(3):
        t.fd(dlzka)
        t.rt(120)
```

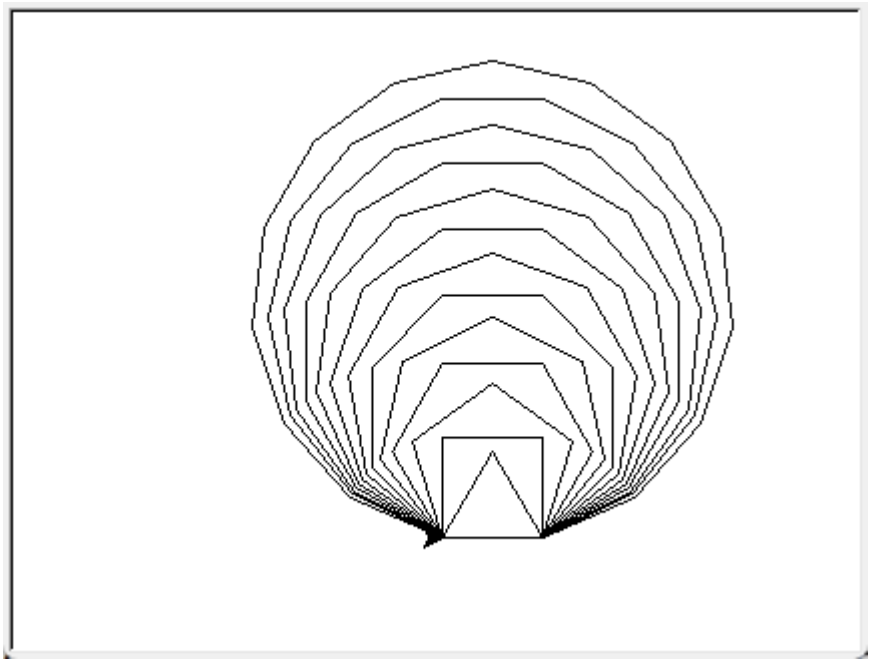
```
    t.end_fill()

def dom(d):
    t.pu()
    stvorec(d)
    t.lt(60)
    trojuholnik(d)
    t.rt(60)

def posun():
    t.pu()
    t.setpos(random.randint(-200, 200), random.randint(-100, 100))
    t.seth(random.randint(-30, 30))
    t.pd()

turtle.delay(0)
t = turtle.Turtle()
for i in range(20):
    posun()
    dom(random.randint(10, 40))
turtle.done()
```



Ďalší príklad predvedie funkciu, ktorá nakreslí ľubovoľný rovnostranný `n`-uholník a tiež príkaz `clear()`, ktorý zmaže nakreslený obrázok, aby sa mohol kresliť ďalší už v prázdnej grafickej ploche:

```
import turtle

def n_uholnik(n, d):
    for i in range(n):
        t.fd(d)
        t.lt(360 / n)

t = turtle.Turtle()
for n in range(3, 16):
    t.clear()
    n_uholnik(n, 50)
turtle.done()
```

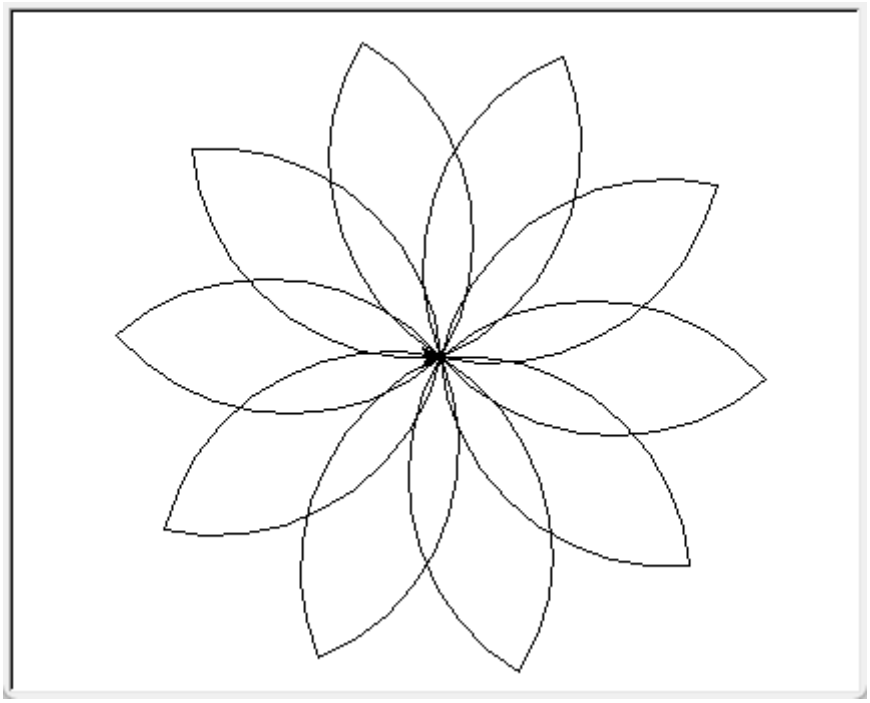
Ak by sme tu vyhodili príkaz `clear()`, mohli by sme v jednej kresbe vidieť všetky n-uholníky:



Pomocou n-uholníkov môžeme nakresliť aj kružnicu (napríklad ako 36-uholník s malou dĺžkou strany), ale aj len časti kružníc, napríklad 18 strán z 36-uholníka nakreslí polkruh, a 9 strán nakreslí štvrťkruh.

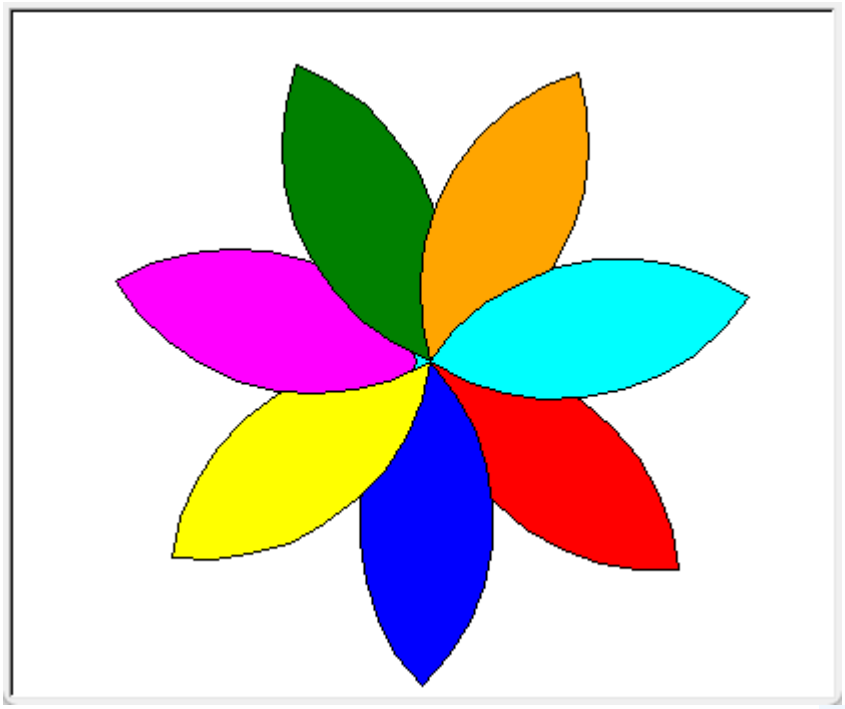
Nasledovný príklad najprv definuje `oblúk` (štvrťkruh), potom `lupen` (dva priložené štvrťkruhy) a nakoniec kvet ako `n` lupeňov:

```
import turtle

def obluk(d):
    for i in range(9):
        t.fd(d)
        t.rt(10)

def lupen(d):
    for i in 1, 2:
        obluk(d)
        t.rt(90)

def kvet(n, d):
    for i in range(n):
        lupen(d)
        t.rt(360 / n)

turtle.delay(0)
t = turtle.Turtle()
kvet(10, 20)
turtle.done()
```



Nakreslíme kvet zložený z farebných lupeňov (každý lupeň bude vyfarbený inou náhodnou farbou):

```
import turtle

def obluk(d):
    for i in range(9):
        t.fd(d)
        t.rt(10)

def lupen(d):
    for i in 1, 2:
        obluk(d)
        t.rt(90)

def kvet(d, farby):
    for f in farby:
        t.fillcolor(f)
        t.begin_fill()
        lupen(d)
        t.end_fill()
        t.rt(360 / len(farby))

turtle.delay(0)
t = turtle.Turtle()
kvet(20, ['red', 'blue', 'yellow', 'magenta', 'green', 'orange', 'cyan])
turtle.done()
```

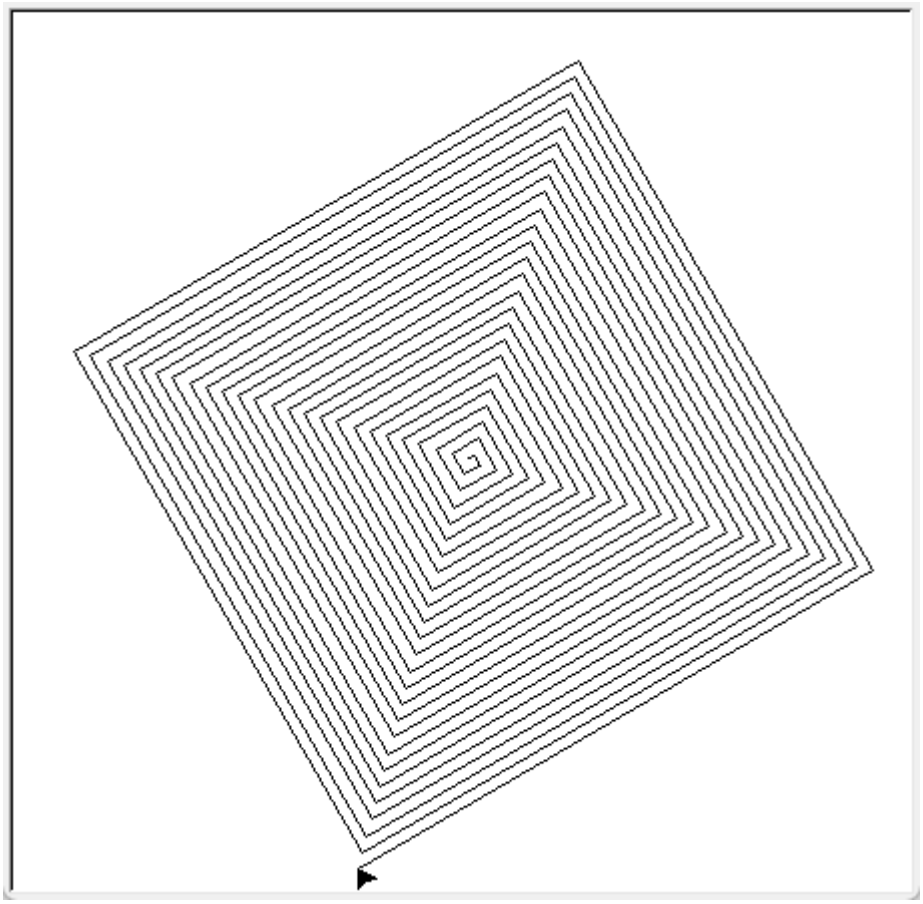


Počet lupeňov kvetu sa tu určuje podľa počtu farieb v zozname `farby`.

Špirály

Rôzne špirály môžeme kresliť tak, že opakujeme kreslenie stále sa zväčšujúcich čiar a za každým sa otočíme o pevný uhol, napríklad:

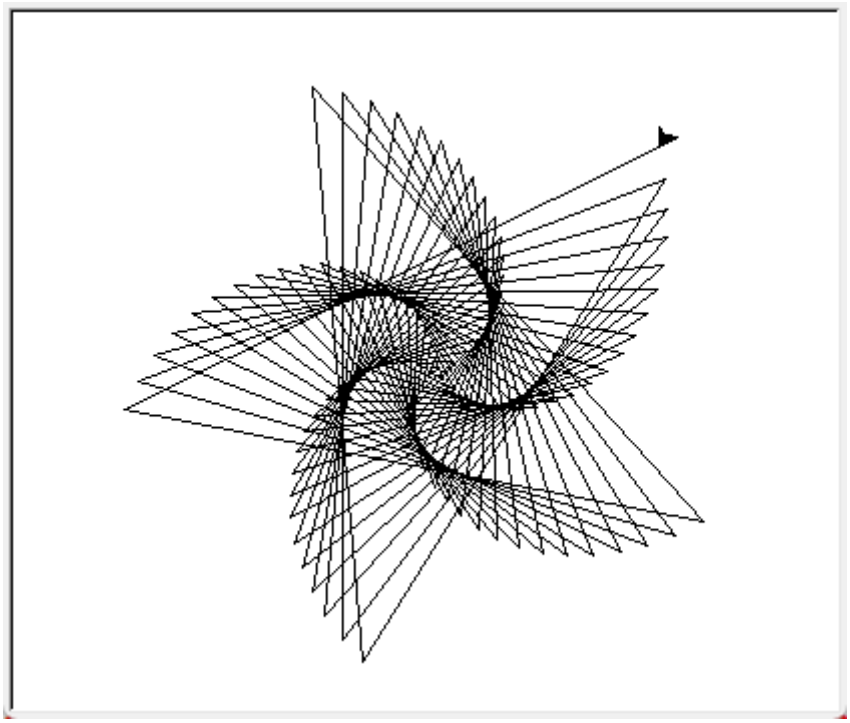
```
import turtle

t = turtle.Turtle()
t.lt(30)
for i in range(3, 300, 3):
    t.fd(i)
    t.rt(90)
turtle.done()
```



Program nakreslí štvorcovú špirálu.

S uhlami môžete experimentovať, napríklad:

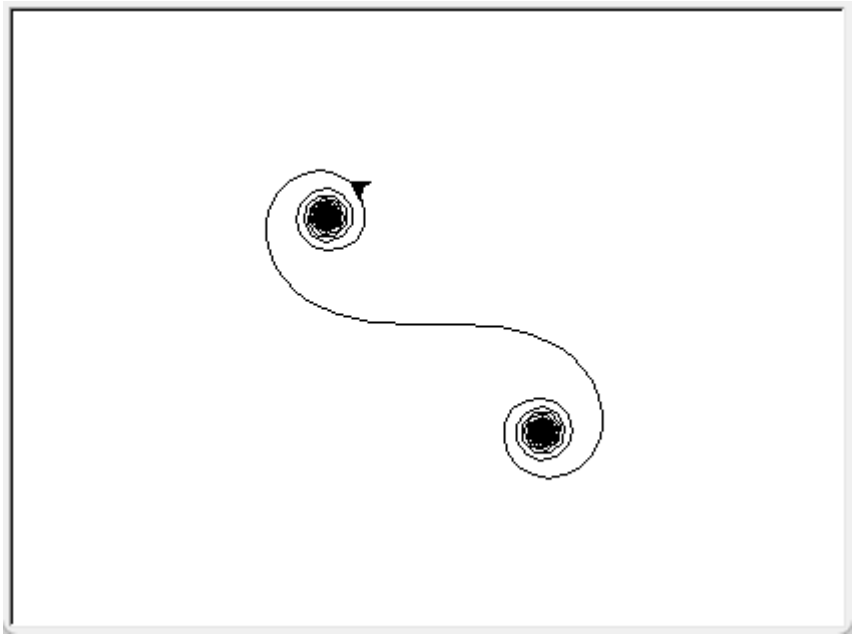
```
import turtle
import random

turtle.delay(0)
t = turtle.Turtle()
while True:
    uhol = random.randint(30, 170)
    print('spirala s uhlom', uhol)
    for i in range(3, 300, 3):
        t.fd(i)
        t.rt(uhol)
    t.reset()
# turtle.done()
```



Tento program kreslí špirály s rôznymi náhodne generovanými uhlami. Zároveň do textovej plochy vypisuje informáciu o uhle momentálne kreslenej špirály. Všimnite si, že vonkajší cyklus, v ktorom sa kreslia špirály, je nekonečný.

Zaujímavé špirály vznikajú, keď nemeníme dĺžku čiar ale uhol, napríklad:

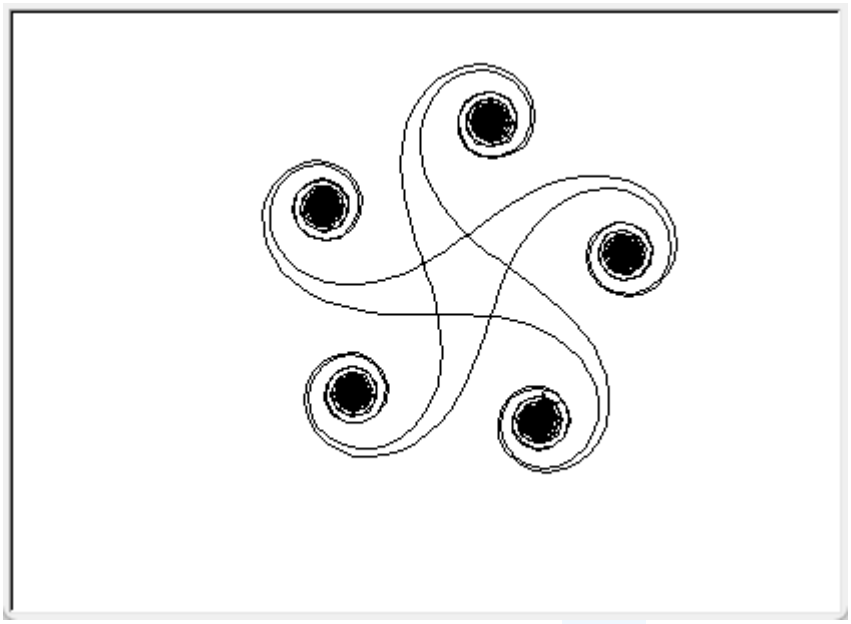
```
import turtle

turtle.delay(0)
t = turtle.Turtle()
for uhol in range(1, 700):
    t.fd(8)
    t.rt(uhol)
turtle.done()
```



Tu môžeme vyskúšať rôzne malé zmeny uhla, o ktorý sa mení kreslenie čiar útvaru:

```
import turtle

turtle.delay(0)
t = turtle.Turtle()
```

```
for uhol in range(1, 2000):
    t.fd(8)
    t.rt(uhol + 0.1)
turtle.done()
```



Vyskúšajte rôzne iné zmeny uhla v príkaze `t.rt()`.

Zhrnutie užitočných metód

metóda	variant	význam	príklad
`forward(d)`	`fd`	choď dopredu	`t.fd(100); t.fd(-50)`
`back(d)`	`backward`, `bk`	cúvaj	`t.bk(50); t.bk(-10)`
`right(u)`	`rt`	otoč sa vpravo	`t.rt(90); t.rt(-120)`
`left(u)`	`lt`	otoč sa vľavo	`t.lt(90); t.lt(-45)`
`penup()`	`pu`, `up`	zdvihni pero	`t.pu()`
`pendown()`	`pd`, `down`	spusti pero	`t.pd()`
`setpos(x, y)`	`setposition`, `goto`	choď na pozíciu	`t.setpos(50, 70)`
`pos()`	`position`	zisti pozíciu korytnačky	`t.pos()`
`xcor()`		zisti x-ovú súradnicu	`t.xcor()`
`ycor()`		zisti y-ovú súradnicu	`t.ycor()`
`heading()`		zisti uhol korytnačky	`t.heading()`
`setheading(u)`	`seth`	nastav uhol korytnačky	`t.seth(120)`
`pensize(h)`	`width`	nastav hrúbku pera	`t.pensize(5)`

metóda	variant	význam	príklad
`pencolor(f)`		nastav farbu pera	`t.pencolor('red')`
`pencolor()`		zisti farbu pera	`t.pencolor()`
`fillcolor(f)`		nastav farbu výplne	`t.fillcolor('blue')`
`fillcolor()`		zisti farbu výplne	`t.fillcolor()`
`color(f1, f2)`		nastav farbu pera aj výplne	`t.color('red'); t.color('blue', 'white`
`color()`		zisti farbu pera aj výplne	`t.color()`
`reset()`		zmaž kresbu a inicializuj korytnačku	`t.reset()`
`clear()`		zmaž kresbu	`t.clear()`
`begin_fill()`		začiatok budúceho vyfarbenia	`t.begin_fill()`
`end_fill()`		koniec vyfarbenia	`t.end_fill()`

Globálne korytnačie funkcie

Modul `turtle` má ešte tieto funkcie, ktoré robia globálne nastavenia a zmeny (majú vplyv na všetky korytnačky):

- `turtle.delay(číslo)` - vykonávanie korytnačích metód sa spomalí na zadaný počet milisekúnd (štandardne je 10)
  - každú jednu korytnačku môžeme ešte individuálne zrýchľovať alebo spomaľovať pomocou `t.speed(číslo)`, kde `číslo` je od 0 do 10 (0 najrýchlejšie, štandardne je 3)
- `turtle.tracer(číslo)` - zapne alebo vypne priebežné zobrazovanie zmien v grafickej ploche (štandardne je `číslo` 1):
  - `turtle.tracer(0)` - vypne zobrazovanie zmien, t. j. teraz je vykresľovanie veľmi rýchle bez pozdržiavania, ale zatiaľ žiadnu zmenu v grafickej ploche nevidíme
  - `turtle.tracer(1)` - zapne zobrazovanie zmien, t. j. teraz je vykresľovanie už pomalé (podľa nastavených `turtle.delay()` a `t.speed()`), lebo vidíme všetky zmeny kreslenia v grafickej ploche
- `turtle.bgcolor(farba)` - zmení farbu pozadia grafickej plochy, pričom všetky kresby v ploche ostávajú bez zmeny

Tvar korytnačky

Korytnačkám môžeme meniť ich tvar - momentálne je to malý trojuholník.

Príkaz `shape()` zmení tvar na jeden s preddefinovaných tvarov (pre korytnačku `t`):

```
t.shape('arrow')        # tvarom korytnačky bude šípka
t.shape('turtle')       # tvarom korytnačky bude korytnačka
t.shape('circle')       # tvarom korytnačky bude kruh
t.shape('square')       # tvarom korytnačky bude štvorec
t.shape('triangle')     # tvarom korytnačky bude trojuholník
t.shape('classic')      # tvarom korytnačky bude hrot šípky
```

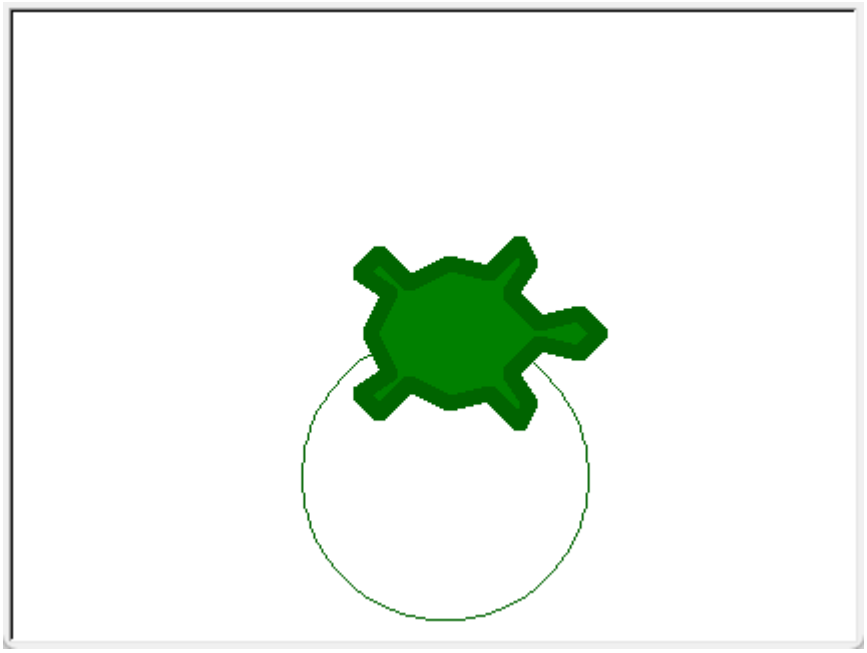
Default tvar je `'classic'`.

Príkaz `shapesize()` nastavuje zväčšenie tvaru a hrúbku obrysu tvaru (pre korytnačku `t`):

```
t.shapesize(sirka, vyska, hrubka)
```

Mohli ste si všimnúť, že keď korytnačke zmeníte farbu pera, zmení sa obrys jej tvaru. Podobne, keď sa zmení farba výplne, tak sa zmení aj výplň tvaru korytnačky. Napríklad:

```
import turtle

t = turtle.Turtle()
t.shape('turtle')
t.shapesize(5, 5, 8)
t.color('darkgreen', 'green')
for i in range(90):
    t.fd(5)
    t.rt(4)
turtle.done()
```



V tomto príklade sa nastaví korytnačke zväčšený tvar a pomaly nakreslí kružnicu (90-uholník).

Zobrazovanie tvaru korytnačky môžeme skryť príkazom `hideturtle()` (skratka `ht()`) a opätovné zobrazovanie zapnúť príkazom `showturtle()` (skratka `st()`).

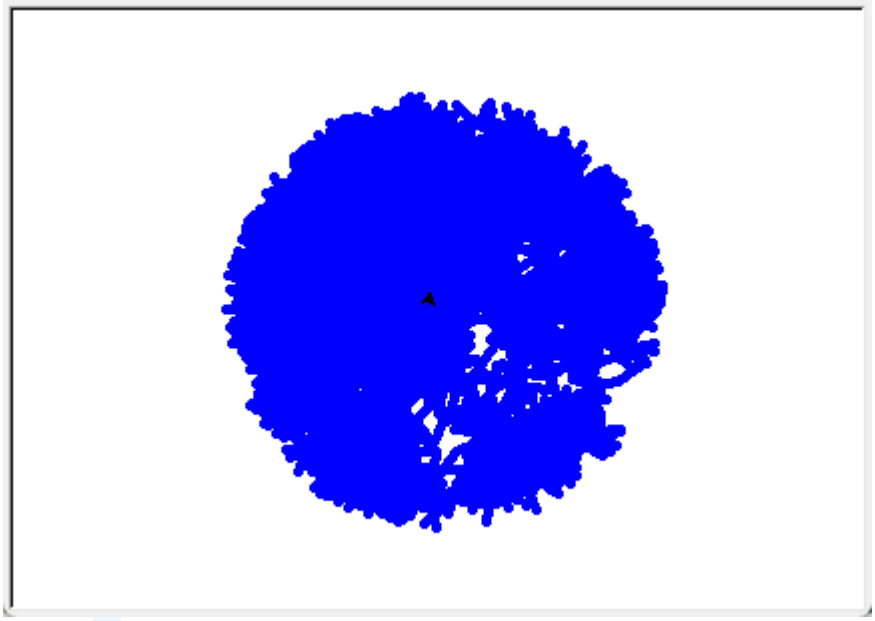
Náhodné prechádzky

Náhodnými prechádzkami budeme nazývať taký pohyb korytnačky, pri ktorom sa korytnačka veľa-krát náhodne otočí a prejde nejakú malú vzdialenosť. Napríklad:

```
import turtle
import random

turtle.delay(0)
t = turtle.Turtle()
for i in range(10000):
    t.seth(random.randint(0, 359))
    t.fd(10)
turtle.done()
```

Po čase odíde z grafickej plochy - upravme to tak, aby nevyšla z nejakej konkrétnej oblasti, napríklad:

```
import turtle
import random

turtle.delay(0)
t = turtle.Turtle()
t.pensize(5)
t.pencolor('blue')
for i in range(10000):
    t.seth(random.randint(0, 359))
    t.fd(10)
    if t.xcor()**2 + t.ycor()**2 > 50**2:
        t.fd(-10)
turtle.done()
```

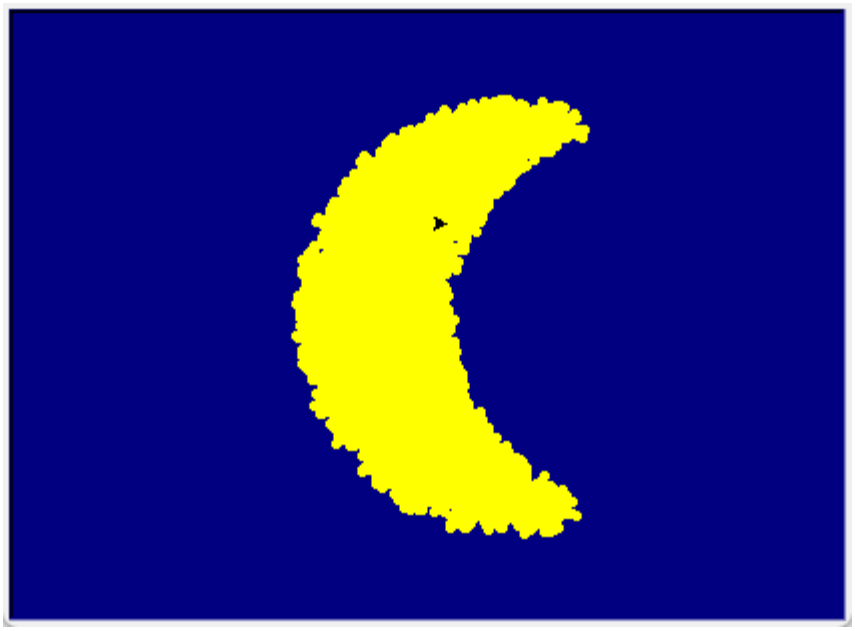


Príkaz `if` stráži korytnačku: keď sa vzdiali od (0,0) viac ako 50, vráti ju späť - počítali sme tu vzdialenosť korytnačky od počiatku.

Môžeme využiť metódu `distance()`, ktorá vypočíta vzdialenosť korytnačky od nejakého bodu alebo inej korytnačky:

```
import turtle
import random
```

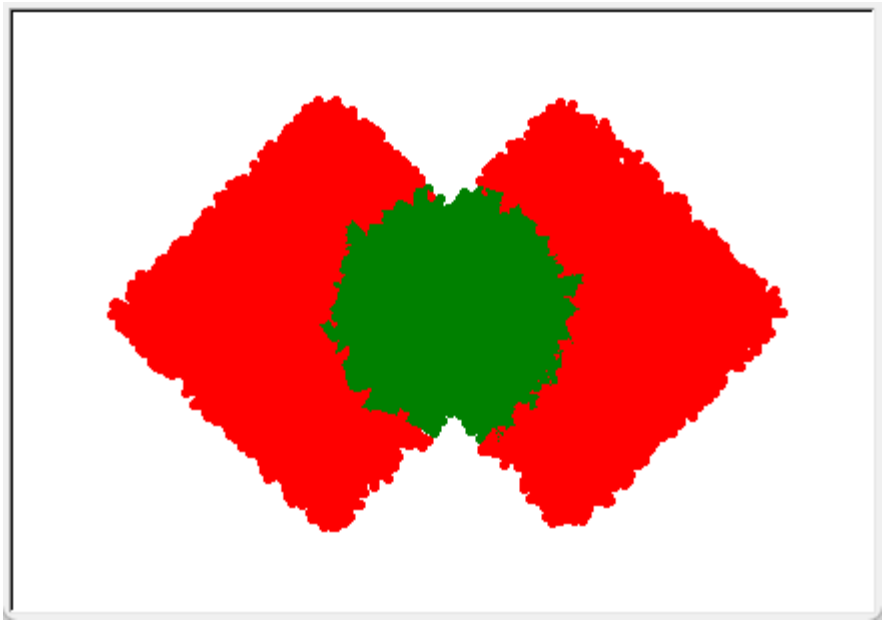
```
turtle.delay(0)
t = turtle.Turtle()
turtle.bgcolor('navy')
t.pensize(5)
t.pencolor('yellow')
for i in range(10000):
    t.seth(random.randint(0, 359))
    t.fd(10)
    if t.distance(40, 0) > 100 or t.distance(100, 0) < 100:
        t.fd(-10)
turtle.done()
```



Korytnačka sa teraz pohybuje v oblasti, ktorá ma tvar mesiaca: nesmie vyjsť z prvého kruhu a zároveň vojsť do druhého.

Tvar stráženej oblasti môže byť definovaný aj zložitejšou funkciou, napríklad:

```
import turtle
import random

def fun(pos):
    x, y = pos               # pos je dvojica súradníc
    if abs(x - 60) + abs(y) < 100:
        return False
    return abs(x + 60) + abs(y) > 100

turtle.delay(0)
t = turtle.Turtle()
t.speed(0)
t.pensize(5)
for i in range(10000):
    t.seth(random.randint(0, 359))
    if t.distance(0, 0) < 60:
        t.pencolor('green')
    else:
        t.pencolor('red')
    t.fd(10)
    if fun(t.pos()):       # funkcia fun stráži nejakú oblasť
        t.fd(-10)
turtle.done()
```



Okrem stráženia oblasti tu meníme farbu pera podľa nejakej podmienky

Viac korytnačiek

Doteraz sme pracovali len s jednou korytnačkou (vytvorili sme ju pomocou `t = turtle.Turtle()`). Korytnačiek ale môžeme vytvoriť ľubovoľne veľa. Aby rôzne korytnačky mohli využívať tú istú kresliacu funkciu (napríklad `stvorec()`) musíme globálnu premennú `t` vo funkcii prerobiť na parameter (v našom príklade sme ho nazvali `tu`):

```
import turtle

def stvorec(tu, velkost):
    for i in range(4):
        tu.fd(velkost)
        tu.rt(90)

t = turtle.Turtle()
stvorec(t, 100)
turtle.done()
```

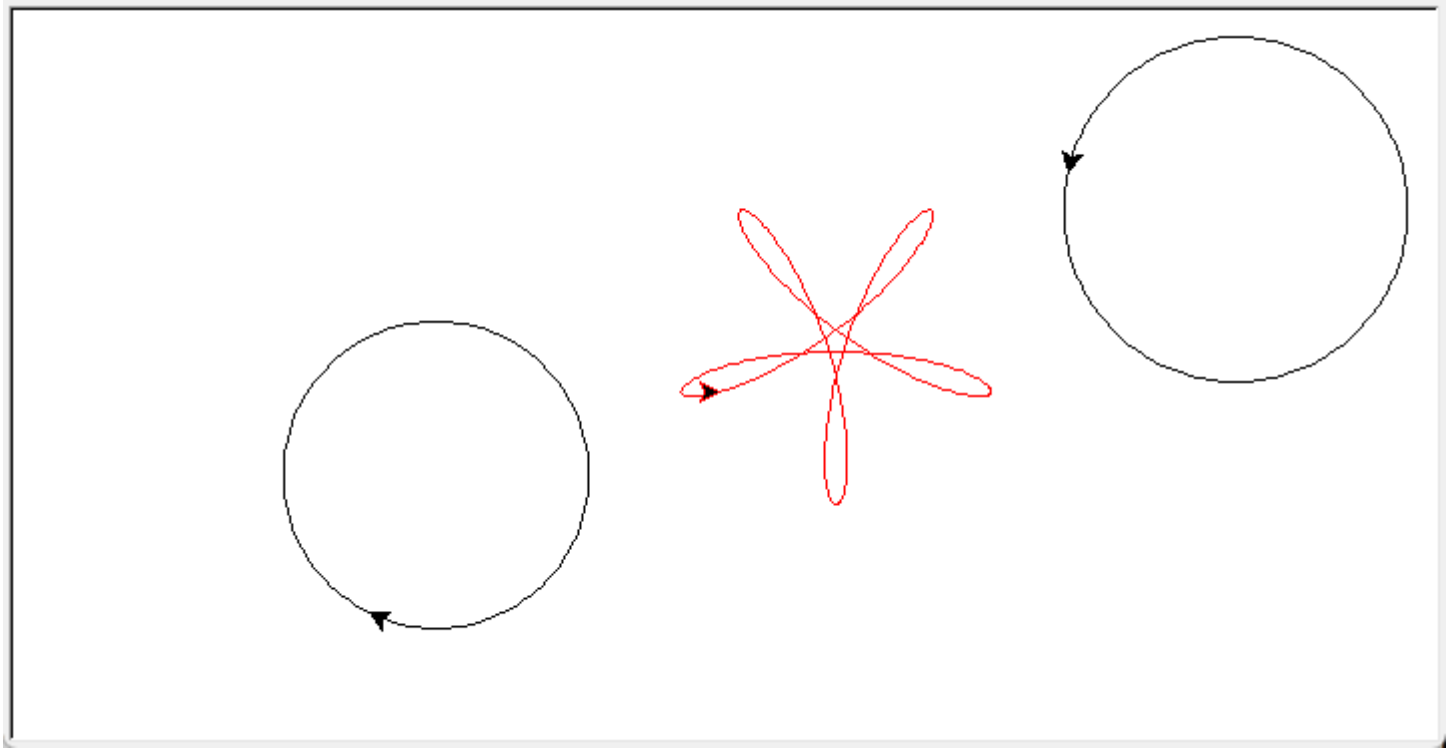
Vytvorme ďalšiu korytnačku a necháme ju tiež kresliť štvorce tou istou funkciou:

```
t1 = turtle.Turtle()
t1.lt(30)
for i in range(5):
    stvorec(t1, 50)
    t1.lt(72)
turtle.done()
```

Kým sme vytvárali funkcie, ktoré pracovali len pre jednu korytnačku, nemuseli sme ju posielať ako parameter. Ak ale budeme potrebovať funkcie, ktoré by mali pracovať pre ľubovoľné ďalšie korytnačky, vytvoríme vo funkcii nový parameter (najčastejšie ako prvý parameter funkcie) a ten bude v tele funkcie zastupovať tú korytnačku, ktorú do funkcie pošleme.

V ďalšom príklade vyrobíme 3 korytnačky: 2 sa pohybujú po nejakej stálej trase a tretia sa vždy nachádza presne v strede medzi nimi (ako keby bola v strede gumenej nite):

```
import turtle

def posun(k, pos):     # pos je pozícia v tvare (x, y)
    k.pu()
    k.setpos(pos)
    k.pd()

def stred(k1, k2):
    x = (k1.xcor() + k2.xcor()) / 2
    y = (k1.ycor() + k2.ycor()) / 2
    return (x, y)

turtle.delay(0)
t1 = turtle.Turtle()
posun(t1, (-150, 30))
t2 = turtle.Turtle()
posun(t2, (250, 0))
t3 = turtle.Turtle()
posun(t3, stred(t1, t2))
t3.pencolor('red')

while True:
    t1.fd(4)
    t1.rt(3)
    t2.fd(3)
    t2.lt(2)
    t3.setpos(stred(t1, t2))
# turtle.done()
```



Zoznam korytnačiek

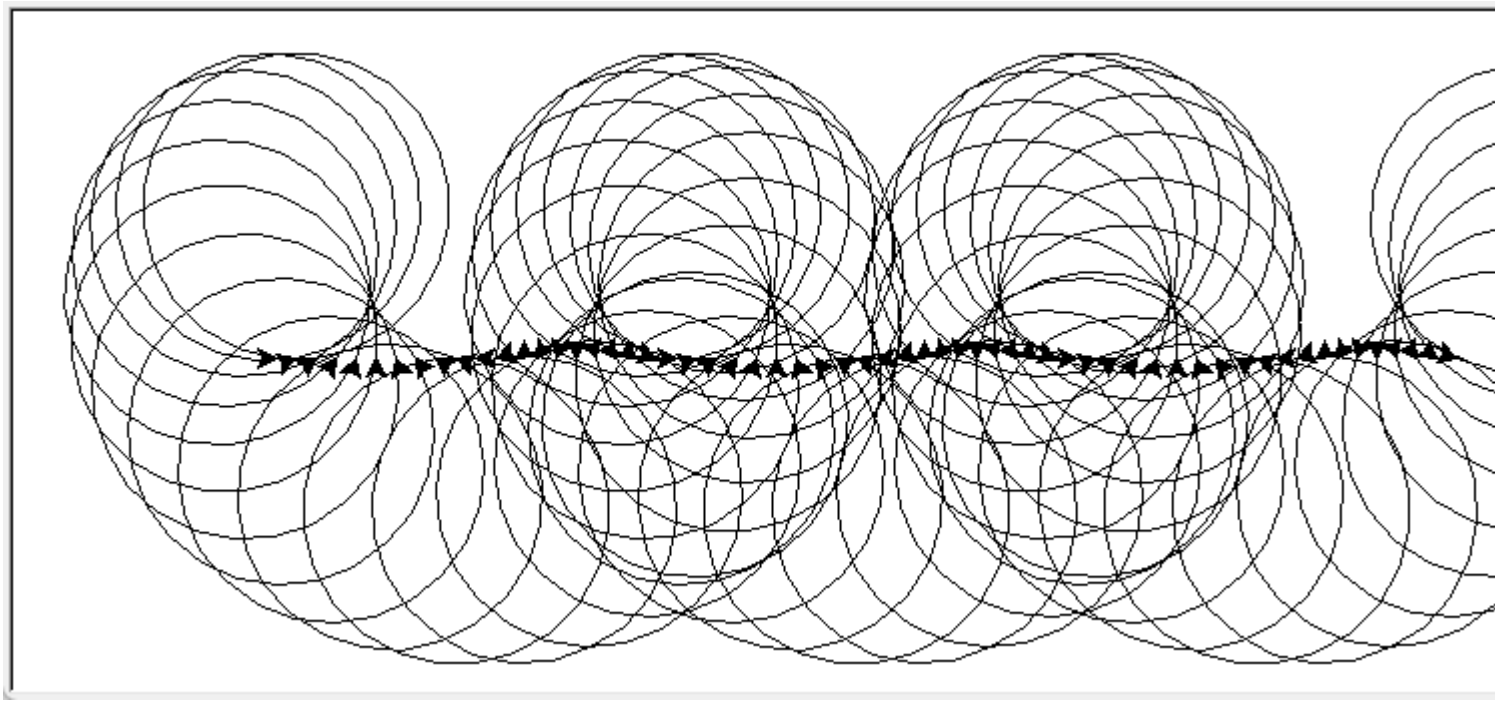
Do premennej typu zoznam postupne priradíme vygenerované korytnačky, pričom každú presunieme na inú pozíciu (všetky ležia na x-ovej osi) a nastavíme jej iný smer:

```
import turtle

turtle.delay(0)
```

```
zoznam = []
for i in range(60):
    t = turtle.Turtle()
    t.pu()
    t.setpos(-300 + 10*i, 0)
    t.pd()
    t.seth(i * 18)
    zoznam.append(t)
turtle.done()
```

Necháme ich kresliť rovnaké kružnice:

```
import turtle

turtle.delay(0)
zoznam = []
for i in range(60):
    t = turtle.Turtle()
    t.pu()
    t.setpos(-300 + 10*i, 0)
    t.pd()
    t.seth(i * 18)
    zoznam.append(t)

for t in zoznam:
    for i in range(24):
        t.fd(20)
        t.lt(15)
turtle.done()
```



Takto kreslila jedna za druhou: ďalšia začala kresliť až vtedy, keď predchádzajúca skončila.

Pozmeňme to tak, aby všetky kreslili naraz:

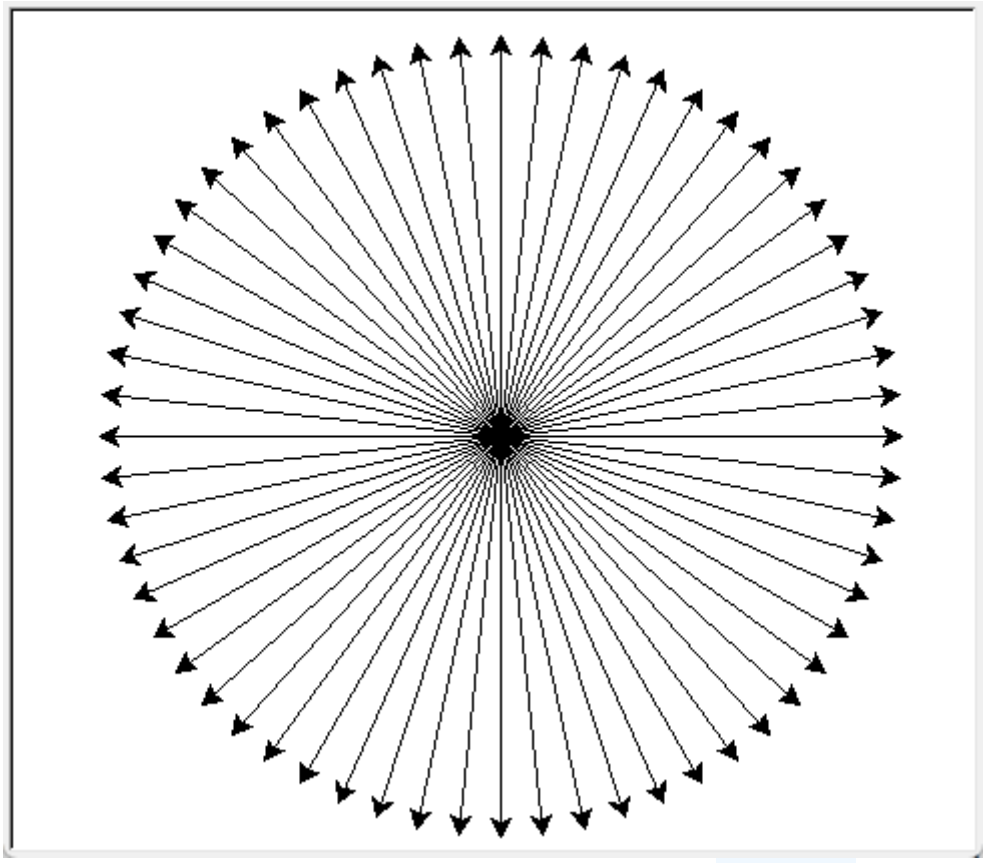
```
import turtle

turtle.delay(0)
zoznam = []
for i in range(60):
    t = turtle.Turtle()
    t.pu()
```

```
    t.setpos(-300 + 10*i, 0)
    t.pd()
    t.seth(i * 18)
    zoznam.append(t)

for i in range(24):
    for t in zoznam:
        t.fd(20)
        t.lt(15)
turtle.done()
```

Tu sme zmenili len poradie for-cyklov.

V ďalšom príklade vygenerujeme všetky korytnačky v počiatku súradnej sústavy ale s rôznymi smermi a necháme ich prejsť dopredu rovnakú vzdialenosť:

```
import turtle

turtle.delay(0)
zoznam = []
for i in range(60):
    zoznam.append(turtle.Turtle())
    zoznam[-1].seth(i * 6)

for t in zoznam:
    t.fd(200)
turtle.done()
```



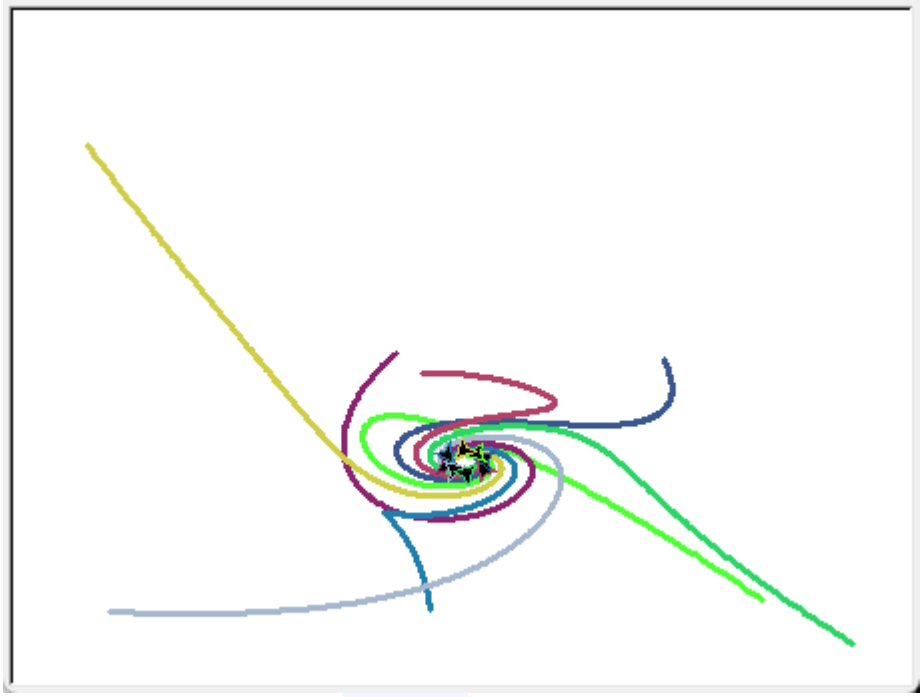
Všimnite si, ako pracujeme so zoznamom korytnačiek (`zoznam[-1]` označuje posledný prvok zoznamu, t. j. naposledy vygenerovanú korytnačku).

Korytnačky sa naháňajú

Na náhodných pozíciách vygenerujeme `n` korytnačiek a potom ich necháme, nech sa naháňajú podľa takýchto pravidiel:

- každá sa otočí smerom k nasledovnej (prvá k druhej, druhá k tretej, …, `n`-tá k prvej)
- každá prejde stotinu vzdialenosti k nasledovnej

```
import turtle
import random

n = 8
t = []
turtle.delay(0)
for i in range(n):
    nova = turtle.Turtle()
    nova.pu()
    nova.setpos(random.randint(-200, 200), random.randint(-200, 200))
    nova.pencolor(f'#{random.randrange(256**3):06x}')
    nova.pensize(3)
    nova.pd()
    t.append(nova)

while True:
    for i in range(n):
        j = (i+1) % n               # index nasledovnej
        uhol = t[i].towards(t[j])
        t[i].seth(uhol)
        vzdialenost = t[i].distance(t[j])
        t[i].fd(vzdialenost / 100)
# turtle.done()
```

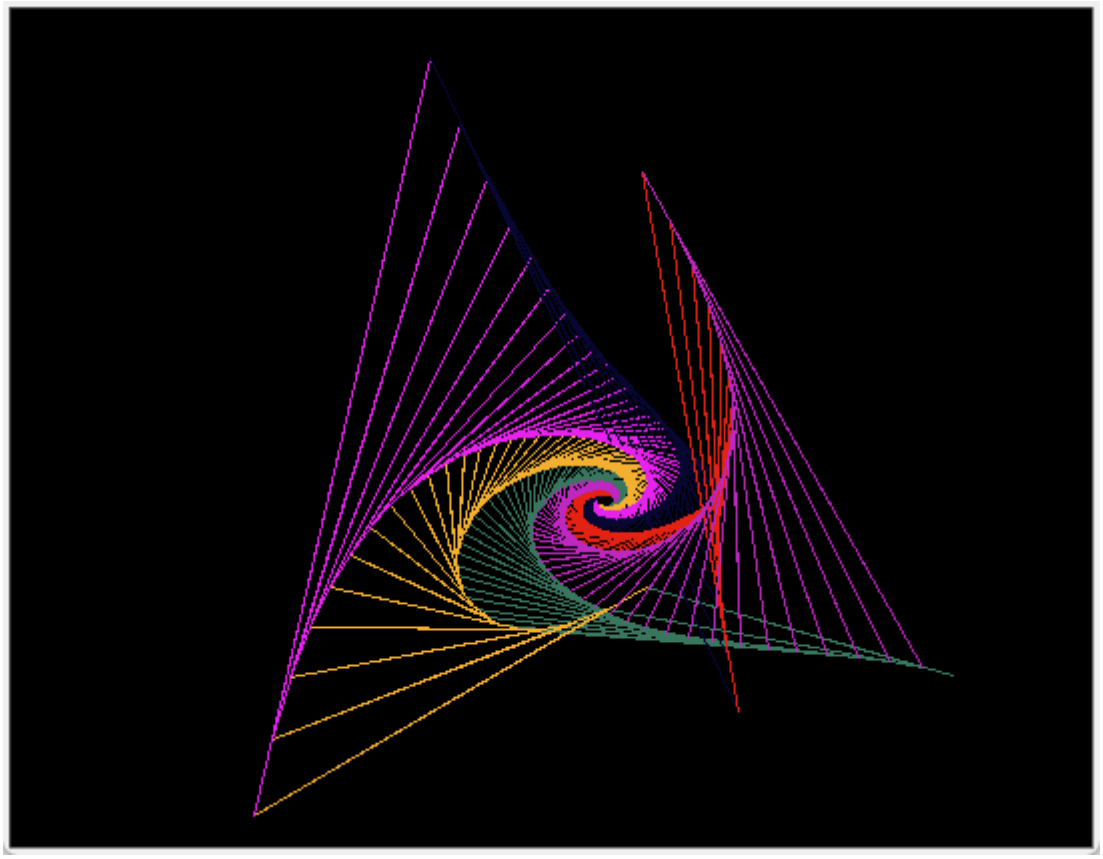


Využili sme novú metódu `towards()`, ktorá vráti uhol otočenia k nejakému bodu alebo k pozícii inej korytnačky.

Trochu pozmeníme: okrem prejdenia `1/10` vzdialenosti k nasledovnej nakreslí aj celú spojnicu k nasledovnej:

```
import turtle
import random

turtle.delay(0)
while True:
    turtle.bgcolor('black')
```

```
    n = random.randint(3, 8)
    t = []
    for i in range(n):
        nova = turtle.Turtle()
        nova.speed(0)
        nova.pu()
        nova.setpos(random.randint(-200, 200), random.randint(-200, 200))
        nova.pencolor(f'#{random.randrange(256**3):06x}')
        nova.pd()
        nova.ht()
        t.append(nova)

    for k in range(100):
        for i in range(n):
            j = (i+1) % n                # index nasledovnej
            uhol = t[i].towards(t[j])
            t[i].seth(uhol)
            vzdialenost = t[i].distance(t[j])
            t[i].fd(vzdialenost)
            t[i].fd(vzdialenost/10 - vzdialenost)

    for tt in t:
        tt.clear()
# turtle.done()
```



Po dokreslení, obrázok zmaže a začne kresliť nový. Uvedomte si, že každý ďalší prechod while-cyklu vytvorí nové a nové korytnačky a tie pôvodné staré tam ostávajú, hoci majú skrytý tvar a teda ich nevidíme. Ak by sme naozaj chceli zrušiť korytnačky z grafickej plochy, tak namiesto záverečného cyklu:

```
for tt in t:
    tt.clear()
```

by sme mali zapísať:

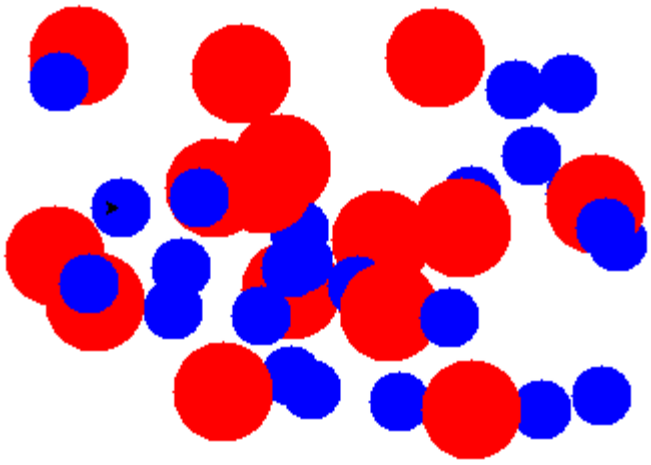
```
turtle.getscreen().clear()
```

Cvičenia

L.I.S.T.

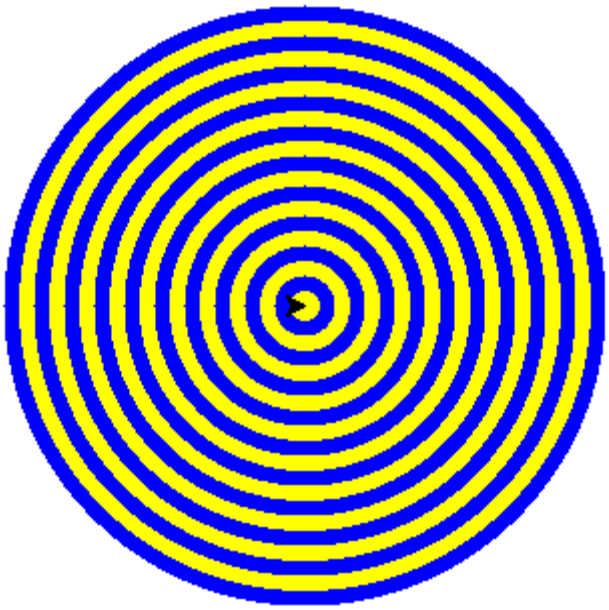
- riešenia **aspoň 12 úloh** odovzdaj na úlohový server https://list.fmph.uniba.sk/
- pozri si Riešenie úloh 11. cvičenia

1. Napíš funkciu `bodka(velkost, farba)`, ktorá na pozícii korytnačky nakreslí farebnú bodku danej veľkosti.
    - bodku budeš kresliť takto: zmení hrúbku a farbu pera na parametrami zadané hodnoty a potom nakreslí čiaru dĺžky `0`
    - funkciu otestuj nakreslením 100 bodiek na náhodných pozíciách, pričom v pomere 1:2 budú červené veľkosti 50 ku modrým veľkosti 30 (v cykle generuj náhodné čísla `randrange(3)` a pri 0 kresli väčšie červené a inak menšie modré)
    - v tomto testovacom programe nahraď volanie tvojej funkcie `bodka(...)` na korytnačiu metódu `t.dot(...)` - mal by si dostať rovnaký výsledok (môžeš poštudovať `help(t.dot)`); skontroluj, či táto metóda `dot` kreslí bodku aj pri zdvihnutom pere



2. Napíš funkciu `slnko(pocet, velkost)`, pomocou ktorej korytnačka nakreslí slnko s daným počtom lúčov. Každý lúč je úsečka danej veľkosti a hrúbky `10`. Okrem lúčov nakresli kruh danej veľkosti (priemer kruhu je tiež `velkost`) - použi metódu `t.dot(...)`. Na kreslenie lúčov aj slnka použi farbu `'gold'`. Mal by si dostať:

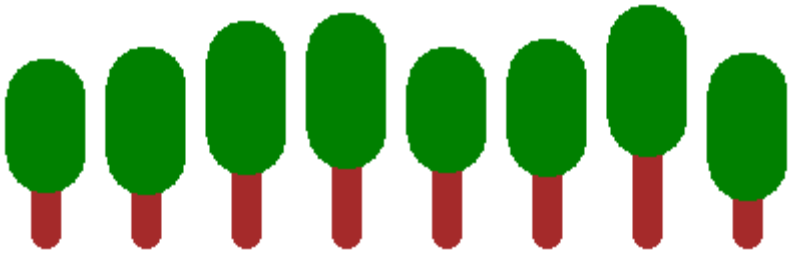
3. Napíš funkciu `terc(pocet)`, ktorá pomocou korytnačej metódy `t.dot(...)` nakreslí zadaný počet rôzne veľkých bodiek: najmenšia má veľkosť `15`, každá ďalšia je o `15` väčšia. Najväčšia bodka nech je modrá, menšia žltá a takto sa ďalej striedajú tieto dve farby na všetky bodky. Otestuj napríklad `terc(40)`:



4. Napíš funkciu `strom(kmen, koruna)`, ktorá nakreslí strom s hnedým kmeňom (hrúbka 15) a zelenou korunou (hrúbka 40). Po zavolaní:

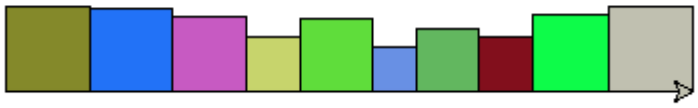
```
for i in range(8):
    strom(random.randint(30, 60), random.randint(10, 40))
    t.pu()
    t.fd(50)
    t.pd()
```

by sa malo nakresliť:

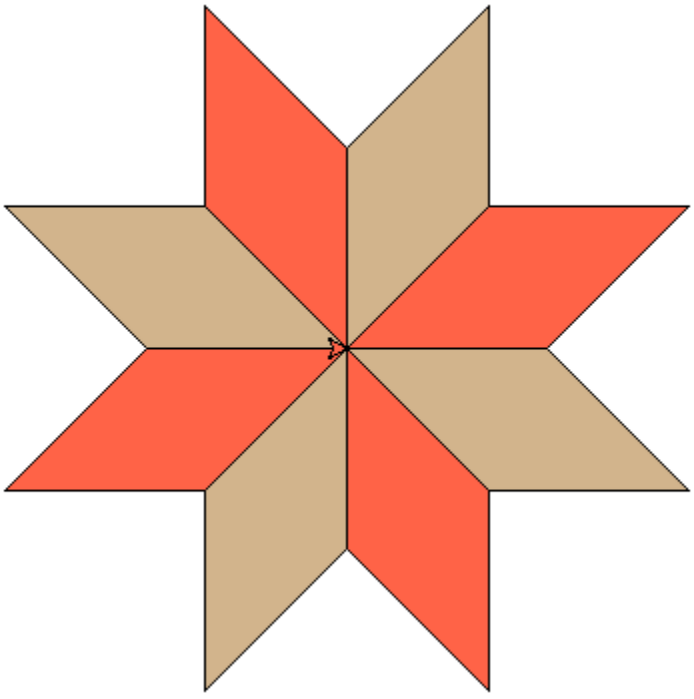


5. Napíš funkciu `stvorec(velkost)`, pomocou ktorej korytnačka nakreslí náhodne zafarbený štvorec danej veľkosti. Otestuj nakreslením 10 štvorcov v rade vedľa seba s náhodnými veľkosťami strán od `20` do `50`. Nasledovný obrázok vznikol takýmto volaním:

```
for i in range(10):
    stvorec(random.randint(20, 50))
```



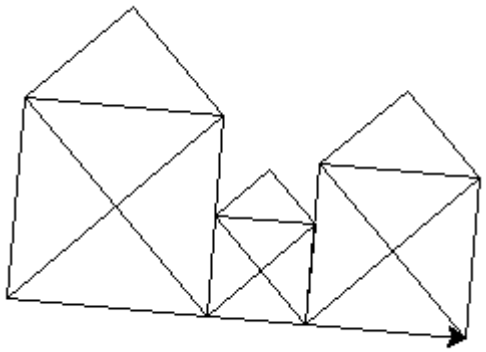
6. Napíš funkciu `kosostvorec(velkost, farba)`, pomocou ktorej korytnačka nakreslí kosoštvorec s danou veľkosťou strany. Vnútorný uhol nech je `45` stupňov. Korytnačka začína aj končí pri vrchole, ktorý má týchto 45 stupňov. Vnútro kosoštvorca vyfarbi danou farbou. Otestuj nakreslením 8 kosoštvorcov, ktoré majú jeden spoločný vrchol; týmto kosoštvorcom pravidelne striedaj dve farby výplne `'tan'` a `'tomato'`. Mal by si dostať:



7. Napíš funkciu `domcek(d)`, ktorá nakreslí domček **jedným ťahom**, bez dvíhania pera a bez `setpos`, teda pomocou 8 úsečiek (príkazov `forward`). Veľkosť štvorca je `d`, všetky vnútorné uhly sú `45` stupňov. Vyskúšaj:

```
t.rt(5)
domcek(100)
domcek(50)
domcek(80)
```

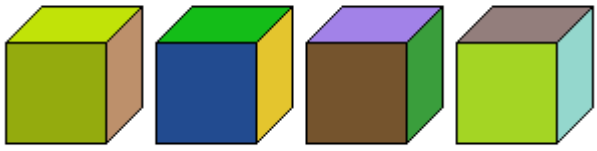
dostaneš:



8. Napíš funkciu `kocka(d)`, ktorá nakreslí 3D kocku. Strana štvorca má veľkosť `d`, tri šikmé čiary sú pod uhlom 45 a majú dĺžku `d/2`. Všetky tri štvoruholníky sú zafarbené náhodnými farbami. Nepoužívaj metódu `setpos`. Vyskúšaj:

```
for i in range(4):
    kocka(50)
    t.pu()
    t.fd(75)
    t.pd()
```

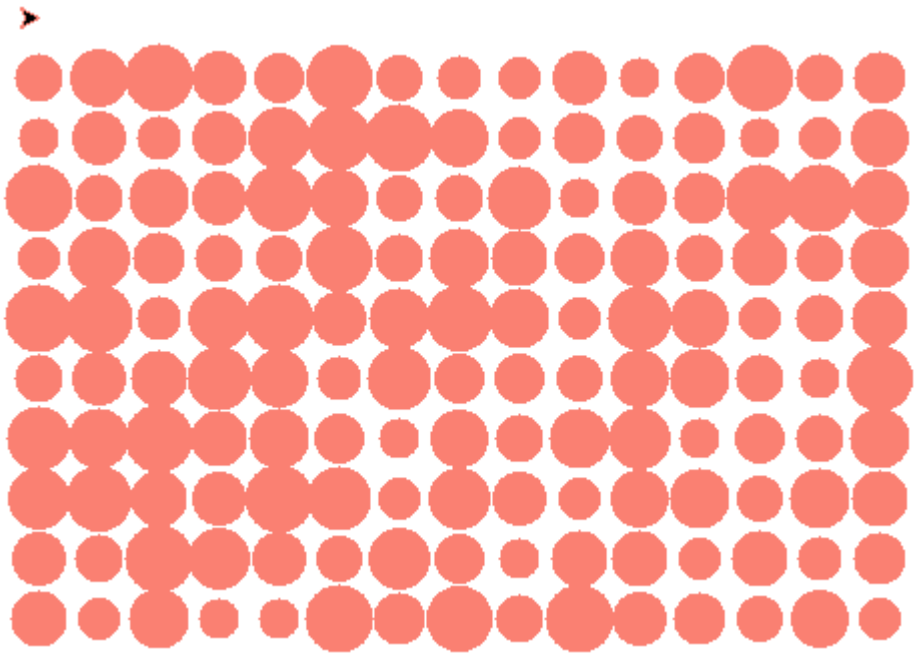
dostaneš:



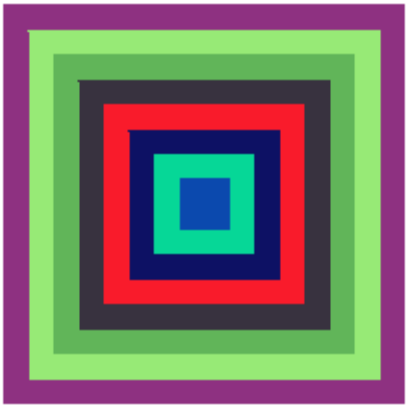
9. Napíš funkciu `polkruznica(velkost, smer)`, pomocou ktorej korytnačka nakreslí 18 strán z pravidelného 36-uholníka so stranou danej veľkosti. Parameter `smer` určuje, či sa pri kreslení korytnačka otáča vľavo (`smer=True`) alebo vpravo (`smer=False`). Otestuj nakreslením 10 polkružníc tak, aby z toho vznikli vlnky (veľkosť nech je napríklad `3`). Mal by si dostať:

10. Na rovnakom princípe, ako bola funkciu `polkruznica` z predchádzajúcej úlohy, napíš funkciu `polkruh(velkost, smer)`. Korytnačka teraz vnútro nakresleného oblúka vyfarbí náhodnou farbou. Otestuj podobne ako predchádzajúcu úlohu, ale po nakreslení 10 farebných polkruhov dokresli ďalších 10 tak, aby sa vlnky doplnili do farebných kruhov - zrejme každý z takýchto kruhov sa bude skladať z dvoch rôzne zafarbených polovíc. Mal by si dostať:

11. Napíš funkciu `bodky(n, m)`, ktorá nakreslí `n` radov bodiek (`t.dot(...)`) po `m` v každom rade. Stredy bodiek v radoch aj stĺpcoch sú vo vzdialenosti `30`. Všetky bodky majú farbu `'salmon'` a náhodnú veľkosť od `20` do `35`. Nepoužívaj metódu `setpos`. Otestuj, napríklad `bodky(10, 15)`



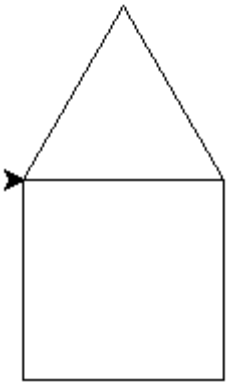
12. Napíš funkciu `stvorce(dlzka, krok)`, ktorá nakreslí niekoľko farebných štvorcov. Všetky budú zafarbené náhodnými farbami a budú bez obrysu (kreslia sa so zvihnutym perom). Najväčší z nich je prvý a má veľkosť strany `dlzka`. Každý ďalší má rovnaký stred (ako predchádzajúci), ale stranu štvorca má o `krok` menšiu. Ak by mal takýto štvorec nulovú alebo zápornú stranu, kreslenie skončí. Nepoužívaj metódu `setpos`. Napríklad volanie `stvorce(200, 25)` nakreslí:



Asi sa ti oplatí vytvoriť pomocnú funkciu `stvorec(dlzka)`, ktorá nakreslí náhodne zafarbený štvorec so stranou `dlzka`.

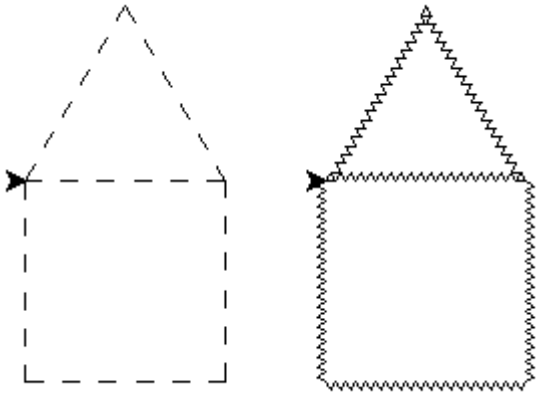
13. Táto úloha je podobná predchádzajúcej, ale funkcia `veza(dlzka, krok)` bude kresliť náhodne zafarbené štvorce nad seba. Každý menší je vycentrovaný na predchádzajúci. Nepoužívaj `setpos`. Napríklad volanie `veza(120, 30)` nakreslí:

14. Napíš funkciu `dom(d)`, ktorá nakreslí domček zo štvorca a rovnostranného trojuholníka tak, že po každej čiare prejde len raz. Pozícia korytnačky na obrázku je pri štarte. Po skončení kreslenia domčeka bude asi inde. Pre volanie `dom(100)` by si mal dostať:



Teraz napíš ďalšie dve funkcie `prerusovana_ciara(d)` a `cikcakova_ciara(d)`, pomocou ktorých nakreslíme buď prerušovanú čiaru alebo cikcakovú čiaru. Prerušovaná čiara označuje rozdelenie úsečky dĺžky `d` na `11` rovnakoveľkých častí, pričom každá druhá sa prejde so zdvihnutým perom. Cikcaková čiara označuje, že úsečka dĺžky `d` sa rozdelí na úseky dĺžky `5` a každý úsek sa nakreslí pod uhlom `60` ako dve strany rovnostranného trojuholníka so stranou `5` (predpokladáme, že `d` je deliteľné číslom `5`).

Ak by sme teraz vo funkcii `dom` nahradili volanie metódy `t.fd(d)` buď volaním `prerusovana_ciara(d)` alebo `cikcakova_ciara(d)`, dostaneme domček z prerušovaných alebo cikcakových čiar:

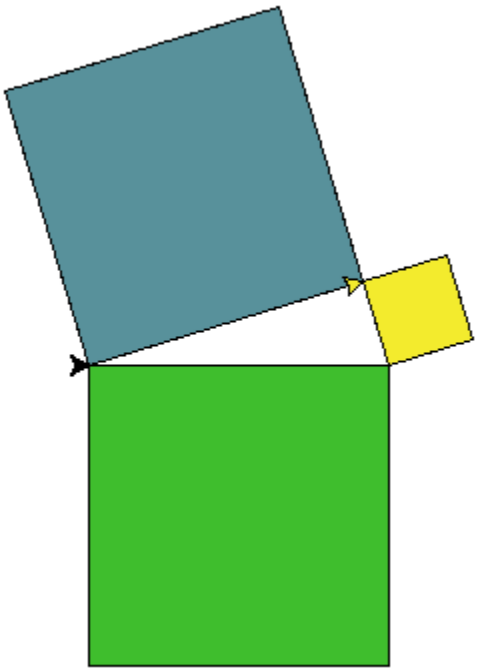


Nepoužívaj metódu `setpos`.

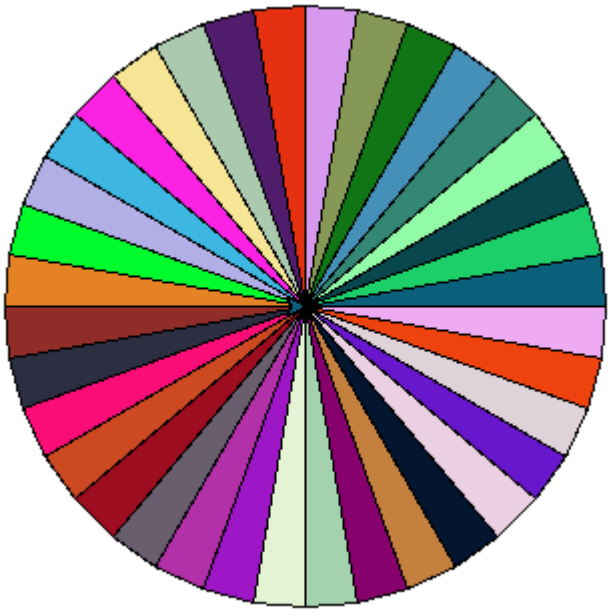
15. (trochu matematiky) Budeš kresliť obrázok, ktorý demonštruje Pytagorovu vetu: súčet štvorcov nad odvesnami sa rovná štvorcu nad preponou. Napíš funkciu `pytagoras(prepona, uhol)`, ktorá nakresli štvorec nad preponou, vypočíta dĺžky oboch odvesien a nakreslí oba štvorce. `uhol` je vnútorný uhol pravouhlého trojuholníka. Uvedom si, že dĺžky odvesien môžeš počítať ako `prepona * cos(uhol)` a `prepona * sin(uhol)`, daj pozor na radiány.

    Funkcia potom vypíše (do shellu) obsah štvorca nad preponou a obsahy oboch štvorcov nad odvesnami. Pri kreslení štvorcov môžeš použiť funkciu na kreslenie náhodne vyfarbeného štvorca (z úlohy (5)). Pre volanie `pytagoras(150, 17)` by si mohol dostať takýto výstup:

```
stvorec nad preponou = 22500
stvorec nad 1. odvesnou = 20576.672691244217
stvorec nad 2. odvesnou = 1923.3273087557814
sucet = 22500.0
```



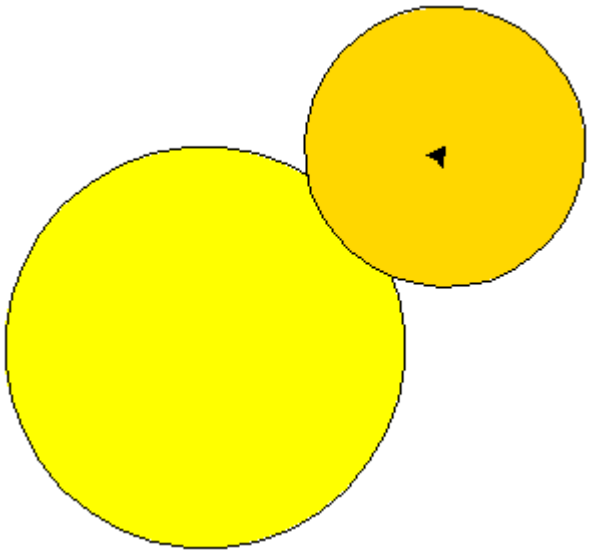
16. (trochu matematiky) Napíš funkciu `troj(rameno, uhol)`, ktorá nakreslí rovnoramenný trojuholník s danou dĺžkou ramena a uhlom, ktorý zvierajú tieto dve ramená. V tomto vrchole začne aj skonči kresliť. Trojuholník vyplň náhodnou farbou.
    - otestuj nakreslením 36 takýchto trojuholníkov s ramenom 300 a s uhlom 10, po každom trojuholníku sa otoč o 10 stupňov



17. Napíš funkciu `kruznica(r)`, ktorá nakreslí kružnicu s polomerom `r` a so stredom, ktorý je v momentálnej pozícii korytnačky. Kružnicu kresli ako pravidelný 36-uholník. Uvedom si, že ak by strana tohto 36-uholníka bola `d`, tak obvod vypočítame ako `2*pi*r = 36*d`. Z tohto vzťahu vieš vypočítať `d` a teda nakresliť pravidelný 36-uholník. Po skončení kreslenia, korytnačka bude v rovnakej pozícii ako začala. Nepoužívaj metódu `setpos`. Vyskúšaj:

```
t.dot(200, 'yellow')
kruznica(100)
t.pu()
t.fd(120)
t.lt(90)
t.fd(100)
t.rt(37)
t.pd()
t.dot(140, 'gold')
kruznica(70)
```

dostaneš (dva žlté kruhy kreslené pomocou `dot` sú tu len na kontrolu):

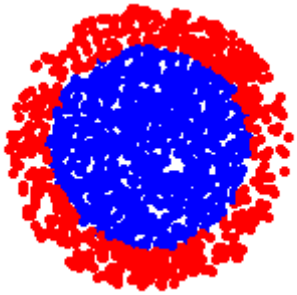


18. Napíš funkciu `prechadzka(n)`, ktorá zrealizuje `n` krokov náhodnej prechádzky:
    - korytnačka chodí náhodne so zdvihnutým perom s krokom `20`
    - ak sa vzdiali od `(0, 0)` o viac ako `70`, cúvne `20`
    - inak, ak sa vzdiali od `(0, 0)` o viac ako `50`, na momentálnej pozícii nakreslí červenú bodku (veľkosti `5`)
    - inak na momentálnej pozícii nakreslí modrú bodku (veľkosti `5`)

    Vyskúšaj:

```
turtle.tracer(0)
prechadzka(2000)
turtle.tracer(1)
```

mal by si dostať niečo takéto:



19. Napíš dve funkcie `vyrob(n, poz)` a `smerom(zoznam, poz)`:
    - prvá funkcia `vyrob(n, poz)` vytvorí na náhodných pozíciách `n` korytnačiek, každej nastaví náhodnú farbu pera a hrúbku pera `5`; každú korytnačku ešte natočí smerom k bodu `poz` (parameter `poz` je dvojica nejakých súradníc `(x, y)`); všetky tieto korytnačky vloží do zoznamu a tento zoznam vráti (`return`) ako výsledok funkcie; zrejme využiješ korytnačiu metódu `towards`
    - druhá funkcia `smerom(zoznam, poz)` dostáva zoznam korytnačiek a nechá postupne všetky korytnačky z tohto zoznamu presúvať k bodu `poz` tak, že 50-krát robí toto:
        - každá korytnačka si vypočíta vzdialenosť k bodu `poz` a prejde (`forward`) desatinu tejto vzdialenosti

Otestuj, napríklad:

```
pp = (0, -300)
zoz = vyrob(100, pp)
smerom(zoz, pp)
```