**Python**

**Python** je moderný programovací jazyk, ktorého popularita stále rastie.

* jeho autorom je [Guido van Rossum](http://sk.wikipedia.org/wiki/Guido_van_Rossum) (vymyslel ho v roku 1989)
* používajú ho napr. v Google, YouTube, Dropbox, Mozilla, Facebook,...
* beží na rôznych platformách, napr. Windows, Linux, Mac. Je to freeware a tiež open source.

Na rozdiel od mnohých iných jazykov, ktoré sú kompilačné (napr. Pascal, C/C++) je Python interpreter. To znamená, že

* interpreter nevytvára spustiteľný kód (napr. .exe súbor vo Windows)
* na spustenie programu musí byť v počítači nainštalovaný Python

Ako ho získať

* zo stránky <https://www.python.org/> stiahnete najnovšiu verziu Pythonu - momentálne je to verzia **3.5.1** (pre 32 bitový operačný systém Windows – **Windows x86 executable installer**, pre 64 bitový operačný systém Windows – **Windows x86-64 executable installer**)
* spustite inštalačný program (zaklikneme Customize installation a tiež poslednú možnosť Add Python 3.5 PATH, v okne Advanced Options potvrdíme možnosť Install for all users, spúšťame ho ikonou pre IDLE(Python 3.5...))

Po spustení **IDLE (Python GUI)** - čo je vývojové prostredie (Integrated Development Environment), vidíme informáciu o verzii Pythonu a tiež riadok s tromi znakmi >>> (tzv. výzva, t.j. **prompt**). Za túto výzvu budeme písať príkazy pre Python.

Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>>

Ako to funguje

* Python je interpreter a pracuje v niekoľkých možných režimoch
* teraz sme ho spustili v **príkazovom režime**: očakáva zadávanie textových príkazov (do riadka za znaky >>>), každý zadaný príkaz vyhodnotí a vypíše prípadnú reakciu alebo **chybovú správu**, ak sme zadali niečo nesprávne
* po skončení vyhodnocovania riadka sa do ďalšieho riadka znovu vypíšu znaky >>> a očakáva sa opätovné zadávanie ďalšieho príkazu
* takémuto interaktívnemu oknu hovoríme **shell**

Môžeme teda zadávať rôzne matematické výrazy

>>> 123+15

138

>>> 2\*3

6

>>> 1\*2\*3\*4\*5\*6\*7\*8\*9

362880

>>>

* ak okno zatvoríme nič sa v ňom neuchovalo, preto na programovanie budeme používať iný režim

**Programovací režim**

Spustíme IDLE a vytvoríme nové okno (menu **File** a položka **New File**), resp. stlačíme <Ctrl-N>. Otvorí sa nové textové okno, ale už bez promptu >>>. Do tohto okna budeme zadávať príkazy.

**Grafika**

Vytvorenie grafickej plochy

Keď chceme pracovať s grafickou plochou, najprv musíme do programu naimportovať modul na prácu s grafickou plochou (je to nejaký program, v ktorom už niekto naprogramoval príkazy, ktoré budeme používať). My budeme na prácu s grafikou používať modul tkinter. Tento modul naimportujeme na začiatku príkazom import tkinter.

Po naimportovaní modulu vytvoríme grafickú plochu (plátno), do ktorej budeme neskôr kresliť   
canvas = tkinter.Canvas(). Posledný príkaz canvas.pack() zabezpečí zobrazenie nového okna aj s vytvorenou grafickou plochou. Celý program zatiaľ vyzerá takto:

# môj program

import tkiner

canvas = tkinter.Canvas()

canvas.pack()

* takýmto programom hovoríme **skript** - väčšinou sú uložené v súbore s príponou .py
* spúšťame ich **Run**/**Run Module** (alebo F5)
* po ich spustení sa v pôvodnom okne **shell** najprv celý Python reštartuje (zabudne, všetko, čo sme ho doteraz naučili) a takto vyčistený vykoná všetky príkazy programu
* do prvého riadka sme zapísali komentár - text za znakom # sa ignoruje, teda je to komentár pre nás (znak # stačí jeden, ak ho zadávame my, ale môžeme ho pridať aj Format/Comment Out Region – vtedy Python pridáva dva ##)

Ďalšie poznámky pre učiteľa: príkaz import tkiner oznámi, že budeme pracovať s **modulom**, ktorý obsahuje grafické príkazy

* týmto príkazom vznikla nová premenná tkinter, ktorá obsahuje referenciu na tento modul, t.j. všetky funkcie a premenné, ktoré sú definované v tomto module, sú prístupné pomocou tejto premennej a preto k nim budeme pristupovať tzv. bodkovou notáciou , t.j. vždy uvedieme meno premennej tkinter, za tým bodku a meno funkcie alebo premennej, napr. tkinter.Canvas
* zápis tkinter.Canvas(),vytvorí grafickú plochu a aby sme s touto plochou mohli pracovať, uložíme si jej referenciu do premennej canvas (canvas = tkinter.Canvas())
* ak priraďujeme do premennej dávame medzery pred a za =
* ak chceme nastaviť veľkosť grafickej plochy tak v príkaze canvas = tkinter.Canvas() do zátvoriek uvedieme parametre bg – background (pozadie – napíšeme anglický názov farby v apostrofoch), width (zadáme číslo – šírka grafickej plochy), height (výška grafickej plochy) - pričom na poradí parametrov nezáleží: canvas = tkinter.Canvas(bg=ˈwhiteˈ, width=1000, height=800) – v tomto prípade medzery pred a za = nedávame, lebo priraďujeme do pomenovaného parametra
* nezabudnime uviesť okrúhle zátvorky: vďaka nim sa zavolá funkcia, ktorá vytvorí nové okno, bez týchto zátvoriek zápis tkinter.Canavas označuje len referenciu na nejakú funkciu bez jej zavolania
* kým nezadáme aj príkaz canvas.pack(), grafická plocha sa ešte nezobrazí – volanie canvas.pack() zabezpečí zobrazenie nového okna aj s novovytvorenou grafickou plochou

**Parametre grafickej plochy**

Pri vytváraní grafickej plochy (pomocou tkinter.Canvas()) môžeme nastaviť veľkosť plochy ale aj farbu pozadia grafickej plochy. Môžeme uviesť tieto parametre:

* bg= nastavuje farbu pozadia (z anglického “background”)
* width= nastavuje šírku grafickej plochy
* height= výšku plochy

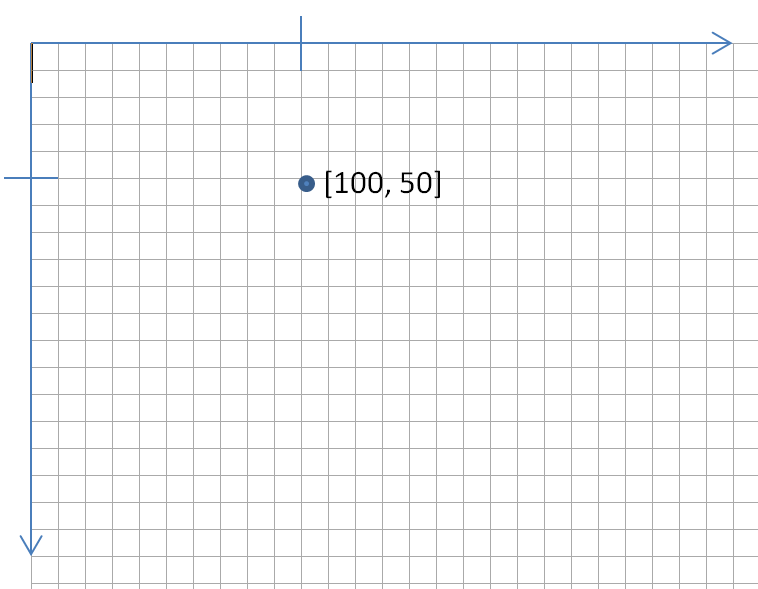
Všetky grafické príkazy pracujú s grafickou plochou, ku ktorej máme prístup prostredníctvom premennej canvas. Tieto príkazy sú v skutočnosti funkciami, ktoré budeme volať s nejakými parametrami. Všeobecný tvar väčšiny grafických príkazov je

canvas.create\_utvar(x1, y1, x2, y2, param=hodnota, param=hodnota) kde

* create\_utvar je meno funkcie na vytvorenie grafického objektu, napr. create\_line, create\_rectangle, create\_oval, ...
* x1, y1, x2, y2, ... je postupnosť dvojíc súradníc bodov v grafickej ploche (rôzne príkazy majú rôzny počet bodov v rovine)
* param=hodnota je dvojica: meno doplnkového parametra (napr. fill, width, ...) a jej hodnota (napr. 'red', 5, 'Arial 20', ...)

Pre všetky grafické príkazy platí, že keď pre doplnkové parametre nejaké hodnoty neuvedieme, tak tieto majú nastavené svoje inicializované (default) hodnoty (často takýmito náhradnými default hodnotami sú napr.fill='black' alebo width=1).

**Súradnicová sústava**

* **x-ová os** prechádza po hornej hrane grafickej plochy zľava doprava
* **y-ová os** prechádza po ľavej hrane grafickej plochy zhora nadol
* začiatok [0, 0] je v ľavom hornom rohu plochy
* môžeme používať aj záporné súradnice, vtedy označujeme bod, ktorý je mimo plochu
* ak neurčíme veľkosť grafickej plochy – tak je 378x264 bodov (pixelov)

**Otázky:**

Na obrázku vidíme zakreslený bod so súradnicami [100, 50].

Aké súradnice bude mať bod, ktorý je o 20 bodov vľavo od [100, 50]?

Aké súradnice bude mať bod, ktorý je o 20 bodov vpravo od [100, 50]?

Aké súradnice bude mať bod, ktorý je o 20 bodov hore od [100, 50]?

Aké súradnice bude mať bod, ktorý je o 20 bodov dole od [100, 50]?

Čo majú spoločné všetky body, ktoré sú vľavo alebo vpravo od [100, 50]?

Čo majú spoločné všetky body, ktoré sú hore alebo dole od [100, 50]?

Pri práci s grafikou budeme často experimentovať v priamom režime a keď budeme s výsledkom spokojní, prekopírujeme príkazy zo shellu do programu v editovacom okne a prípadne znovu spustíme pomocou F5.

**Kreslenie čiar**

Čiary v grafickom režime kreslíme pomocou funkcie canvas.**create\_line()** Táto funkcia dokáže kresliť lomenú čiaru, ktorá sa skladá z jednej alebo viacerých úsečiek.

* parametrom funkcie je postupnosť súradníc v tvare canvas.create\_line(x1, y1, x2, y2, ...)
* ak je zadaných bodov (dvojíc x, y) viac ako 2, kreslí sa lomená čiara zložená z navzájom nadväzujúcich úsečiek

Pri kreslení čiar môžeme ešte špecifikovať aj ďalšie parametre tejto kresby: za postupnosť bodov môžeme uviesť hrúbku nakreslenej čiary (parameter width=) a jej farbu (parameter fill=). Zrejme, keď tieto dva parametre nešpecifikujeme, kreslí sa čierna čiara hrúbky 1.

**Kreslenie čiar – funkcia** canvas.**create\_line()**

canvas.**create\_line**(x1, y1, x2, y2, ...) alebo

canvas.**create\_line**(x1, y1, x2, y2, ..., width=číslo, fill=farba)

Parametre:

* **x, y** – dvojica súradníc jedného bodu lomenej čiary
* **width=číslo** – nastavenie hrúbky čiary, ak tento parameter chýba, predpokladá sa hrúbka 1
* **fill=farba** – nastavenie farby čiary, ak tento parameter chýba, predpokladá sa black
* zoznam farieb nájdeme *C:Program Files/Python35/Tools/pynche/X/rgb*

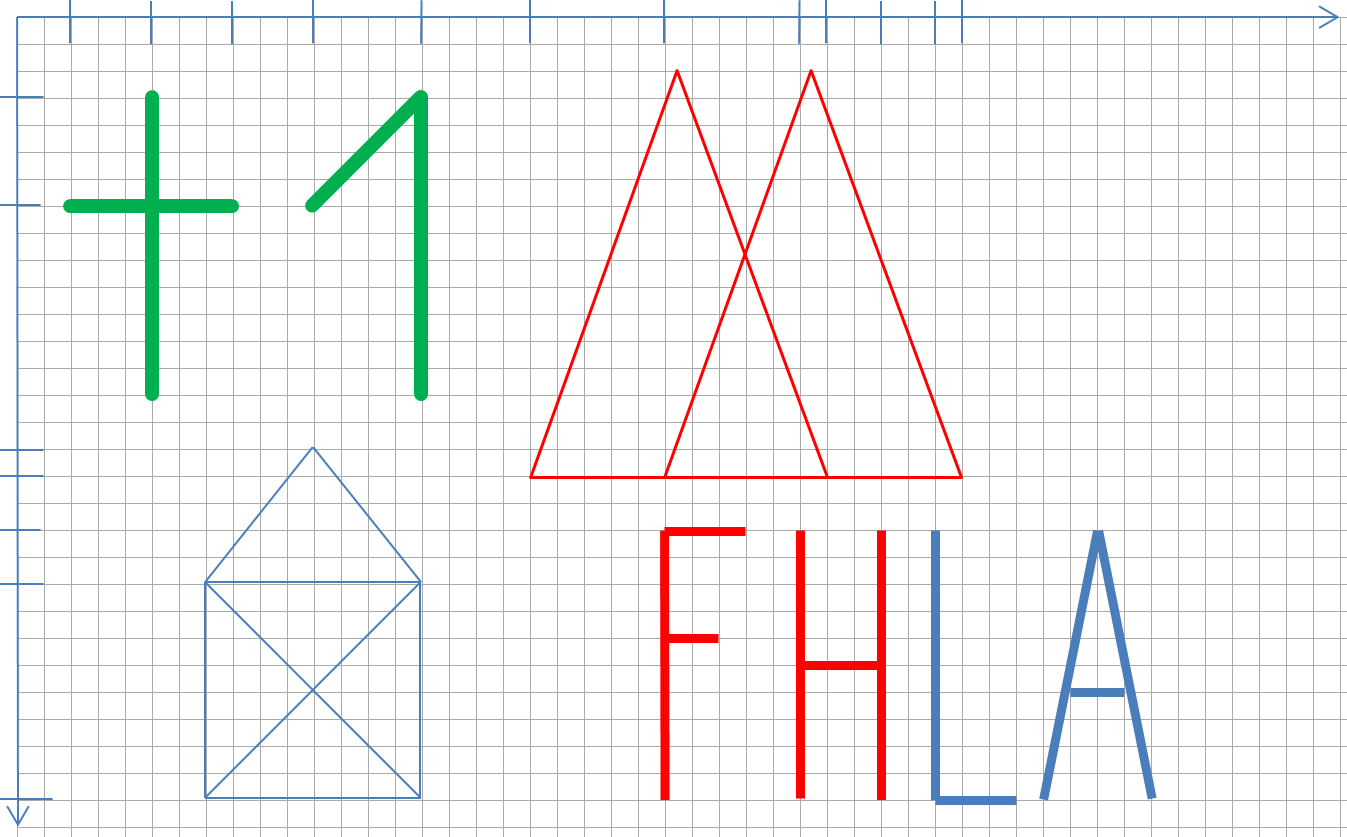
Funkcia canvas.create\_line() kreslí lomenú čiaru, ktorá sa môže skladať aj z niekoľkých navzájom napojených úsečiek. Postupnosť dvojíc súradníc by mala obsahovať aspoň dva body (teda aspoň jednu úsečku). Číselné hodnoty pre súradnice môžu byť aj desatinné čísla (napr. 10.5)

**Úlohy:**

Nakresli vodorovnú čiaru s dĺžkou 100. Nastav jej hrúbku na 5 bodov a farbu na červenú.

Nakreslite obdĺžnik so stranami 100, 50, pričom strany obdĺžnika sú rovnobežné so súradnicovými osami.

Pomocou čiar nakreslite obrázky. (ak nechceme, aby sa nás Python stále pýtal či to chceme uložiť tak nastavíme automatické ukladanie bez otázky: *Options/Configure IDLE/General/No prompt* – treba to začiarknuť)



import tkinter

canvas = tkinter.Canvas(bg='white', width=1000, height=800)

canvas.pack()

#krizik

canvas.create\_line(50, 30, 50, 140, width=5, fill='green')

canvas.create\_line(20, 70, 80, 70, width=5, fill='green')

#jednotka

canvas.create\_line(110, 70, 150, 30, 150, 140, width=5, fill='green')

#trojuholnik

canvas.create\_line(190, 170, 245, 30, 315, 170, 190, 170, width=3, fill='red')

#posunuty trojuholnik

canvas.create\_line(240, 170, 295, 30, 365, 170, 240, 170, width=3, fill='red')

#domcek jednym tahom

canvas.create\_line(70, 290, 70, 210, 150, 210, 110, 160, 70, 210, 150, 290, 70, 290, 150, 210, 150, 290,

width=2)

#pismeno F

canvas.create\_line(240, 190, 240, 290, fill='red', width=3)

canvas.create\_line(240, 190, 270, 190, fill='red', width=3)

canvas.create\_line(240, 230, 260, 230, fill='red', width=3)

#pismeno H

canvas.create\_line(290, 190, 290, 290, fill='red', width=3)

canvas.create\_line(320, 190, 320, 290, fill='red', width=3)

canvas.create\_line(290, 240, 320, 240, fill='red', width=3)

#pismeno L

canvas.create\_line(340, 190, 340, 290, fill='blue', width=3)

canvas.create\_line(340, 290, 370, 290, fill='blue', width=3)

#pismeno A

canvas.create\_line(380, 290, 400, 190, 420, 290, fill='blue', width=3)

canvas.create\_line(390, 250, 411, 250, fill='blue', width=3)

#**toto je len pre seminaristov**

# utvaru priradime identifikator

id1 = canvas.create\_line(380, 290, 400, 190, 420, 290, fill='blue', width=3

#

canvas.move(id1, 10, 10)

#posunutemu A zmeni farbu

canvas.itemconfig(id1, fill='green')

#nakoniec to posunute vymazeme

canvas.delete(id1)

**Kreslenie polygónov**

Útvary nakreslené pomocou príkazu canvas.create\_line() nevieme vypĺňať farbou, ak chceme nejaké n-uholníky vyfarbiť musíme použiť iný príkaz a to canvas.create\_polygon().

Polygónom rozumieme uzavretú krivku, ktorá môže byť vyplnená nejakou farbou. Súradnicami sa tento príkaz podobá na canvas.create\_line() a ďalšie doplnkové parametre   
(width=, fill=, outline=) má rovnaké ako napr. canvas.create\_rectangle()

* parametrom je postupnosť súradníc v tvare   
  canvas.create\_polygon(x1, y1, x2, y2, x3, y3,...)
* aby mal polygón zmysel, postupnosť súradníc musí obsahovať aspoň 3 body

**Kreslenie polygónu – funkcia** canvas.**create\_polygon()**

canvas.**create\_polygon**(x1, y1, x2, y2, x3, y3, ...) alebo

canvas.**create\_polygon**(x1, y1, x2, y2, x3, y3..., width=číslo, fill=farba, outline=farba)

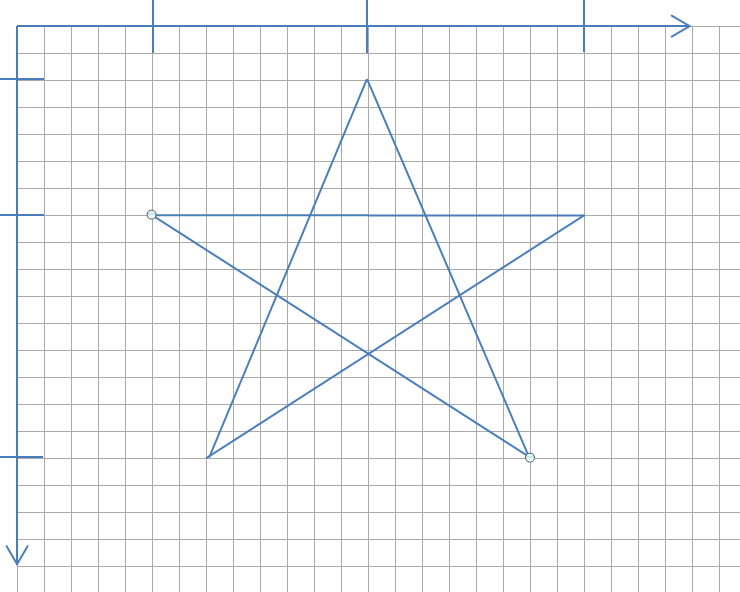
Parametre:

* **x, y** – dvojica súradníc jedného bodu lomenej čiary
* **width=číslo** – nastavenie hrúbky čiary, ak tento parameter chýba, predpokladá sa hrúbka 1
* **fill=farba** – nastavenie farby výplne, ak tento parameter chýba, predpokladá sa black, prázdny reťazec '' označuje n-uholník bez výplne
* **outline=farba** – nastavenie farby čiary obrysu, ak tento parameter chýba, predpokladá sa priesvitná farba, ''
* Funkcia canvas.create\_polygon() na základe postupnosti bodov nakreslí n-uholník

**Úloha:**

Nakreslite obdĺžnik so stranami dĺžky 100 a 50, pričom strany obdĺžnika sú rovnobežné so súradnicovými osami. Nastavte mu farbu výplne na zelenú, farbu obrysu na modrú a hrúbku čiary obrysu na 2.

Nakreslite polygón – pozri obr., nastavte mu výplň a farbu obrysu.

****

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=400)

canvas.pack()

##kreslenie polygonu

canvas.create\_polygon(70, 160, 130, 20, 180, 160, 50, 70, 210, 70, fill='blue',

outline='')

#pre seminaristov posuny utvarov

id1=canvas.create\_polygon(70, 160, 130, 20, 180, 160, 50, 70, 210, 70, fill='green',

outline='')

canvas.move(id1, 50, 50)

canvas.move(id1, 100, 150)

Poznámka: ak chceme vyvolať kontextové menu CTRL+SPACE(medzerovník)

**Kreslenie obdĺžnika**

Obdĺžnik vieme nakresliť pomocou príkazu na kreslenie čiar (najmenej koľko bodov (súradníc týchto bodov) musíme v príkaze na kreslenie čiar použiť?) a pomocou príkazu na kreslenie polygónov čiar (najmenej koľko bodov (súradníc týchto bodov) musíme v príkaze na kreslenie polygónov použiť?).

Vieme jednoznačne určiť obdĺžnik aj pomocou menšieho počtu bodov? Najmenej koľko ich potrebujeme?

Obdĺžniky kreslíme pomocou funkcie canvas.create\_rectangle(). V tejto funkcii sú parametrami súradnice dvoch protiľahlých vrcholov kresleného obdĺžnika. Jej tvar:

canvas.create\_rectangle(x1, y1, x2, y2). Strany tohto obdĺžnika sú rovnobežné so súradnicovými osami. Takto kreslené obdĺžniky (resp. štvorce) majú nevyplnené vnútro. Tiež obrys je tenká čierna čiara hrúbky 1. Pri kreslení obdĺžnikov môžeme ešte špecifikovať aj ďalšie parametre tejto kresby: za postupnosť bodov môžeme uviesť ďalšie doplnkové parametre: hrúbku obrysu (parameter width=), farbu obrysu (parameter outline=) a výplň obdĺžnika (parameter fill=)

**Kreslenie obdĺžnika – funkcia** canvas.**create\_rectangle()**

canvas.**create\_rectangle**(x1, y1, x2, y2) alebo

canvas.**create\_rectangle**(x1, y1, x2, y2, width=číslo, fill=farba, outline=farba)

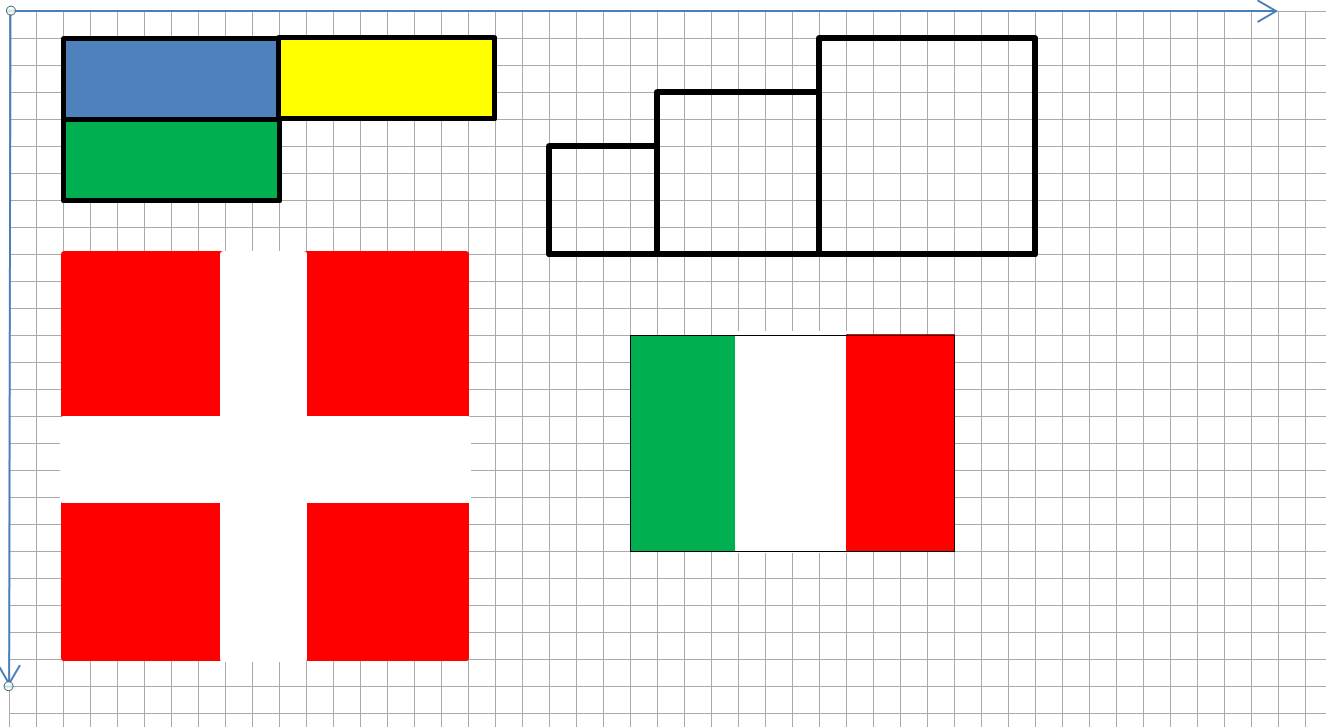
Parametre:

* **x1, y1** – dvojica súradníc jedného vrcholu obdĺžnika
* **width=číslo** – nastavenie hrúbky čiary, ak tento parameter chýba, predpokladá sa hrúbka 1
* **fill=farba** – nastavenie farby výplne, ak tento parameter chýba, predpokladá sa priesvitná farba, t.j. ''
* **outline=farba** – nastavenie farby čiary obrysu, ak tento parameter chýba, predpokladá sa farba black, prázdny reťazec '' označuje obdĺžnik bez obrysu
* Funkcia canvas.create\_rectangle() na základe dvoch bodov nakreslí obdĺžnik, ktorého strany sú rovnobežné so súradnicovými osami. Číselné hodnoty pre súradnice môžu byť aj desatinné čísla.

**Úlohy:**

Príkaz canvas.create\_rectangle(10, 50, 110, 100) nakreslí obdĺžnik, ktorý je zadaný pomocou bodov so súradnicami [10, 50], [110, 100]. Akú šírku a výšku v bodoch má tento obdĺžnik. Môžeme ho nakresliť aj týmto príkazom canvas.create\_rectangle(110, 100, 10, 50). Môžeme tento istý obdĺžnik nakresliť ešte pomocou iných súradníc? Čo sa stane po zadaní tohto príkazu canvas.create\_rectangle(10, 50, 110, 50)? Čo sa stane po zadaní tohto príkazu canvas.create\_rectangle(10, 50, 10, 50) ? Akú šírku a výšku majú predchádzajúce obdĺžniky?

Nakreslite obdĺžniky podľa predlohy:



import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

#1. uloha

canvas.create\_rectangle(20, 10, 100, 40, fill='blue', width=2)

canvas.create\_rectangle(100, 10, 180, 40, fill='yellow', width=2)

canvas.create\_rectangle(20, 40, 100, 70, fill='green', width=2)

#2. uloha

canvas.create\_rectangle(200, 50, 240, 90, width=2)

canvas.create\_rectangle(240, 30, 300, 90, width=2)

canvas.create\_rectangle(300, 10, 380, 90, width=2)

#3. uloha

canvas.create\_rectangle(20, 90, 170, 240, fill='red', outline='')

canvas.create\_rectangle(80, 90, 110, 240, fill='white', outline='')

canvas.create\_rectangle(20, 150, 170, 180, fill='white', outline='')

#4. uloha

canvas.create\_rectangle(230, 120, 270, 240, fill='green')

canvas.create\_rectangle(270, 120, 310, 240, fill='white')

canvas.create\_rectangle(310, 120, 350, 240, fill='red')

**Otázky:**

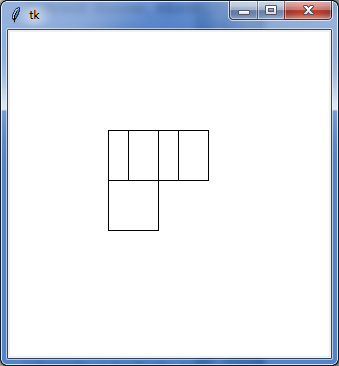
Čo sa stane, keď parametre na nastavenie farby pomiešame medzi súradnice bodov canvas.create\_rectangle(fill='white',120, 100, 170, 150, outline='yellow')

Čo sa stane keď parameter outline nastavíme takto otline=ˈˈ?

Čo sa stane keď parameter fill nastavíme takto fill=ˈˈ?

Čo sa stane keď parameter outline a súčasne parameter fill nastavíme takto otline=ˈˈ, fill=ˈˈ?

Koľko obdĺžnikov budeme vidieť po spustení týchto príkazov? (6)

****import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

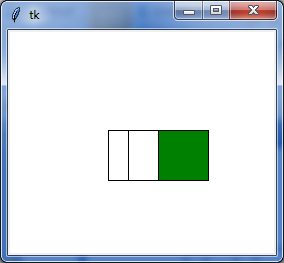
canvas.pack()

canvas.create\_rectangle(120, 100, 170, 150)

canvas.create\_rectangle(100, 100, 150, 150)

canvas.create\_rectangle(150, 100, 200, 150)

canvas.create\_rectangle(100, 150, 150, 200)

Koľko obdĺžnikov budeme vidieť po spustení týchto príkazov?

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

#Kolko obdlznikov uvidime

canvas.create\_rectangle(120, 100, 170, 150, fill='white')

canvas.create\_rectangle(100, 100, 150, 150, fill='')

canvas.create\_rectangle(150, 100, 200, 150, fill='green')

canvas.create\_rectangle(100, 150, 150, 200, fill='', outline='')

Adam chcel nakresliť vlajku Francúzska a napísal tento program. Nájdite chybu v tomto programe.

import tkinter

import random

canvas = tkinter.Canvas(bg='white', width=1000, height=800)

canvas.pack()

canvas.create\_rectangle(10, 50, 110, 300, fill='blue', outline='')

canvas.create\_rectangle(60, 50, 160, 300, fill='white', outline='')

canvas.create\_rectangle(110, 50, 210, 310, fill='red', outline='')

Možná oprava programu

#Adamov program - opraveny

canvas.create\_rectangle(10, 50, 110, 300, fill='blue', outline='')

canvas.create\_rectangle(**110**, 50, **210**, 300, fill='white', outline='')

canvas.create\_rectangle(**210**, 50, **310**, 310, fill='red', outline='')

**Kreslenie elipsy**

Už vieme, že príkaz canvas.create\_rectangle(100, 50, 200, 100) nakreslí obdĺžnik, ktorého ľavý horný bod má súradnicu [100, 50] a ľavý dolný bod má súradnicu [200, 100]. Ak použijeme rovnaké súradnice, ale zmeníme canvas.create\_rectangle na canvas.create\_oval, nakreslí sa elipsa – vpísaná do obdĺžnika.

**Otázky:**

Obdĺžnik vieme nakresliť jedným príkazom. Potrebujeme špeciálny príkaz na kreslenie štvorca? Podľa čoho spoznáme, že súradnice v príkaze pre obdĺžnik nakreslia štvorec?

Môžeme pomocou príkazu na kreslenie elipsy kresliť kruhy? Potrebujeme špeciálny príkaz na kreslenie kruhov? Podľa čoho spoznáme, že súradnice v príkaze pre elipsu nakreslia kruh?

**Kreslenie elipsy– funkcia** canvas.**create\_oval()**

canvas.**create\_oval**(x1, y1, x2, y2) alebo

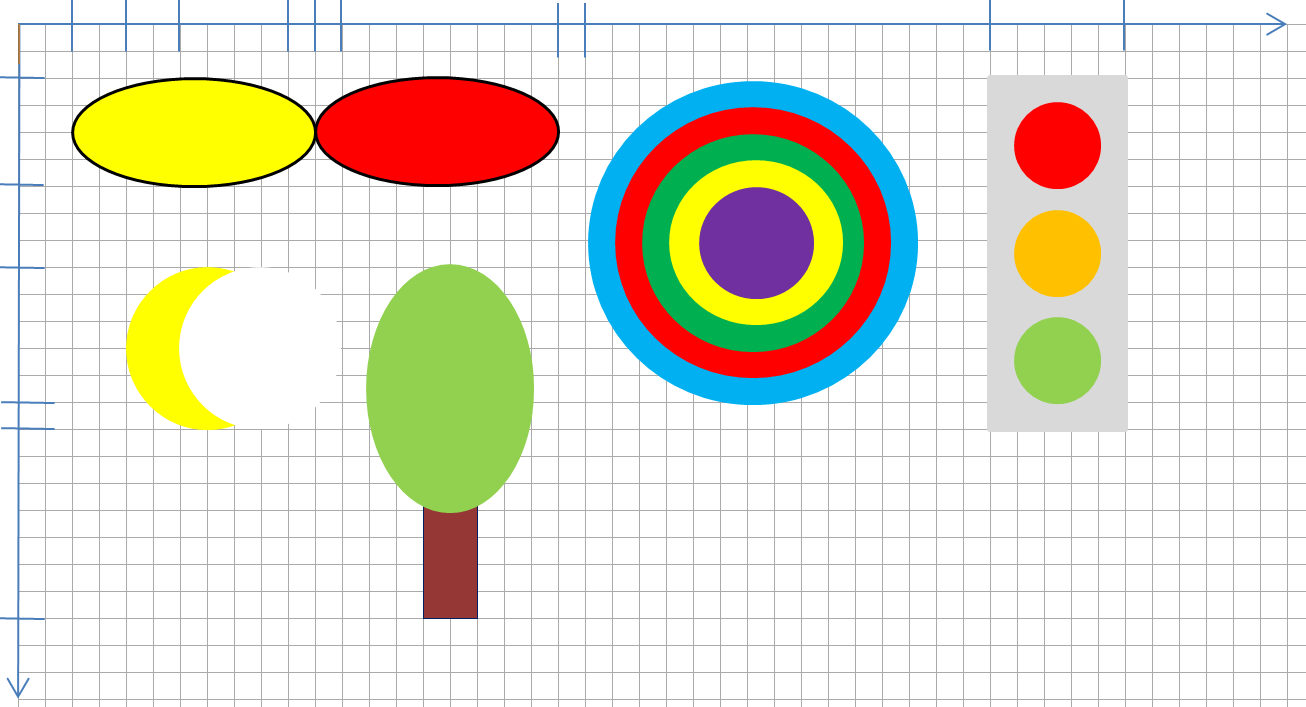
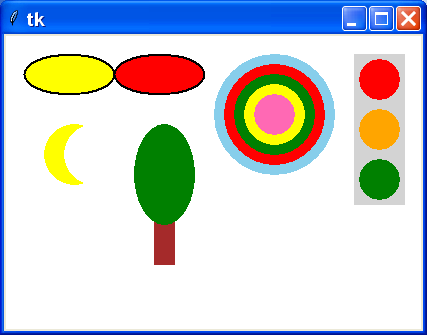
canvas.**create\_oval**(x1, y1, x2, y2, width=číslo, fill=farba, outline=farba)

Parametre:

* **x1, y1** – dvojica súradníc jedného vrcholu opísaného obdĺžnika elipsy
* **width=číslo** – nastavenie hrúbky čiary, ak tento parameter chýba, predpokladá sa hrúbka 1
* **fill=farba** – nastavenie farby výplne, ak tento parameter chýba, predpokladá sa priesvitná farba, t.j. ''
* **outline=farba** – nastavenie farby čiary obrysu, ak tento parameter chýba, predpokladá sa farba black, prázdny reťazec '' označuje elipsu bez obrysu
* Funkcia canvas.create\_oval() na základe dvoch bodov opísaného obdĺžnika nakreslí elipsu. Strany takéhoto mysleného obdĺžnika sú rovnobežné so súradnicovými osami

**Úlohy**

Nakreslite obrázky podľa predlohy

****

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

**#2elipsy**

canvas.create\_oval(20, 20, 110, 60, fill='yellow', width=2)

canvas.create\_oval(110, 20, 200, 60, fill='red', width=2)

**#sustredne kruhy**

canvas.create\_oval(270-60, 80-60, 270+60, 80+60, fill='Sky blue',

outline='Sky blue')

canvas.create\_oval(270-50, 80-50, 270+50, 80+50, fill='red',

outline='red')

canvas.create\_oval(270-40, 80-40, 270+40, 80+40, fill='green',

outline='green')

canvas.create\_oval(270-30, 80-30, 270+30, 80+30, fill='yellow',

outline='yellow')

canvas.create\_oval(270-20, 80-20, 270+20, 80+20, fill='hot pink',

outline='hot pink')

**#semafor**

canvas.create\_rectangle(350, 20, 400, 170, fill='light gray',

outline='light gray')

canvas.create\_oval(355, 25, 395, 65, fill='red', outline='red')

canvas.create\_oval(355, 75, 395, 115, fill='orange', outline='orange')

canvas.create\_oval(355, 125, 395, 165, fill='green', outline='green')

**#mesiac**

canvas.create\_oval(40, 90, 100, 150, fill='yellow', outline='yellow')

canvas.create\_oval(60, 90, 120, 150, fill='white', outline='white')

**#strom**

canvas.create\_rectangle(150, 180, 170, 230, fill='brown', outline='brown')

canvas.create\_oval(130, 90, 190, 190, fill='green', outline='green')

**Písanie textu**

Pomocou funkcie canvas.create\_text() môžeme do grafickej plochy písať aj texty. Prvé dva parametre funkcie sú súradnice stredu vypisovaného textu a ďalším doplnkovým parametrom je samotný text v tvare: canvas.create\_text(x, y, text='text') napr. canvas.create\_text(150, 120, text='Python', fill='blue', font='arial 30 bold')

**Písanie textu – funkcia** canvas.**create\_text()**

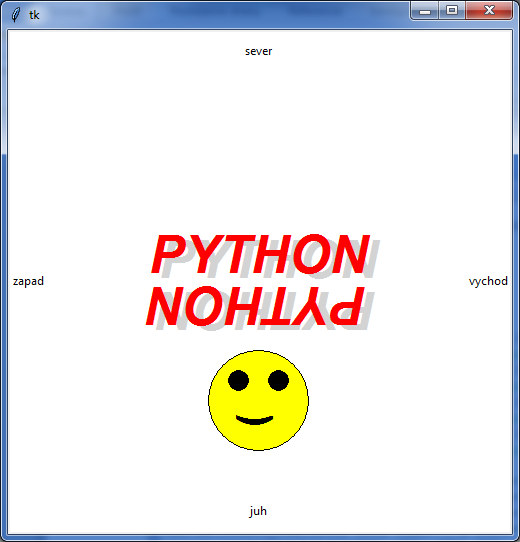
canvas.**create\_text**(x, y,text='text') alebo

canvas.**create\_rectangle**(x, y, text='text', font='písmo', fill=farba, angle=číslo)

Parametre:

* **x, y** – dvojica súradníc stredu vypisovaného textu
* **text=**'**text**' – zadaný text
* **fill=farba** – nastavenie farby textu
* **font=**'písmo' – nastavenie typu písma a aj jeho veľkosti – tento reťazec môže obsahovať aj informáciu o tvare písma **bold**, **italic**
* **angle=číslo** – otočenie výpisu o zadaný uhol v stupňoch
* Funkcia canvas.create\_text() na zadané súradnice bodu vypíše text

**Úlohy:**

Vypíšte nejaké slovo napr. PYTHON do stredu grafickej plochy veľkými písmenami červenej farby. Nastavte nejaké väčšie písmo. Otočte ho o 180°.

Presne do stredov strán grafickej plochy napíšte slová sever, východ, juh, západ.

Ak nejaký text napíšeme najprv svetlo šedou farbou a potom inou napr. červenou ale ho posunieme o niekoľko bodov, napr. smerom nadol a vľavo, získame dojem tieňa. Vyskúšajte tento efekt s nejakým väčším zaujímavým písmom.

Nakreslite smajlíka – využite príkazy čo sme sa doteraz naučili – príkaz na nakreslenie kruhu, na písanie a otáčanie textu.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

canvas.create\_text(250, 20, text='sever')

canvas.create\_text(250, 480, text='juh')

canvas.create\_text(20, 250, text='zapad')

canvas.create\_text(480, 250, text='vychod')

canvas.create\_text(260, 230, text='PYTHON', fill='light gray',

font='arial 40 bold italic')

canvas.create\_text(250, 225, text='PYTHON', fill='red',

font='arial 40 bold italic')

canvas.create\_text(260, 280, text='PYTHON', fill='light gray',

font='arial 40 bold italic', angle=180)

canvas.create\_text(250, 275, text='PYTHON', fill='red',

font='arial 40 bold italic', angle=180)

#smajlík

canvas.create\_oval(200, 320, 300, 420, fill='yellow')

canvas.create\_oval(220, 340, 240, 360, fill='black')

canvas.create\_oval(280, 340, 260, 360, fill='black')

canvas.create\_text(250, 390, text='D', font='arial 30 bold', angle=-90)

canvas.create\_rectangle(230, 370, 270, 385, fill='yellow', outline='yellow')

**Poznámka: Zmeny nakreslených útvarov**

Všetky útvary, ktoré kreslíme do grafickej plochy si systém pamätá tak, že ich dokáže dodatočne meniť (napr. farbu), posúvať po ploche, ale aj mazať. Všetky útvary sú v ploche vykresľované presne v tom poradí, ako sme zadávali jednotlivé grafické príkazy: skôr nakreslené útvary sú pod neskôr nakreslenými a môžu ich prekrývať.

Každý grafický príkaz (napr. canvas.create\_line()) je v skutočnosti funkciou, ktorá vracia celé číslo - identifikátor nakresleného útvaru (vidíme ho v okne shell). Toto číslo nám umožní neskoršie modifikovanie, resp. jeho zmazanie (Toto číslo nám umožní napr. v okne shell jeho mazanie príkazom canvas.delete(identifikátor), ak ako identifikátor použijeme reťazec ˈallˈ, príkaz zmaže všetky doteraz nakreslené útvary.)

**Zmazanie nakresleného objektu**

**funkcia canvas.delete()**

* slúži na zmazanie ľubovoľného nakresleného útvaru
* jeho tvar je canvas.delete(identifikátor)
  + kde identifikátor je návratová hodnota príkazu kreslenia útvaru
* ak ako identifikátor použijeme reťazec 'all', príkaz zmaže všetky doteraz nakreslené útvary

Napr.

id1 = canvas.create\_line(10, 20, 30, 40)

id2 = canvas.create\_oval(10, 20, 30, 40)

canvas.delete(id1)

zmaže prvý grafický objekt, t.j. úsečku, pričom druhý objekt kružnica ostáva bez zmeny.

**Posúvanie útvarov**

Pomocou identifikátora útvaru ho môžeme posúvať ľubovoľným smerom. Ostatné útvary sa pri tom nehýbu.

**funkcia canvas.move()**

* slúži na posúvanie ľubovoľného nakresleného útvaru
* jeho tvar je canvas.move(identifikátor, dx, dy)
  + kde identifikátor je návratová hodnota príkazu kreslenia útvaru
  + dx a dy označujú číselné hodnoty zmeny súradníc útvaru
* posúvaný útvar môže byť ľubovoľne komplikovaný (môže sa skladať aj z väčšieho počtu bodov), príkaz canvas.move() posunie všetky vrcholy útvaru
* ak ako identifikátor použijeme reťazec 'all', príkaz posunie všetky doteraz nakreslené útvary

Napr.

id1 = canvas.create\_line(10, 20, 30, 40)

id2 = canvas.create\_oval(10, 20, 30, 40)

canvas.move(id1, -5, 10)

posunie prvý nakreslený útvar, teda úsečku, druhý útvar (kružnicu) pri tom nehýbe.

**Zmena parametrov útvaru**

Nakresleným útvarom môžeme zmeniť ľubovoľné doplnkové parametre pomocou príkazu canvas.itemconfig().

**funkcia canvas.itemconfig()**

* slúži na zmenu ľubovoľných doplnkových parametrov nakresleného útvaru
* má tvar canvas.itemconfig(identifikátor, parametre)
  + kde identifikátor je návratová hodnota príkazu kreslenia útvaru
  + parametre sú ľubovoľné doplnkové parametre pre daný útvar

Napr.

**>>>** id1 = canvas.create\_line(10, 20, 30, 40)

**>>>** id2 = canvas.create\_oval(10, 20, 30, 40)

**>>>** canvas.itemconfig(id1, width=5, fill='blue')

**>>>** canvas.itemconfig(id2, outline='', fill='red')

**Zmena súradníc**

Okrem posúvania útvaru môžeme zmeniť aj jeho kompletnú postupnosť súradníc. Napr. pre canvas.create\_line() alebo canvas.create\_polygon() môžeme zmeniť aj počet bodov útvaru.

**funkcia canvas.coords()**

* slúži na zmenu súradníc nakresleného útvaru
* má tvar canvas.coords(identifikátor, postupnosť)
  + kde identifikátor je návratová hodnota príkazu kreslenia útvaru
  + postupnosť je ľubovoľná postupnosť súradníc, ktorá je vhodná pre daný útvar - táto postupnosť musí obsahovať párny počet čísel (celých alebo desatinných)

Napr.

**>>>** i1 = canvas.create\_line(10, 20, 30, 40)

**>>>** canvas.coords(i1, 30, 40, 50, 60, 70, 90)

**Premenné**

Po zadaní výrazov v príkazovom režime – okno shell (za promptom >>>) sa tieto vyhodnotili a hneď aj vypísali. Po zadaní v programovacom režime sa tiež vyhodnotia, ale ich hodnota sa nevypíše ale ignoruje. Ak chceme, aby sa táto hodnota neignorovala, musíme ju priradiť do premennej.

V Pythone konštanty alebo výrazy môžu byť rôznych typov. V Pythone má každý typ svoje meno:

int ako **celé čísla**, napr. 0, 1 15, -123456789, ...

float ako **desatinné čísla**, napr. 0.0, 3.14159, 2.00000001, 33e50, ...

str ako **znakové reťazce**, napr. 'a', 'abc'

Rôzne typy majú zadefinované rôzne operácie:

* **celočíselné operácie** (oba operandy musia byť celočíselného typu)
  + **+** súčet, napr. 1+2 má hodnotu 3
  + **–** rozdiel, napr. 2-5 má hodnotu -3
  + **\*** násobenie, napr. 3\*37 má hodnotu 111
  + **//** celočíselné delenie, napr. 22//7 má hodnotu 3
  + **%** zvyšok po delení, napr. 22%7 má hodnotu 1
  + **\*\*** umocňovanie, napr. 2\*\*8 má hodnotu 256
  + **nemôžeme deliť nulou**
* **operácie s desatinnými číslami** (aspoň jeden operand musí byť desatinného typu okrem delenia /)
  + **+** súčet, napr. 1+0.22 má hodnotu 1.2
  + **–** rozdiel, napr. 6-2.86 má hodnotu 3.14
  + **\*** násobenie, napr. 1.5\*2.5 má hodnotu 3.75
  + **/** delenie, napr. 23/3 má hodnotu 7.66666666666667
  + **//** delenie zaokrúhlené nadol, napr. 23.0//3 má hodnotu 7.0
  + **%** zvyšok po delení, napr. 23.0%3 má hodnotu 2.0
  + **\*\*** umocňovanie, napr. 3\*\*3. má hodnotu 27.0
  + **nemôžeme deliť nulou**
* **operácie so znakovými reťazcami**
  + **+** zreťazenie (spojenie dvoch reťazcov), napr. ˈaˈ+ˈbˈ má hodnotu ˈabˈ
  + **\*** viacnásobné zreťazenie toho istého reťazca, napr. 3\* ˈaˈ má hodnotu ˈaaaˈ (môžeme to zapísať aj ˈaˈ\*3)

**Premenné a priradenie**

* meno premennej:
  + môže obsahovať písmená, číslice a znak podčiarkovník
  + pozor na to, že v Pythone sa rozlišujú malé a veľké písmená
  + musí sa líšiť od Pythonovských príkazov (napr. for, if, return, ...)
* premenná sa vytvorí priraďovacím príkazom (ak ešte doteraz neexistovala):
  + zapisujeme: premenná = hodnota
    - tento zápis znamená, že do *premennej* sa má priradiť zadaná *hodnota*
  + v skutočnosti sa v Pythone do premennej priradí **referencia** (odkaz) na danú hodnotu (a nie samotná hodnota)
  + ďalšie priradenie do tej istej premennej zmení túto referenciu
  + na tú istú hodnotu sa môže odkazovať aj viac premenných
  + meno môže referencovať (mať priradenú**) maximálne jednu hodnotu**

Python si v svojej pamäti udržuje všetky premenné (v tzv. pamäti mien premenných) a všetky momentálne vytvorené hodnoty (v tzv. pamäti hodnôt).

V Pythone majú programátori dohodu, že na premenné sa používajú len malé písmená (prípadne aj číslice a podčiarkovník). Pri písaní programov používame čo najvhodnejšie mená premenných. Zápisu programu má rozumieť hlavne človek – preto mená premenných môžu byť napr.

polomer = 16

pi = 3.14

obvod = 2\*pi\*polomer

obsah = 2\*pi\*\*polomer

print('obvod= ', obvod)

print('obsah= ', obsah)

Vráťme sa k priraďovaciemu príkazu v tvare:

meno\_premennej = hodnota

* najprv sa zistí hodnota na pravej strane priraďovacieho príkazu => táto hodnota sa vloží do **pamäte hodnôt**
* ak sa toto meno\_premennej ešte nenachádzalo v **pamäti mien premenných**, tak sa vytvorí toto nové meno
* meno\_premennej dostane referenciu na novú vytvorenú hodnotu

Aktualizácia (update) premennej znamená, že ak v premennej je priradená nejaká hodnota a túto hodnotu zmeníme napr.:

ab = 12

ab = ab + 5

Python na aktualizácie nejakej premennej ponúka špeciálny zápis priraďovacieho príkazu:

meno\_premennej += hodnota # meno\_premennej = meno\_premennej + hodnota

meno\_premennej -= hodnota # meno\_premennej = meno\_premennej – hodnota

meno\_premennej /= hodnota # meno\_premennej = meno\_premennej / hodnota

meno\_premennej //= hodnota # meno\_premennej = meno\_premennej // hodnota

meno\_premennej %= hodnota # meno\_premennej = meno\_premennej % hodnota

meno\_premennej \*\*= hodnota # meno\_premennej = meno\_premennej \*\* hodnota

Každý z týchto zápisov je len skrátenou formou bežného priraďovacieho príkazu.

Ďalším užitočným tvarom priraďovacieho príkazu je možnosť naraz priradiť tej istej hodnoty do viacerých premenných, napr.

x = 0

sucet = 0

pocet = 0

ab = 0

môžeme to nahradiť jediným priradením:

x = sucet = pocet = ab =0

V takomto hromadnom priradení dostávajú všetky premenné tú istú hodnotu, teda referencujú (odkazujú) na tú istú hodnotu v pamäti hodnôt.

Posledným užitočným variantom priradenia je tzv. paralelné priradenie: naraz priraďujeme aj rôzne hodnoty do viacerých premenných, napr.

x = 120

sucet = 125

pi = 3.14

ab = 'bod A'

môžeme zapísať jediným paralelným priradením:

x, sucet, pi, ab = 120, 125, 3.14, 'bod A'

Samozrejme, že na oboch stranách priraďovacieho príkazu musí byť rovnaký počet mien premenných a počet hodnôt. Veľmi užitočným využitím takéhoto paralelného priradenia je napr. výmena obsahov dvoch premenných:

pi = 3.14

ab = 'bod A'

pi, ab = ab, pi

Paralelné priradenie totiž funguje takto:

* najprv sa zistí postupnosť všetkých hodnôt na pravej strane priraďovacieho príkazu (bola to dvojica ab, pi a teda hodnoty 'bod A', 3.14)
* tieto dve zapamätané hodnoty sa naraz priradia do dvoch premenných pi, ab, teda sa vymenia ich obsahy

Paralelné priradenie funguje aj v prípade, že na pravej strane príkazu je jediný znakový reťazec nejakej dĺžky a na ľavej strane príkazu je presne toľko premenných ako je počet znakov, napr.

>>> a, b, c, d, e, f = 'Python'

>>> print(a, b, c)

P y t

**Úlohy:**

50

30

[x, y]

20

40

60

10

10

10

10

10

10

10

10

100

50

40

1. Nakreslite štvorec, ktorého pravý dolný roh má súradnice x, y a veľkosť strany je a.

[x, y]

a

1. Nakreslite kružnicu so stredom v x, y a s polomerom r.

[x, y]

r

1. Nakreslite pravouhlý trojuholník, ktorého vrchol s pravým uhlom je v x, y a odvesny majú dĺžky d1 a d2.

[x, y]

d1

d2

1. Nakreslite dve dotýkajúce sa kružnice s polomerom r1 a r2, pričom stred prvej je v x, y a druhá má rovnakú y súradnicu.
2. Nakreslite štvorec, u ktorého poznáme súradnice x1, y1 a x2, y2 spodných dvoch vrcholov – predpokladajte, že y1 = y2.

dlzka

vyska

[x1, y1]

[x2, y1]

1. Nakreslite rovnoramenný trojuholník, ak je známa dĺžka spodnej strany dlzka, výška vyska a súradnice jedného vrcholu (pozor bude treba deliť dvoma, výpočet sa bude líšiť od voľby vrcholu).
2. Nakreslite robota. Ako základný bod si určíme napr. ľavý horný bod tela obdĺžnika, ten si všeobecne označíme ako bod so súradnicou [x, y] a všetky ďalšie body budú mať súradnice podľa toho, ako ďaleko sa nachádzajú od tohto bodu. Na obrázku vidíme označený základný bod a veľkosti jednotlivých častí obrázku.

**Riešenie jednotlivých úloh:**

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

#1 **premenne**

a, x, y = 70, 80, 80

canvas.create\_rectangle(x, y, x-a, y-a)

#2

r, x, y = 50, 150, 60

canvas.create\_oval(x-r, y-r, x+r, y+r)

#3

x, y, d1, d2 = 10, 170, 50, 70

canvas.create\_polygon(x, y, x+d1, y, x, y-d2, fill='red')

#4

r1, r2, x, y = 50, 70, 150, 170

canvas.create\_oval(x-r1, y-r1, x+r1, y+r1)

canvas.create\_oval(x+r1+r2-r2, y-r2, x+r1+r2+r2, y+r2)

#5

x1, y1, x2, y2 = 10, 230, 50, 230

canvas.create\_rectangle(x1, y1, x2, y2-(x2-x1))

#6

x, y, dlzka, vyska = 250, 90, 70, 80

canvas.create\_polygon(x, y, x+dlzka, y, x+dlzka/2, y-vyska, fill='red')

import tkinter

canvas = tkinter.Canvas(bg='white', width=200, height=300)

canvas.pack()

#**robot**

x, y = 70, 70

#hlava

canvas.create\_rectangle(x+10, y-50, x+50, y, fill='dark olive green')

#oci

canvas.create\_rectangle(x+15, y-40, x+25, y-30, fil='sky blue')

canvas.create\_rectangle(x+35, y-40, x+45, y-30, fil='sky blue')

#usta

canvas.create\_rectangle(x+15, y-20, x+45, y-10, fil='red')

#trup

canvas.create\_rectangle(x, y, x+60, y+100, fill='dark olive green')

#nohy

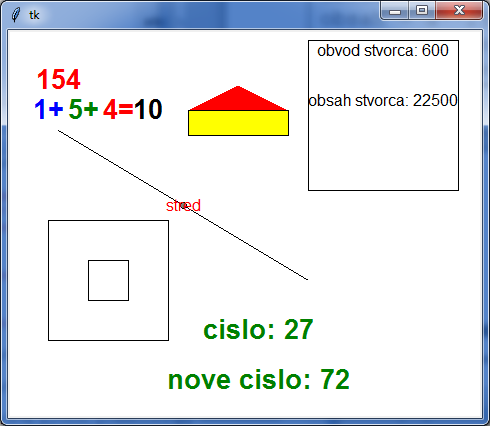
canvas.create\_rectangle(x, y+100, x+20, y+180, fill='dark olive green')

canvas.create\_rectangle(x+40, y+100, x+60, y+180, fill='dark olive green')

#ruky

canvas.create\_line(x, y+10, x-40, y+50, width=15)

canvas.create\_line(x+60, y+10, x+100, y+50, width=15)

**Ďalšie úlohy**

1. Rozoberte trojciferné číslo na cifry. Do grafickej plochy každú vypíšte inou farbou a zistíte ich súčet, ktorý tiež vypíšte do grafickej plochy.
2. Nasledujúci program nakreslí domček, zložený z obdĺžnika a rovnoramenného trojuholníka, ktorého výška je rovnaká ako výška obdĺžnika. Pritom výška obdĺžnika je štvrtina šírky.
3. Nakreslite úsečku, vypočítajte jej stred. Presne do stredu nakreslite kruh s polomerom 3 a napíšte 'stred'
4. Nakreslite štvorec, potom v jeho vnútri presne v strede nakreslite štvorec s tretinovou veľkosťou strany.
5. Zadané dvojciferné číslo rozoberte na cifry a potom z nich zložte prevrátené číslo. Toto prevrátené číslo vypíšte.
6. Program nakreslí štvorec so stranou A, potom vypíše jeho obsah a obvod.

**Pretypovanie hodnôt**

Mená typov int, float a str zároveň súžia ako mená pretypovacích funkcií, ktoré dokážu z jedného typu vyrobiť hodnotu iného typu:

* int(hodnota) z danej hodnoty vyrobí celé číslo, napr.
  + int(3.14) => 3
  + int('37') => 37
* float(hodnota) z danej hodnoty vyrobí desatinné číslo, napr.
  + float(333) => 333.0
  + float('3.14') => 3.14
* str(hodnota) z danej hodnoty vyrobí znakový reťazec, napr.
  + str(356) => '356'
  + str(3.14) => '3.14'

Zrejme pretypovanie reťazca na číslo bude fungovať len vtedy, keď je to správne zadaný reťazec, inak funkcia vyhlási chybu.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

#1 ciferny sucet

cislo = 154

canvas.create\_text(50,50,text=str(cislo), font='Arial 20 bold', fill='red')

cifra1 = cislo // 100

cislo -= cifra1\*100

cifra2 = cislo //10

cifra3 = cislo % 10

sucet = cifra1 + cifra2 + cifra3

canvas.create\_text(40, 80, text=str(cifra1)+'+',font='Arial 20 bold', fill='blue' )

canvas.create\_text(75, 80, text=str(cifra2)+'+',font='Arial 20 bold', fill='green')

canvas.create\_text(110, 80, text=str(cifra3)+'=',font='Arial 20 bold', fill='red')

canvas.create\_text(140, 80, text=str(sucet),font='Arial 20 bold', fill='black')

#domcek

sirka = 100

vyska = sirka / 4

x, y = 180, 80

canvas.create\_rectangle(x, y, x+sirka, y+vyska, fill='yellow')

canvas.create\_polygon(x, y, x+sirka, y, x + sirka/2, y-vyska, fill='red')

#usecka a stred

x1, y1, x2, y2 = 50, 100, 300, 250

canvas.create\_line(x1, y1, x2, y2)

xs = (x1+x2)/2

ys = (y1+y2)/2

canvas.create\_oval(xs-3, ys-3, xs+3, ys+3, fill='gray')

canvas.create\_text(xs, ys, text='stred', fill='red', font='bold')

#stvorec a v jeho vnutri dalsi stvorec

x, y, a = 100, 250, 60

a3 = a/3

canvas.create\_rectangle(x-a, y-a, x+a, y+a)

canvas.create\_rectangle(x-a3, y-a3, x+a3, y+a3)

#prevratene cislo

cislo = 27

cifra1 = cislo//10

cifra2 = cislo%10

prevratenecislo = cifra2\*10+cifra1

canvas.create\_text(250, 300, text='cislo: '+str(cislo), font='Arial 20 bold',

fill='green')

canvas.create\_text(250, 350, text='nove cislo: '+str(prevratenecislo),

font='Arial 20 bold', fill='green')

#obvod a obsah stvorca

x, y, a = 300, 10, 150

canvas.create\_rectangle(x, y, x+a, y+a)

obvod = 4\*a

obsah = a\*\*2

canvas.create\_text(375, 20, text='obvod stvorca: '+str(obvod), font="bold")

canvas.create\_text(375, 70, text='obsah stvorca: '+str(obsah), font="bold")

**Generátor náhodných čísel**

V mnohých našich programoch sa nám môže hodiť, keď niektoré premenné nebudú mať pri každom spustení rovnakú hodnotu, ale zakaždým dostanú podľa nejakých pravidiel nejakú náhodnú hodnotu. Využijeme modul random, ktorý obsahuje niekoľko užitočných funkcií. Aby sme mohli používať funkcie z tohto modulu, musíme na začiatku programu zapísať

**import** **random**

Vznikne premenná random, ktorá obsahuje referenciu na tento modul a pomocou nej budeme pristupovať k funkciám v tomto module. Každá takáto funkcia bude teda začínať random a za bodkou bude uvedené meno funkcie.

Modul obsahuje niekoľko užitočných funkcií, my budeme najčastejšie používať tieto tri.

**funkcia random.randint()**

* funkcia má dva parametre: hranice intervalu čísel
* vyberie náhodnú hodnotu z tohto intervalu, pričom sa do výberu počítajú aj hraničné body intervalu

**funkcia random.randrange()**

* funkcia má 1, 2 alebo 3 parametre s rovnakým významom ako range()
* vyberie náhodnú hodnotu z tohto rozsahu
* napr. random.randrange(100), random.randrange(10,100), random.randrange(10,100,5)

**poznámka** funkcia range() – pomocou tejto funkcie budeme najmä vo for-cykloch generovať rôzne postupnosti hodnôt. Táto funkcia môže mať rôzny počet parametrov a podľa toho sa bude generovať výsledná postupnosť. Všetky parametre musia byť celočíselné.

range(stop), range(start, stop), range(start, stop, krok)

Parametre:

* **start** – prvý prvok vygenerovanej postupnosti, (ak chýba, predpokladá sa 0)
* **stop** – hodnota, na ktorej sa už generovanie ďalšej hodnoty postupnosti zastaví – táto hodnota už v postupnosti nebude
* **krok** – hodnota, o ktorú sa zvýši (resp. zníži pre záporný krok) každý nasledovný prvok postupnosti, ak tento parameter chýba, predpokladá sa 1

Najlepšie si to ukážeme na príkladoch – rôzne vygenerované postupnosti celých čísel. V tabuľke vidíme výsledky pre rôzne parametre.

|  |  |
| --- | --- |
| 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 |
| range(10, 1) | *prázdna postupnosť* |
| range(1, 1) | *prázdna postupnosť* |

**funkcia random.choice()**

* funkcia má jeden parameter: ľubovoľnú postupnosť hodnôt
* vyberie náhodnú hodnotu z tejto postupnosti
* napr. random.choice(('red','blue','green'))

**Úlohy:**

1. Na náhodnej pozícii sa nakreslí štvorec veľkosti 100x100, ktorý má náhodnú hrúbku strán od 1 do10
2. Na náhodnej pozícii grafickej plochy sa nakreslí náhodne veľký kruh od 10 do 30 tak, aby sa celý zmestil do grafickej plochy.
3. Nakreslia sa 2 kruhy tesne vedľa seba na náhodných pozíciách, pričom prvý má náhodný polomer od 10 do 30 a druhý od 20 do 50.
4. Nakreslí sa náhodný trojuholník, ktorého vrcholy sú od okrajov plochy vzdialené aspoň na 20. Výplň trojuholníka sa vyberá z troch farieb.
5. Na náhodnej pozícii sa nakreslia tri vpísané kruhy tak, že najväčší má náhodná polomer r1 od 70 do 100, druhý kruh má náhodný polomer r2 od 40 do r1-5 a najmenší má náhodný polomer od 10 do r2-5. Vyberte im aj náhodnú farbu z postupnosti aspoň 5 farieb:
6. Program, ktorý vymyslí štvormiestny PIN pre náš mobilný telefón: program by mal náhodne vygenerovať 4 čísla z intervalu <1, 9> a vypísať ich vedľa seba.

**Riešenie:**

Poznámka – aby sme lepšie videli, kde sa kreslia útvary, môžeme do spusteného programu (už je v ňom vytvorené okno s grafickou plochou a importovaný modul random) zadávať aj príkazy s príkazového riadku (v okne shell). Hneď po potvrdení príkazu budeme vidieť, čo sa udeje v grafickej ploche.

Práve na príkazovom riadku si môžeme vyskúšať, ktoré náhodné čísla bude počítač žrebovať. Ak by sme nespúšťali žiaden program, v ktorom importujeme modul na náhodné čísla, môžeme modul importovať aj priamo z príkazového riadku.´

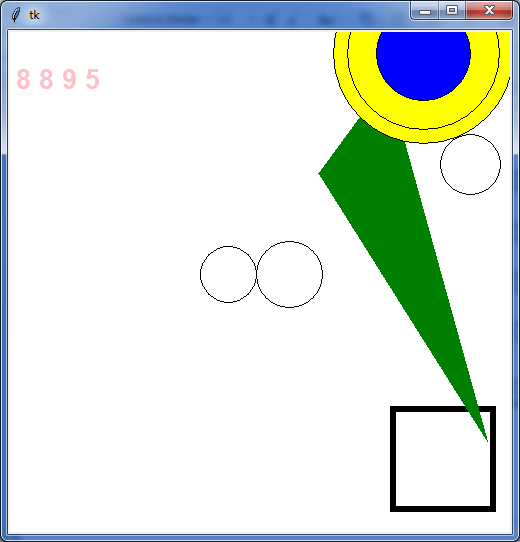
Šírku a výšku grafickej plochy určuje canvas['width'] a canvas['height'], ale tieto funkcie vrátia reťazec, preto ho musíme pretypovať ak ho chceme používať ako číslo a vložiť do premenných napr.

sirka = int(canvas['width'])

vyska = int(canvas['height'])

import tkinter, random

canvas = tkinter.Canvas(bg='white', width='500', height='500')

canvas.pack()

# nahodny stvorec

sirka = int(canvas['width'])

vyska = int(canvas['height'])

x = random.randint(0, sirka)

y = random.randint(0, vyska)

hrubka = random.randint(1, 10)

canvas.create\_rectangle(x, y, x+100, y+100, width=hrubka)

# nahodhny kruh

x = random.randint(0, sirka)

y = random.randint(0, vyska)

r = random.randint(10, 30)

canvas.create\_oval(x-r, y-r, x+r, y+r)

# 2 kruhy vedla seba

x = random.randint(0, sirka)

y = random.randint(0, vyska)

r1 = random.randint(10, 30)

r2 = random.randint(20, 50)

canvas.create\_oval(x-r1, y-r1, x+r1, y+r1)

canvas.create\_oval(x+r1+r2-r2, y-r2, x+r1+r2+r2, y+r2)

# nahodny trojuholnik

x1 = random.randint(20, sirka-20)

y1 = random.randint(20, vyska-20)

x2 = random.randint(20, sirka-20)

y2 = random.randint(20, vyska-20)

x3 = random.randint(20, sirka-20)

y3 = random.randint(20, vyska-20)

farba = random.choice(('red', 'blue', 'green'))

canvas.create\_polygon(x1, y1, x2, y2, x3, y3, fill=farba)

# vpisane kruhy

x = random.randint(0, sirka)

y = random.randint(0, vyska)

r1 = random.randint(70, 100)

r2 = random.randint(40, r1-5)

r3 = random.randint(10, r2-5)

farba = random.choice(('red', 'blue', 'green', 'yellow', 'violet'))

canvas.create\_oval(x-r1, y-r1, x+r1, y+r1, fill=farba)

farba = random.choice(('red', 'blue', 'green', 'yellow', 'violet'))

canvas.create\_oval(x-r2, y-r2, x+r2, y+r2, fill=farba)

farba = random.choice(('red', 'blue', 'green', 'yellow', 'violet'))

canvas.create\_oval(x-r3, y-r3, x+r3, y+r3, fill=farba)

#PIN

C1 = random.randint(1, 9)

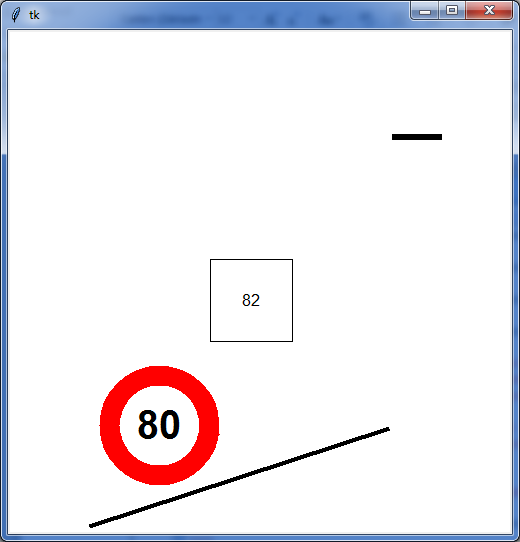
C2 = random.randint(1, 9)

C3 = random.randint(1, 9)

C4 = random.randint(1, 9)

canvas.create\_text(50, 50, text=str(C1)+' '+str(C2)+' '+str(C3)+ ' '+str(C4),

font='Arial 20 bold', fill='pink')

**Ďalšie úlohy:**

1. Na náhodnom mieste nakreslite štvorec s náhodnou stranou od 70 do 150. Upravte program tak, aby sme vo štvorci videli aj vyžrebovanú veľkosť štvorca.
2. Vytvorte program, ktorý bude kresliť náhodnú čiaru s náhodnou hrúbkou.
3. Vytvorte program, ktorý bude na náhodnom mieste kresliť vodorovnú čiaru s dĺžkou 50 bodov.
4. Na náhodnom mieste nakreslite značku s maximálne povolenou rýchlosťou (30, 40, 60, 80). Biely kruh s červeným obrysom a čiernym číslom uprostred.
5. Upravte program s robotom tak, aby ho kreslil na náhodnom mieste.

**Riešenie:**

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

#nahodny stvorec

x = random.randint(0, sirka)

y = random.randint(0, vyska)

a = random.randint(70, 150)

canvas.create\_rectangle(x-a/2, y-a/2, x+a/2, y+a/2)

canvas.create\_text(x, y, text=str(a), font='bold')

#nahodna usecka

x1 = random.randint(0, sirka)

y1 = random.randint(0, vyska)

x2 = random.randint(0, sirka)

y2 = random.randint(0, vyska)

hrubka = random.randint(1, 10)

canvas.create\_line(x1, y1, x2, y2, width=hrubka)

#vodorovna usecka

x1 = random.randint(0, sirka)

y1 = random.randint(0, vyska)

hrubka = random.randint(1, 10)

canvas.create\_line(x1, y1, x1+50, y1, width=hrubka)

#dopravna znacka

x = random.randint(70, sirka-70)

y = random.randint(70, vyska-70)

##**dve zatvorky v choise –pozor**

rychlost = random.choice((30, 40, 60, 80))

canvas.create\_oval(x-50, y-50, x+50, y+50, outline='red', width=20, fill='white')

canvas.create\_text(x, y, text=str(rychlost), font='Arial 30 bold')

**Opakovanie časti programu – for cyklus**

Ak potrebujeme niečo v programe opakovať použijeme na to konštrukciu for-cyklu:

for i in (1, 2, 3, 4, 5):

print('programujem v Pythone')

alebo

for i in 1, 2, 3, 4, 5:

print('programujem v Pythone')

Ako to funguje:

* 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 'programujem v Pythone', 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

**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
* niekedy sa môže hodiť **prázdny príkaz** pass, ktorý nerobí nič, len oznámi čitateľovi, že sme na telo cyklu nezabudli, ale zatiaľ tam nechceme mať nič
* prázdne riadky v tele cyklu nemajú žiaden význam, často slúžia na sprehľadnenie čitateľnosti kódu

Podobný výpis dostaneme aj takýmto zápisom for-cyklu:

for i in 1, 1, 1, 1, 1:

print('programujem v Pythone')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

ale ak by sme ale druhý riadok tela cyklu neodsunuli:

for i in 1, 1, 1, 1, 1:

print('programujem v Pythone')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

telo cyklu je teraz len jeden príkaz. Program najprv 5-krát vypíše text 'programujem v Pythone' a až potom jeden riadok s hviezdičkami:

V tele cyklu môžeme použiť aj **premennú cyklu**, ktorej hodnota sa pri každom prechode cyklom automaticky mení. V nasledovnom príklade je premennou cyklu prem:

for prem in (1, 2, 3, 4, 5):

print(prem)

Po spustení programu sa postupne vypíšu všetky nadobudnuté hodnoty:

Premenná cyklu nemusí nadobúdať len celočíselné hodnoty, ale hodnoty úplne ľubovoľného typu (aj desatinné čísla, aj reťazce).

Vo for-cykloch budeme využívať hlavne funkciu range().

For-cyklus sa výborne využije v prípadoch, keď sa v tele cyklu vyskytuje príkaz, ktorý inkrementuje nejakú premennú (alebo ju mení inou operáciou). Zrejme toto inkremenovanie prejde toľkokrát, koľko je prechodov cyklu a často sa pritom využije premenná cyklu.

Ukážme túto ideu na príklade, v ktorom spočítame všetky hodnoty premennej cyklu. Použijeme na to ďalšiu premennú, do ktorej sa bude postupne pripočítavať premenná cyklu:

sucet = 0

for i in range(1, 11):

sucet = sucet + i

print(sucet)

tento program spočíta čísla od 1 do 10.

Tento program nám môže slúžiť ako nejaká **šablóna**, pomocou ktorej riešime podobnú triedu úloh: v každom prechode cyklu niečo pripočítavame, násobime, delíme, ... Takáto **pripočítavacia šablóna** sa skladá z:

* inicializácia pomocnej premennej (alebo viacerých premenných), do ktorej sa bude *pripočítavať*, najčastejšie je to vynulovanie; (ak budeme násobiť tak do pomocnej premennej na začiatku vložíme 1)
* v tele cyklu sa do tejto pomocnej premennej *pripočíta* nejaká hodnota, najčastejšie premenná cyklu;
* po skončení cyklu sa v tejto pomocnej premennej nachádza očakávaný výsledok *sčitovania*.

Môžeme používať aj vnorené cykly, pričom vnorený cyklus musí byť odsadený o ďalšie 4 medzery.

Poznámka: knižnicu random môžeme naimportovať aj príkazom from random import \* - potom už nemusíme používať zápis random.randint(10, 50) ale len randint(10, 50). Pri nasledujúcich úlohách to môžeme použiť.

**Úlohy:**

1. Nakreslite 50 náhodných čiar.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

for i in range(1, 51):

x1 = random.randint(0, sirka)

y1 = random.randint(0, vyska)

x2 = random.randint(0, sirka)

y2 = random.randint(0, vyska)

canvas.create\_line(x1, y1, x2, y2)

1. Vypíšte 30x na náhodnú pozíciu text Python.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

for i in range(1, 31):

x = random.randint(15, sirka-15)

y = random.randint(15, vyska-15)

canvas.create\_text(x, y, text='Python')

1. Program najprv nakreslí modrú krivku z 10 náhodných úsečiek a potom červenú z 10 náhodných úsečiek.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

x1 = random.randint(0, sirka)

y1 = random.randint(0, vyska)

for i in range(1, 11):

x2 = random.randint(0, sirka)

y2 = random.randint(0, vyska)

canvas.create\_line(x1, y1, x2, y2, fill='blue')

x1, y1 = x2, y2

x1 = random.randint(0, sirka)

y1 = random.randint(0, vyska)

for i in range(1, 11):

x2 = random.randint(0, sirka)

y2 = random.randint(0, vyska)

canvas.create\_line(x1, y1, x2, y2, fill='red')

x1, y1 = x2, y2

1. Vypíšte postupne pod seba čísla od 11 do 30.
2. Vypíšte postupne pod seba čísla od 11 do 30 a ku každému pripíšte jeho druhú mocninu.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

x = 50

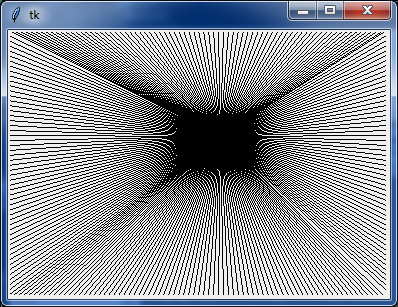
y = 10

for i in range(11, 31):

canvas.create\_text(x, y, text=str(i)+' '+ str(i\*\*2))

y += 12

1. Najprv náhodne vygenerujte bod x, y a potom ho postupne spájajte s bodmi na spodnej hrane plochy (prípadne hornej, alebo bočných hrán plochy).

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

x = random.randint(0, sirka)

y = random.randint(0, vyska)

for i in range(0, sirka, 5):

canvas.create\_line(x, y, i, vyska)

canvas.create\_line(x, y, i, 0)

for i in range(0, vyska, 5):

canvas.create\_line(x, y, sirka, i)

canvas.create\_line(x, y, 0, i)

1. Vyšrafujte obdĺžnik (ktorý je od okrajov plochy vzdialený 20 bodov) vodorovnými úsečkami, medzi ktorými je vzdialenosť 10.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

canvas.create\_rectangle(20, 20, sirka-20, vyska-20)

y = 20

for i in range(20, vyska-20, 10):

canvas.create\_line(20, y, sirka-20, y)

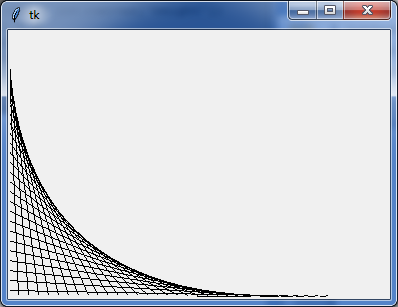
y += 10

#pozdrzanie programu

canvas.update()

canvas.after(500)

1. Program kreslí úsečky, v ktorých sa prvý bod pohybuje po zvislej priamke ([0, 0], [0, 10], [0, 20], …) a druhý po vodorovnej na spodnom okraji plochy (napr. [0,300], [10, 300], [20, 300], …)

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

y1 = 0

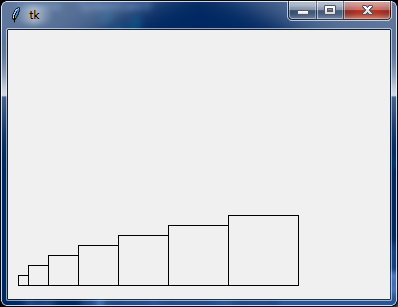
x2 = 0

for i in range(0, (sirka + vyska)//2, 10):

canvas.create\_line(0, y1, x2, vyska)

y1 += 10

x2 += 10

1. Nakreslite 7 postupne sa zväčšujúcich štvorcov pričom sú všetky položené na tej istej y-ovej súradnici a veľkosti strán sú postupne 10, 20, 30, 40, ...

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

vyska = int(canvas['height'])-10

x = 10

a = 10

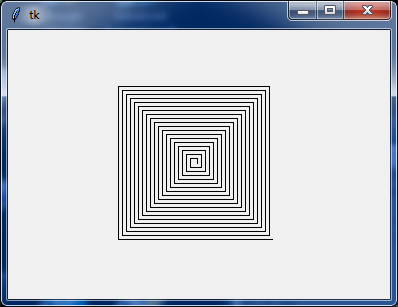
for i in range(1, 8):

canvas.create\_rectangle(x, vyska-a, x+a, vyska)

x += a

a += 10

1. Nakreslite štvorcovú špirálu, ktorá začína v strede grafickej plochy: v cykle sa bude opakovať kreslenie štyroch na seba kolmých úsečiek: každá bude od predchádzajúcej o kúsok dlhšia, otočená o 90°.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

xs = sirka/2

ys = vyska/2

dlzka = 5

x1 = xs

y1 = ys

x2 = x1

y2 = y1 - dlzka

for i in range(1, 20):

canvas.create\_line(x1, y1, x2, y2)

canvas.update()

canvas.after(100)

dlzka += 2

x1, y1 = x2, y2

x2 -= dlzka

canvas.create\_line(x1, y1, x2, y2)

canvas.update()

canvas.after(100)

dlzka += 2

x1, y1 = x2, y2

y2 +=dlzka

canvas.create\_line(x1, y1, x2, y2)

canvas.update()

canvas.after(100)

dlzka += 2

x1, y1 = x2, y2

x2 +=dlzka

canvas.create\_line(x1, y1, x2, y2)

canvas.update()

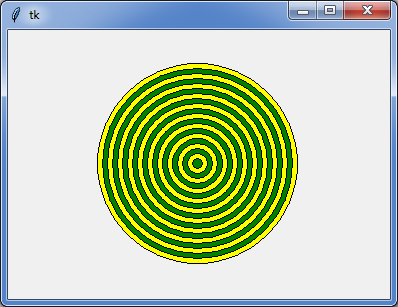
canvas.after(100)

dlzka += 2

x1, y1 = x2, y2

y2 -=dlzka

1. Nakreslite 20 sústredných kruhov, pre ktoré sa strieda farba ich výplne, napr. žltá a zelená.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

f1 = 'yellow'

f2 = 'green'

xs = sirka/2

ys = vyska/2

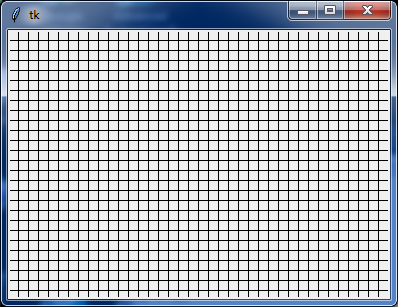
r = 100

for i in range(1, 21):

canvas.create\_oval(xs-r, ys-r, xs+r, ys+r, fill=f1)

r -= 5

f1, f2 = f2, f1

1. Nakreslite štvorčekový papier.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

x = 0

y = 0

a = 10

for j in range(0, vyska, 10):

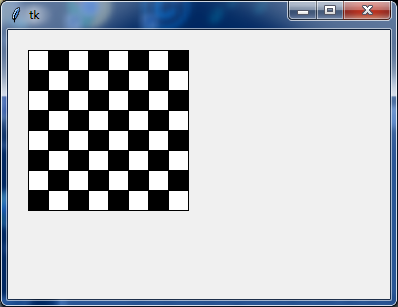
for i in range(0, sirka, 10):

canvas.create\_rectangle(x, y, x+a, y+a)

x += a

x = 0

y += a

1. Nakreslite šachovnicu 8x8 – striedajú sa dve farby výplne biela čierna.

import tkinter, random

canvas = tkinter.Canvas()

canvas.pack()

x = 20

y = 20

f1 = 'white'

f2 = 'black'

a = 20

for j in range(1, 9):

for i in range(1, 9):

canvas.create\_rectangle(x, y, x+a, y+a, fill=f1)

f1, f2 = f2, f1

x += a

x = 20

y += a

f1, f2 = f2, f1

**Poznámka**: For cyklus opakuje príkazy veľmi rýchlo, my teda vidíme hneď výsledok. Ak by sme chceli vykresľovanie spomaliť, môžeme v cykle použiť príkaz canvas.after(1000). Tento príkaz pozdrží vykonávanie programu na čas uvedený v zátvorke. Čas uvádzame v milisekundách, čiže canvas.after(1000) čaká 1 sekundu. K tomuto príkazu musíme ešte použiť príkaz canvas. update(), ktorý zabezpečí, aby sa aj medzi čakaniami aktualizovalo grafické plátno. Ak by sme ho nepoužili, program by pracoval pomalšie (aj s čakaním) a grafické plátno by sa nám prekreslilo až po skončení for cyklu.

Ak chcem odsadiť vnorený for cyklus tak Format/Indent Region

**Podprogramy – funkcie**

Pri kreslení komplikovanejších obrázkov sa náš program stáva neprehľadným, pretože obsahuje veľa príkazov. Sprehľadniť program nám pomáha používanie komentárov. ďalší spôsob je použitie podprogramu. Je to menší kúsok programu, ktorý pomenujeme a keď ho chceme použiť, len napíšeme v programe meno tohto programu. V podstate si takto vytvoríme vlastné príkazy, ktoré môžeme používať v našom programe. Odborne sa takýmto programom V Pythone hovorí funkcie.

Doteraz sme pracovali so štandardnými funkciami, napr.

* generovanie postupnosti čísel pre for-cyklus range()

Všetky tieto funkcie niečo vykonali (vypísali, prečítali, vypočítali, ...) a niektoré z nich vrátili nejakú hodnotu, ktorú sme mohli ďalej spracovať. Tiež sme videli, že niektoré majú rôzny počet parametrov, prípadne sú niekedy volané bez parametrov.

Okrem toho sme pracovali aj s funkciami, ktoré boli definované v iných moduloch:

* keď napíšeme import random, môžeme pracovať napr. s funkciami random.randint() a random.randrange()
* keď napíšeme import math, môžeme pracovať napr. s funkciami math.sin() a math.cos()

Všetky tieto a tisícky ďalších v Pythone naprogramovali programátori pred nejakým časom, aby nám neskôr zjednodušili samotné programovanie. Vytváranie vlastných funkcií pritom vôbec nie je komplikované a teraz sa to naučíme aj my.

**Funkcie**

Funkcia je pomenovaný blok príkazov (niekedy sa tmu hovorí aj podprogram). Popisujeme (**definujeme**) ju špeciálnou konštrukciou:

**def** meno\_funkcie():

prikaz

prikaz

...

Keď zapíšeme definíciu funkcie, zatiaľ sa nič z bloku príkazov (hovoríme tomu **telo funkcie**) nevykoná. Táto definícia sa “len” zapamätá a jej **referencia** sa priradí k zadanému menu - vlastne sa do premennej meno\_funkcie priradí referencia na telo funkcie. Je to podobné tomu, ako sa priraďovacím príkazom do premennej priradí hodnota z pravej strany príkazu.

Poznámka: celý blok príkazov je odsunutý o 4 medzery rovnako ako sme odsúvali príkazy v cykloch.  Definícia tela funkcie končí vtedy, keď sa objaví riadok, ktorý už nie je odsunutý. Touto definíciou sa ešte žiadne príkazy z tela funkcie nevykonávajú. Na to potrebujeme túto funkciu **zavolať.**

**Volanie funkcie**

Volanie funkcie je taký zápis, ktorým sa začnú vykonávať príkazy z definície funkcie. Stačí zapísať meno funkcie so zátvorkami a funkcia sa spustí: meno\_funkcie()

Samozrejme, že funkciu môžeme zavolať až vtedy, keď už Python pozná jej definíciu.

Zapíšme teraz presné kroky, ktoré sa vykonajú pri volaní funkcie:

1. preruší sa vykonávanie programu (Python si presne zapamätá miesto, kde sa to stalo)
2. skočí sa na začiatok volanej funkcie
3. postupne sa vykonajú všetky príkazy
4. keď sa príde na koniec funkcie, zrealizuje sa **návrat** na zapamätané miesto, kde sa prerušilo vykonávanie programu

Pre volanie funkcie sú veľmi dôležité okrúhle zátvorky. Bez nich to už nie je volanie, ale len zisťovanie referencie na hodnotu, ktorá je priradená pre toto meno.

Ak by sme namiesto volania funkcie takto zapísali len meno funkcie bez zátvoriek, ale v programe (nie v interaktívnom režime), táto hodnota referencie by sa nevypísala, ale odignorovala. Toto býva dosť častá chyba, ktorá sa ale ťažšie odhaľuje.

Ak zavoláme funkciu, ktorú sme ešte nedefinovali, Python vyhlási chybu

Hotové funkcie, s ktorými sme doteraz pracovali, napr. random.randin(), mali aj parametre, vďaka čomu riešili rôzne úlohy. Parametre slúžia na to, aby sme mohli funkcii lepšie oznámiť, čo špecifické má urobiť: čo sa má vypísať, z akého intervalu má vygenerovať náhodné číslo, akú úsečku má nakresliť, prípadne akej farby,

**Parametre funkcie**

Parametrom funkcie je **dočasná premenná**, ktorá vzniká pri volaní funkcie a prostredníctvom ktorej, môžeme do funkcie poslať nejakú hodnotu. Parametre funkcií definujeme počas definovania funkcie v **hlavičke funkcie** a ak ich je viac, oddeľujeme ich čiarkami:

**def** meno\_funkcie(parameter):

prikaz

prikaz

...

V prvom riadku definície funkcie (hlavička funkcie) pribudla jedna premenná - parameter. Táto premenná vznikne automaticky pri volaní funkcie, preto musíme pri volaní oznámiť hodnotu tohto parametra.

Už predtým sme popísali mechanizmus volania funkcie, ale to sme ešte nepoznali parametre. Teraz doplníme tento postup o spracovanie parametrov. Najprv trochu terminológie:

* pri definovaní funkcie v hlavičke funkcie uvádzame tzv. **formálne parametre**: sú to nové premenné, ktoré vzniknú až pri volaní funkcie
* pri volaní funkcie musíme do zátvoriek zapísať hodnoty, ktoré sa stanú tzv. **skutočnými parametrami**: tieto hodnoty sa pri volaní priradia do formálnych parametrov

**Mechanizmus volania** :

1. zapamätá sa návratová adresa volania
2. vytvorí sa **nová** premenná (**formálny parameter**) a priradí do nej hodnora **skutočného parametra**
3. vykonajú sa všetky príkazy v definícii funkcie (**telo funkcie**)
4. zrušia sa všetky premenné, ktoré vznikli počas behu funkcie
5. riadenie sa vráti na miesto, kde bolo volanie funkcie

Už vieme, že priraďovací príkaz vytvára premennú a referenciou ju spojí s hodnotou. Premenné, ktoré vzniknú počas behu funkcie, sa stanú **lokálnymi premennými**: budú existovať len počas tohto behu a po skončení funkcie, sa automaticky zrušia. Aj parametre vznikajú pri štarte funkcie a zanikajú pri jej skončení: tieto premenné sú pre funkciu tiež lokálnymi premennými.

Menný priestor

Aby sme lepšie pochopili ako naozaj fungujú **lokálne premenné**, musíme rozumieť, čo to je a ako funguje **menný priestor** (namespace). Najprv trochu terminológie: všetky identifikátory v Pythone sú jedným z troch typov (Python má na ne 3 rôzne tabuľky mien):

* **štandardné**, napr. int, print, ...
  + hovorí sa tomu **builtins**
* **globálne** - definujeme ich na najvyššej úrovni mimo funkcií, napr. funkcia vypis\_sucet
  + hovorí sa tomu **main**
* **lokálne** - vznikajú počas behu funkcie

Tabuľka štandardných mien je je len jedna, tiež tabuľka globálnych mien je len jedna, ale každá funkcia má svoju “súkromnú” lokálnu tabuľku mien, ktorá vznikne pri štarte (zavolaní) funkcie a zruší sa pri konci vykonávania funkcie.

Keď na nejakom mieste použijeme identifikátor, Python ho najprv hľadá (v tzv. **menných priestoroch**):

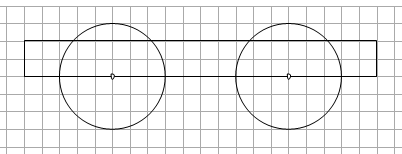
* v lokálnej tabuľke mien, ak tam tento identifikátor nenájde, hľadá ho
* v globálnej tabuľke mien, ak tam tento identifikátor nenájde, hľadá ho
* v štandardnej tabuľke mien

Ak nenájde v žiadnej z týchto tabuliek, hlási chybu NameError: name 'identifikátor' is not defined.

**Úlohy:**

Zadefinujte funkciu **vozik**, ktorá je zložená z pomocných funkcií **doska** a **koleso**. Tieto dve pomocné funkcie musia byť zadefinované ešte pre funkciou vozik, ktorá ich bude volať, a teda tieto dva identifikátory musia byť pre ňu viditeľné. Pri definovaní funkcií využite mriežku – veľkosť štvorčeka zvoľte 5.

[x, y]



Pomocou tejto funkcie a pomocou for cyklu nakreslite vedľa seba päť vozíkov. Zmeňte definíciu dosky a kolesa tak, aby doska bola modrá a kolesá červené

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=800, height=500)

canvas.pack()

a = 5

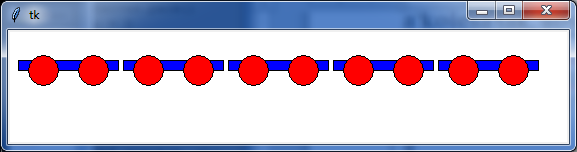
def doska(x, y):

canvas.create\_rectangle(x, y, x+20\*a, y-2\*a, fill='blue')

def koleso(x, y):

canvas.create\_oval(x-3\*a, y-3\*a, x+3\*a, y+3\*a, fill='red')

def vozik(x, y):

 doska(x, y)

koleso(x+5\*a, y)

koleso(x+15\*a, y)

x = 10

y = 40

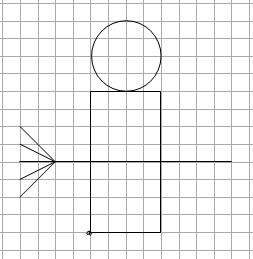
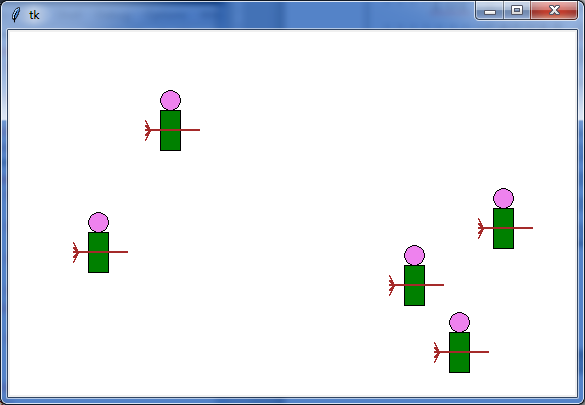
for i in (1, 1, 1, 1, 1):

vozik(x, y)

x += 21\*a

Zadefinujte funkciu bosorka. Nech sa skladá z funkcií hlava, telo, metla. Nakreslite 5 bosoriek na náhodnej pozícii.

[x, y] – tento bod nech určuje polohu bosorky



import tkinter, random

canvas = tkinter.Canvas(bg='white', width=800, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

a = 5

def hlava(x, y):

canvas.create\_oval(x-2\*a, y-2\*a, x+2\*a, y+2\*a, fill='violet')

def telo(x, y):

canvas.create\_rectangle(x, y, x+4\*a, y-8\*a, fill='green')

def metla(x, y):

canvas.create\_line(x, y, x+10\*a, y, fill='brown', width=2)

for i in (2, 1, 0, -1, -2):

canvas.create\_line(x, y, x-a, y-i\*a, fill='brown', width=2)

def bosorka(x, y):

hlava(x+2\*a, y-10\*a)

telo(x, y)

metla(x-2\*a, y-4\*a)

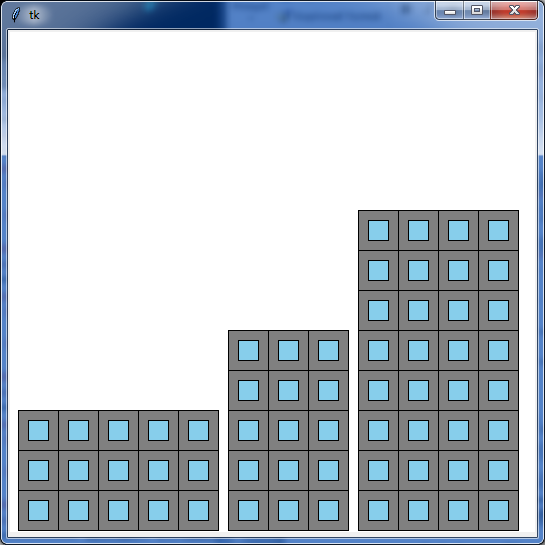
for i in range(1, 6):

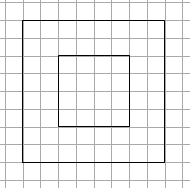
x = random.randint(40, sirka -80)

y = random.randint(120, vyska-10)

bosorka(x, y)

**Ďalšie úlohy na riešenie:**

1. Funkcia **panel** nakreslí šedý štvorec 40x40, v ktorom je bledomodrý menší štvorec (okno). Funkcia **panelak** má 4 parametre: pozíciu paneláka a počet panelov na jednom poschodí (šírka) a počet poschodí (výška). Nakreslite vedľa seba tri rôzne vysoké paneláky.



import tkinter, random

canvas = tkinter.Canvas(bg='white',

width=800, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

a = 5

def panel(x, y):

canvas.create\_rectangle(x, y, x+8\*a, y-8\*a, fill='gray')

canvas.create\_rectangle(x+2\*a, y-2\*a, x+6\*a, y-6\*a, fill='sky blue')

def panelak(x, y, s, v):

for j in range(1, v+1):

for i in range(1, s+1):

panel(x, y)

x += 8\*a

x -= (8\*a)\*s

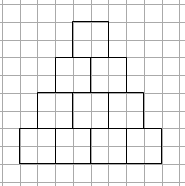
y -=8\*a

panelak(10,vyska,5, 3)

panelak(220, vyska, 3, 5)

panelak(350, vyska, 4, 8)

1. Zadefinujte funkciu **radstvorcov(x, y, n)**, ktorá nakreslí vedľa seba **n** štvorcov veľkosti 20, pričom prvý z nich začína na súradnici **[x, y]**. Pomocou tejto funkcie zadefinujte funkciu **pyramida(n)**, ktorá nakreslí pyramídu výšky **n** poschodí. Na obrázku je pyramida(4)



import tkinter, random

canvas = tkinter.Canvas(bg='white', width=800, height=500)

canvas.pack()

sirka = int(canvas['width'])

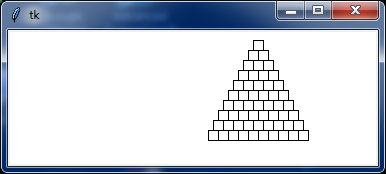
vyska = int(canvas['height'])

a = 10

def radstvorcov(x, y, n):

for i in range(1, n+1):

canvas.create\_rectangle(x, y, x+a, y+a)

 x +=a

x -=a\*n

def pyramida(x, y, n):

for i in range(1, n+1):

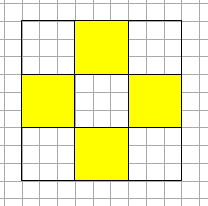
radstvorcov(x, y, n-i+1)

x += a/2

y -= a

pyramida(200, 100, 10)

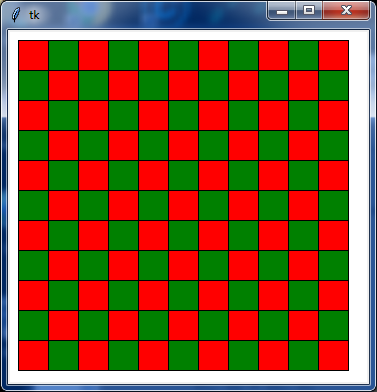
1. Funkcia **stvorec(x, y, farba)** nakreslí farebný štvorec veľkosti 30. Napíšte funkciu **sachovnica(n)**, ktorá nakreslí šachovnicu veľkosti nxn štvorcov (pomocou funkcie **stvorec**), v ktorej sa striedajú dve farby štvorcov. Na obrázku je sachovnica(3). Je asi potrebné zadať pozíciu šachovnice a farby, ktoré sa striedajú.



import tkinter, random

canvas = tkinter.Canvas(bg='white', width=800, height=500)

canvas.pack()

def stvorec(x, y, farba):

canvas.create\_rectangle(x, y, x+30, y+30, fill=farba)

def sachovnica(x, y, n, f1, f2):

for j in range(1, n+1):

for i in range(1, n+1):

stvorec(x, y, f1)

x += 30

f1, f2 = f2, f1

x -= 30\*n

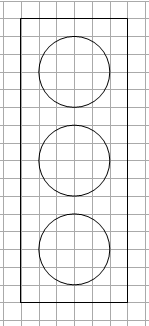
y += 30

if n%2==0:

f1, f2 = f2, f1

sachovnica(10, 10, 11, 'red', 'green')

1. Vytvorte funkciu **kruh(x, y, farba)** s polomerom 20 a pomocou tejto funkcie procedúru **semafor(x, y, farba1, farba2, farba3)** – šírka semaforu nech je 60 a výška 160. Na náhodnú pozíciu nakreslite 3 semafory.

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=300, height=300)

canvas.pack()

sirka = int(canvas['width'])

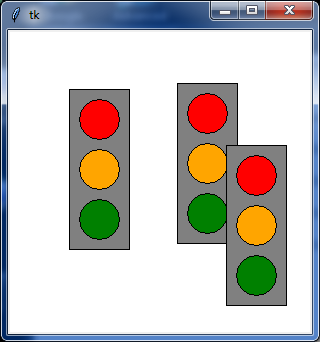
vyska = int(canvas['height'])

def kruh(x, y, farba):

canvas.create\_oval(x-20, y-20, x+20, y+20, fill=farba)

def semafor(x, y, f1, f2, f3):

canvas.create\_rectangle(x, y, x+60, y+160, fill='gray')

 kruh(x+30, y+30, f1)

kruh(x+30, y+80, f2)

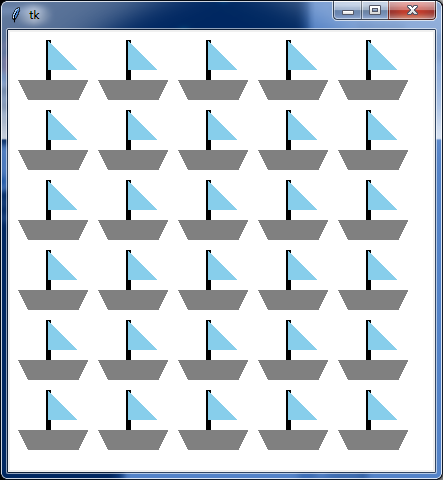
kruh(x+30, y+130, f3)

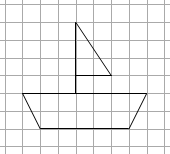
for i in range(1, 4):

x = random.randint(10, sirka-70,)

y = random.randint(10, vyska-170)

semafor(x, y, 'red', 'orange', 'green')

1. Vytvorte funkciu **lodicka(x, y)**. Potom nakreslite 5x6 lodičiek.



import tkinter, random

canvas = tkinter.Canvas(bg='white',

width=1000, height=800)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

def lodicka(x, y):

canvas.create\_polygon(x, y, x+10, y+20, x+60, y+20, x+70, y, fill='gray')

canvas.create\_line(x+30, y, x+30, y-40, width=5)

canvas.create\_polygon(x+30, y-10, x+60, y-10, x+30, y-40, fill='sky blue')

x = 10

y = 50

for j in range(1, 7):

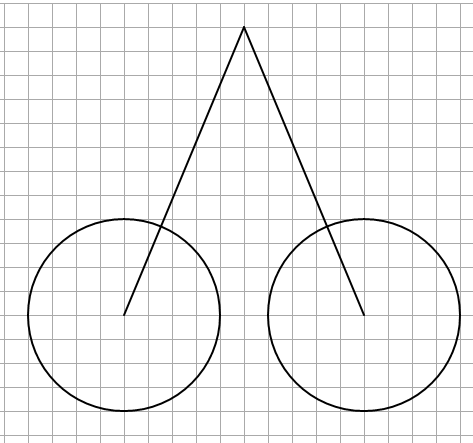
for i in range(1, 6):

lodicka(x, y)

x += 80

x = 10

y += 70

Vytvorte funkciu **ceresna(x, y)** – pozri pomocný  
obrázok (šírka štvorca nech je 1). Pri vytvorení funkcii  
**ceresna** použite pomocnú funkciu kruh**(x, y, r)**,  
pričom bod [x, y] je stred kruhu a r je polomer kruhu.

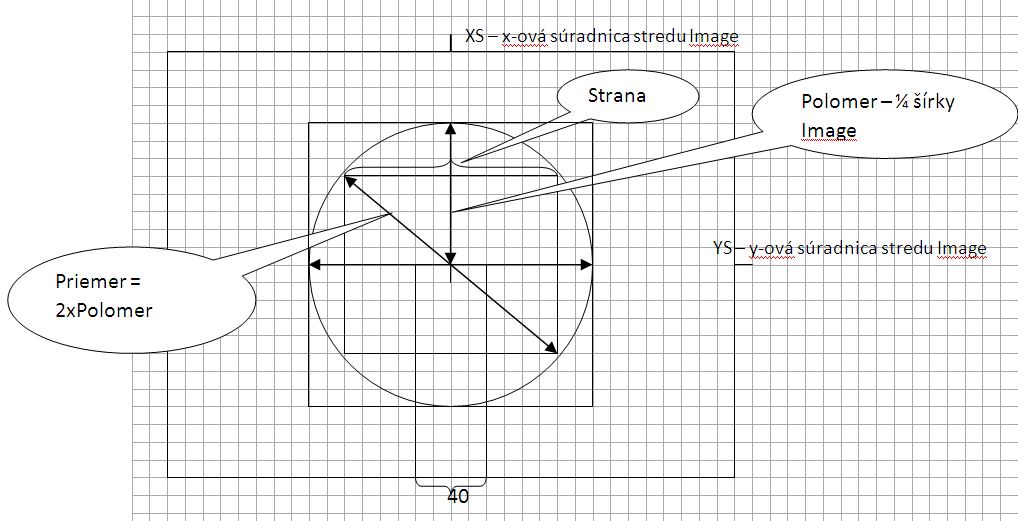
16

[x, y]

18

Potom funkciu použite na nakreslenie nasledujúcich  
stromov.

**Ďalšie pomocné obrázky a pomocné texty**



Na nakreslenie stromu použite funkciu **strom** (bez parametrov). Využite globálne premenné xs, ys, polomer, priemer, strana. Pri tvorbe tejto funkcie využite aj funkciu **kruh(x, y, r)**..

**Pri prvom strome** máte náhodne rozmiestniť 16 čerešní. Pozíciu každej čerešne generujeme náhodne do celej oblasti canvas, alebo len do oblasti štvorca opísaného korune stromu, teda x-ová súradnica pozície čerešne môže byť z intervalu <xs – polomer, xs + polomer> a y-ová súradnica z intervalu  
<ys – polomer, ys + polomer>. V každom prípade ale musíme skontrolovať či bod, ktorý sme vygenerovali leží v korune stromu t.j. či patrí kruhu so stredom [xs, ys] a polomerom polomer, teda musíme preveriť podmienku (x - xs)2  + (y - ys)2 <= polomer2. Generujeme pokiaľ počet čerešní nie je 16 – teda musíme použiť cyklus s podmienkou.

**Pri druhom strome** máme čerešne rozmiestniť rovnomerne do radu na vodorovný priemer koruny stromu. V tomto prípade musíme vypočítať veľkosť medzery medzi čerešňami. Šírka čerešne je 18 – pozri horný obrázok, počet čerešní je 7 – teda na medzery mi zostane priemer – 18x7. Počet medzier je 6 teda medzera := (priemer – 18\*7) // 6. A samozrejme musíme vedieť kde umiestnime prvú čerešňu, teda   
x := xs – polomer + 9 (9 je polovička šírky čerešne), y := ys. Pri riešení použijeme cyklus s pevným počtom opakovaní.

**Pri treťom strome** umiestňujeme čerešne na uhlopriečku (čo je priemer koruny stromu) štvorca vpísaného do koruny stromu. Teda v smere osi x na vzdialenosť strana a v smere osi y tiež na vzdialenosť strana. Znovu použijeme for –cyklus – čerešne sa posúvajú v smere osi x aj v smere osi y, aby boli čerešne rovnomerne rozmiestnené – musíme zase počítať medzeru v smere osi x medzerux a medzeru v smere osi y medzeruy. Na medzery v smere osi x máme strana – 18x6 a počet medzier je 5, teda medzerax := (strana – 18\*6) // 5 a medzeray := (strana – 16\*6) // 5 (výška čerešne je 16 – pozri horný obrázok). Nezabudnite umiestniť prvú čerešňu.

**Pri štvrtom strome** umiestnime čerešne pravidelne do štvorca, ktorého strana je o ¼ menšia ako strana, teda novastrana := strana – strana //4. Znova počítajte medzerux, medzeruy

a musíme použiť vnorený cyklus.

Piaty strom už zvládnete bez pomoci.

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

polomer = sirka/4

xs = sirka/2

ys = vyska/2

def kruh(x, y):

canvas.create\_oval(x-4, y-4, x+4, y+4, fill='red', outline='red')

def ceresna(x, y):

canvas.create\_line(x, y, x-5, y+12, width=2, fill='brown')

canvas.create\_line(x, y, x+5, y+12, width=2, fill='brown')

kruh(x-5, y+12)

kruh(x+5, y+12)

canvas.create\_rectangle(xs-20, vyska-20, xs+20, vyska-polomer-20, fill='brown')

canvas.create\_oval(xs-polomer, ys-polomer, xs+polomer, ys+polomer, fill='green')

pocet = 0

while pocet<=16:

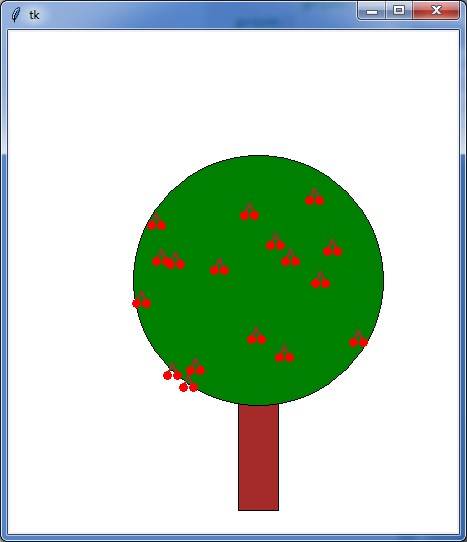
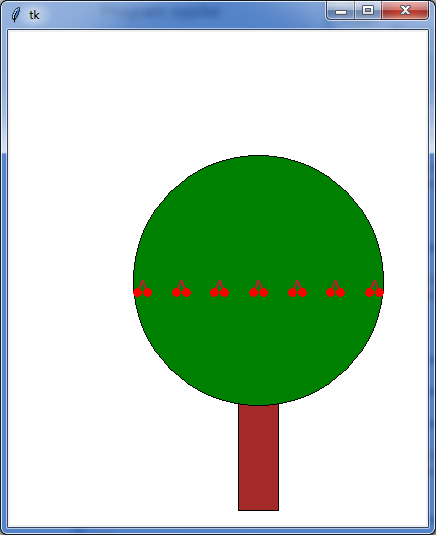
x = random.randint(xs-polomer, xs+polomer)

y = random.randint(ys-polomer, ys+polomer)

if (x-xs)\*\*2 + (y-ys)\*\*2 <=polomer\*\*2:

pocet +=1

ceresna(x, y)



Na náhodných miestach sa kreslí kvet. Kvet sa skladá z troch častí (trs trávy, stonka a kvet). Trs trávy obsahuje dvadsať náhodných stebiel. Každé steblo je od koreňa vzdialené na x-ovej súradnici maximálne 30 bodov vľavo alebo vpravo. Na y-ovej súradnici je od koreňa vzdialené maximálne 30 bodov a vždy je umiestnené smerom od koreňa.

Stonka má dĺžku 30 bodov. Kvet je žltý a obsahuje 30 čiar, ktoré vychádzajú zo stredu kvetu a sú náhodné. Každá čiara kvetu je v smere x-ovej a y-ovej súradnice najďalej 20 bodov od stredu kvetu.

#funkcie

import tkinter

import random

canvas = tkinter.Canvas(bg='white', width=1000, height=800)

canvas.pack()

def trs(x, y):

for i in range(1,21):

canvas.create\_line(x, y, x+random.randint(-30,30), y-random.randint(0,30),

fill='green',width=2)

def kvet(x, y):

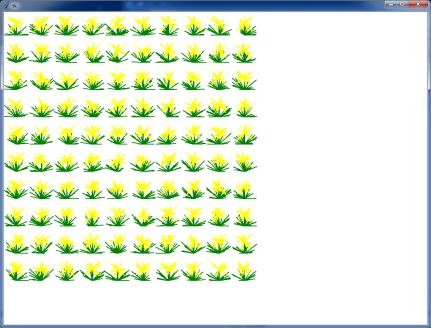
for i in range(1,31):

canvas.create\_line(x, y, x+random.randint(-20,20), y-random.randint(-20,20),

fill='yellow',width=2)

def rastlina(x, y):

trs(x, y)

 canvas.create\_line(x, y, x, y-30, width=3, fill='green')

kvet(x, y-30)

x = 30

y = 60

for i in range(1, 11):

for j in range(1, 11):

rastlina(x, y)

x = x + 60

x = 30

y = y + 70

**Poznámka**: odkrokovanie si môžeme spraviť na stránke <http://www.pythontutor.com/visualize.html#mode=edit> (zapneme voľbu Python 3.3):

* najprv do editovacieho okna zapíšeme nejaký program – grafiku asi neodkrokuje
* spustíme vizualizáciu pomocou tlačidla **Visualize Execution** a potom niekoľkokrát tlačíme tlačidlo **Forward >**

**Udalosti v grafickej ploche**

Naučíme sa v našich programoch využívať tzv. **udalosti**, ktoré vznikajú v bežiacej grafickej aplikácii buď aktivitami používateľa (klikanie myšou, stláčanie klávesov) alebo operačného systému (tikanie časovača). Na úvod si pripomeňme, čo už vieme o grafickej ploche. Pomocou metód grafickej plochy Canvas (definovanej v module tkinter) kreslíme grafické objekty:

* canvas.create\_line() - kreslí úsečku alebo krivku z nadväzujúcich úsečiek
* canvas.create\_oval() - kreslí elipsu
* canvas.create\_rectangle() - kreslí obdĺžnik
* canvas.create\_text() - vypíše text
* canvas.create\_polygon() - kreslí vyfarbený útvar zadaný bodmi na obvode

Ďalšie pomocné metódy manipulujú s už nakreslenými objektmi:

* canvas.delete() - zruší objekt
* canvas.move() - posunie objekt
* canvas.coords() - zmení súradnice objektu
* canvas.itemconfig() - zmení ďalšie parametre objektu (napr. farba, hrúbka, text, obrázok, ...)

Ďalšie metódy umožňujú postupne zobrazovať vytváranú kresbu:

* canvas.update() - zobrazí nové zmeny v grafickej ploche
* canvas.after() - pozdrží beh programu o zadaný počet milisekúnd

**Udalosť**

Udalosťou voláme akciu, 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).

Aby grafická plocha reagovala na klikania myšou, musíme ju zviazať (**bind**) s príslušnou udalosťou (**event**).

**metóda bind()**

Táto metóda grafickej plochy 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)

kde meno\_udalosti je znakový reťazec s popisom udalosti (napr. pre kliknutie tlačidlom myši) a funkcia sa spustí pri vzniku tejto udalosti. Táto funkcia, musí byť definovaná s **práve jedným parametrom,** v ktorom nám systém prezradí detaily vzniknutej udalosti.

Ukážme tieto tri “myšacie” udalosti:

* **kliknutie** (zatlačenie tlačidla myši) - reťazec '<Button-1>' - **1** označuje ľavé tlačidlo myši, **2** by tu znamenala stredné tlačidlo, **3** pravé tlačidlo
* **ťahanie** (posúvanie myšou so zatlačeným tlačidlo) - reťazec '<B1-Motion>'
* **pustenie myši** - reťazec '<ButtonRelease-1>'

## Klikanie myšou

Kliknutie myšou do grafickej plochy vyvolá udalosť s menom '<Button-1>'. Ukážme ako vyzerá samotné zviazanie funkcie:

**import** **tkinter**, **random**

canvas = tkinter.Canvas(width=600, height=450, bg='white')

canvas.pack()

**def** nahodny\_kruh(parameter):

x = random.randrange(600)

y = random.randrange(450)

canvas.create\_oval(x-10, y-10, x+10, y+10, fill='red')

canvas.bind('<Button-1>', nahodny\_kruh)

Na konci programu je príkaz bind(), ktorý má druhý parameter **referenciu na funkciu**, t.j. funkcia bez zátvoriek. Táto funkcia zrejme už musí byť definovaná skôr, ako sa použije ako parameter bind(). Funkcia, ktorá obsluhuje udalosť, musí mať jeden parameter, ktorý sa v tejto ukážke ešte nevyužíva. V tomto parametri (najčastejšie ho budeme označovať event alebo skrátene e) nám Python oznámi detaily udalosti, napr. súradnice, kde sme klikli (sú to atribúty parametra event.x a event.y).

V ďalšom príklade ukážeme, ako využijeme súradnice kliknutého bodu v ploche:

**import** **tkinter**

canvas = tkinter.Canvas(width=600, height=450, bg='white')

canvas.pack()

**def** klik(event):

x = event.x

y = event.y

canvas.create\_oval(x-10, y-10, x+10, y+10, fill='red')

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

Opäť sa pri kliknutí nakreslí červený kruh, len sa pritom využijú súradnice kliknutého miesta: stred kruhu je kliknuté miesto.

Akcia, ktorá sa vykoná pri kliknutí môže byť veľmi jednoduchá, napr. spájanie kliknutého bodu so stredom grafickej plochy:

**import** **tkinter**

canvas = tkinter.Canvas(width=600, height=450, bg='white')

canvas.pack()

**def** klik(event):

canvas.create\_line(300, 225, event.x, event.y)

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

Ale môžu sa nakresliť aj komplexnejšie kresby, napr. 10 sústredných farebných kruhov:

**import** **tkinter**, **random**

canvas = tkinter.Canvas(width=600, height=450, bg='white')

canvas.pack()

**def** klik(event):

x = event.x

y = event.y

**for** r **in** range(50, 0, -5):

farba = '#{:06x}'.format(random.randrange(256\*\*3))

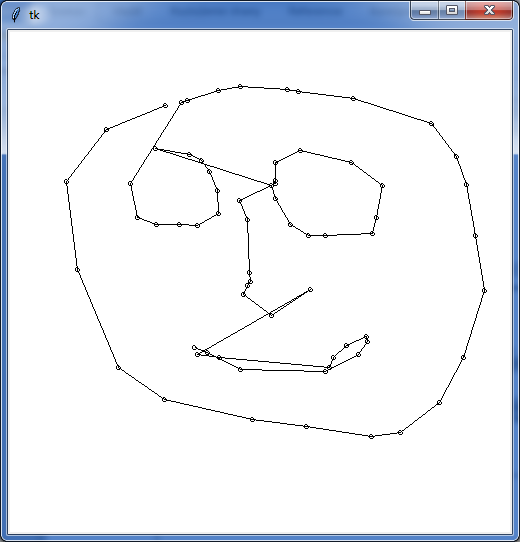
canvas.create\_oval(x-r, y-r, x+r, y+r, fill=farba)

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

Vytvorte program, ktorý bude kliknuté body spájať čiarami. V mieste kliknutia nakreslí malú kružnicu s polomerom 2.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

xx, yy = 0, 0

def klik(event):

global xx, yy

x, y = event.x, event.y

canvas.create\_oval(x-2, y-2, x+2, y+2)

if xx > 0:

canvas.create\_line(xx, yy, x, y)

xx, yy = x, y

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

V tomto prípade sú problémom globálne premenné – ak chcem kresliť úsečky musíme si pamätať kde sme predtým klikli.

Používať globálne premenné vo vnútri funkcii môžeme, len keď ich chceme vo vnútri funkcie meniť (použili sme priradenie xx, yy = x, y), Python pochopí, že obe tieto premenné sú lokálne (priradenie vo funkcii predsa vytvára novú premennú v **lokálnom mennom priestore**) a v príkaze if sme použili lokálnu premennú xx ešte skôr ako sme do nej niečo priradili. Takže s globálnymi premennými vo funkcii sa bude musieť pracovať nejako inak. Zrejme, keď nepotrebujeme do takejto premennej priraďovať, iba ju používať, problémy nie sú.

Tento problém nám pomôže vyriešiť nový príkaz global:

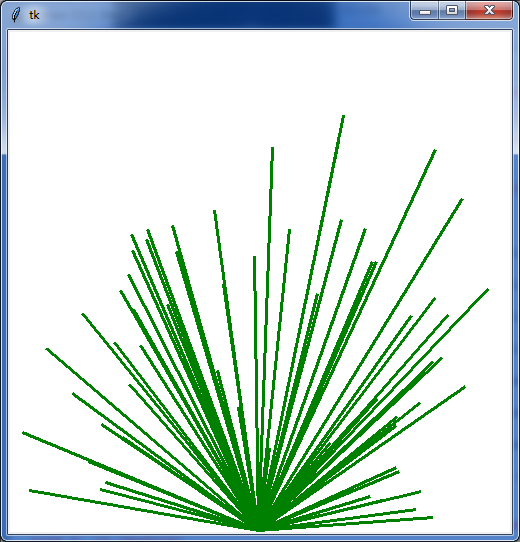
**príkaz global**

príkaz má tvar:

**global** premenná

**global** premenná, premenná, premenná, ...

Príkaz sa používa vo funkcii vtedy, keď v nej chceme pracovať s globálnou premennou (alebo aj s viac premennými) a Python ju nevytvorí v lokálnom mennom priestore, ale v globálnom.

Vytvorte program, ktorý bude kresliť hrubé zelené čiary vychádzajúce zo stredu v spodnej časti okna do miesta kliknutia.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

vyska = int(canvas['height'])

sirka = int(canvas['width'])

def klik(event):

x, y = event.x, event.y

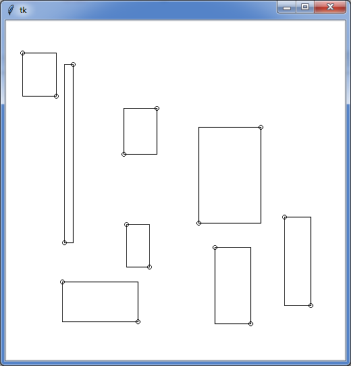
canvas.create\_line(sirka/2, vyska, x, y, fill='green', width=3)

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

Prvé kliknutie nakreslí bodku, druhé kliknutie nakreslí obdĺžnika medzi prvé a druhé kliknutie. Ďalej sa robí to isté pre 3. a 4. kliknutie, pre 5. a 6.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

pocet, xx, yy = 0, 0, 0

def klik(event):

global xx, yy, pocet

pocet += 1

x, y = event.x, event.y

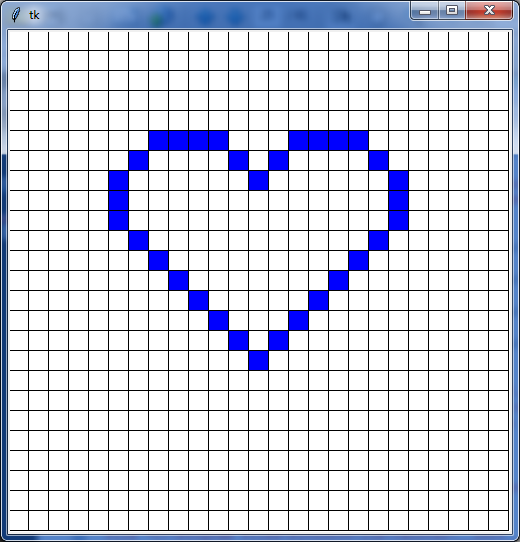
canvas.create\_oval(x-3, y-3, x+3, y+3)

if pocet%2 == 0:

canvas.create\_rectangle(xx, yy, x, y)

xx, yy = x, y

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

Vytvorte štvorčekovú sieť, program zistí do ktorého štvorčeka ste klikli a ten vyfarbí modrou farbou. Funkcia stvorec(r, s, farba) nakresli v r-tom riadku a v s-tom stĺpci štvorček zvolenej farby

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

#a-sirka stvorceka

a = 20

def stvorec(r, s, farba):

canvas.create\_rectangle(s\*a, r\*a, s\*a+a, r\*a+a, fill=farba)

#i-ty riadok

for i in range(0, vyska//a):

#j-ty stlpec

for j in range(0, sirka//a):

stvorec(i, j, 'white')

def klik(event):

#x-je stlpec

x = event.x

#y-je riadok

y = event.y

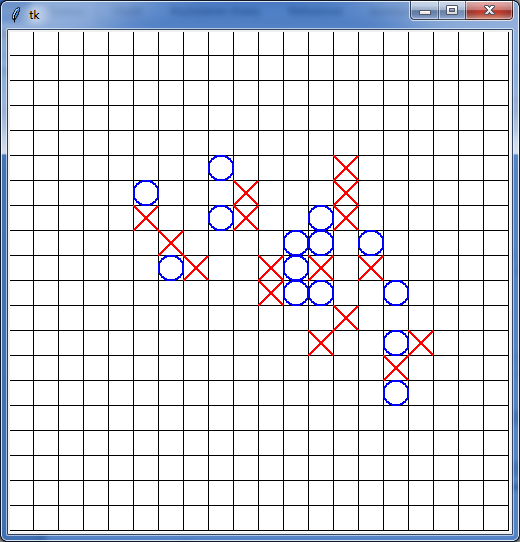
stvorec(y//a, x//a, 'blue')

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

Do štvorčekovej sieti kreslite ľavým tlačidlom modré krúžky a pravým tlačidlom červený krížik.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

a=25

sirka = int(canvas['width'])

vyska = int(canvas['height'])

def stvorec(r, s):

canvas.create\_rectangle(s\*a, r\*a, s\*a+a, r\*a+a)

for i in range(0, vyska//a):

for j in range(0, sirka//a):

stvorec(i, j)

def kruzok(r, s, farba):

canvas.create\_oval(s\*a, r\*a, s\*a+a, r\*a+a, outline=farba, width=2)

def krizik(r, s, farba):

canvas.create\_line(s\*a, r\*a, s\*a+a, r\*a+a, fill=farba, width=2)

canvas.create\_line(s\*a+a, r\*a, s\*a, r\*a+a, fill=farba, width=2)

def klikl(event):

x, y = event.x, event.y

kruzok(y//a, x//a, 'blue')

def klikr(event):

x, y = event.x, event.y

krizik(y//a, x//a, 'red')

#krizik(5, 5, 'red')

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

canvas.bind('<Button-3>', klikr)

## Ťahanie myšou

Obsluha udalosti ťahanie myšou (pohyb myši si zatlačeným tlačidlom) je veľmi podobné klikaniu. Udalosť ma meno '<B1-Motion>' – so zatlačeným ľavým tlačidlo, '<B3-Motion>' – so zatlačeným pravým tlačidlom. Pozrime, čo sa zmení, keď kliknutie '<Button-1>' nahradíme ťahaním '<B1-Motion>':

**import** **tkinter**

canvas = tkinter.Canvas(width=600, height=450, bg='white')

canvas.pack()

**def** klik(event):

canvas.create\_line(300, 225, event.x, event.y)

canvas.bind('<B1-Motion>', klik)

Funguje to veľmi dobre: priťahaní sa kreslia lúče zo stredu plochy k pozícii myši; pri pomalom ťahaní sú čiary kreslené veľmi nahusto. Často v našich programoch budeme spracovávať obe udalosti: kliknutie aj ťahanie. Niekedy je to tá istá funkcia, inokedy sú rôzne, napr.

**import** **tkinter**

canvas = tkinter.Canvas(width=600, height=450, bg='white')

canvas.pack()

**def** klik(event):

canvas.create\_line(300, 225, event.x, event.y, fill='red')

**def** tahanie(event):

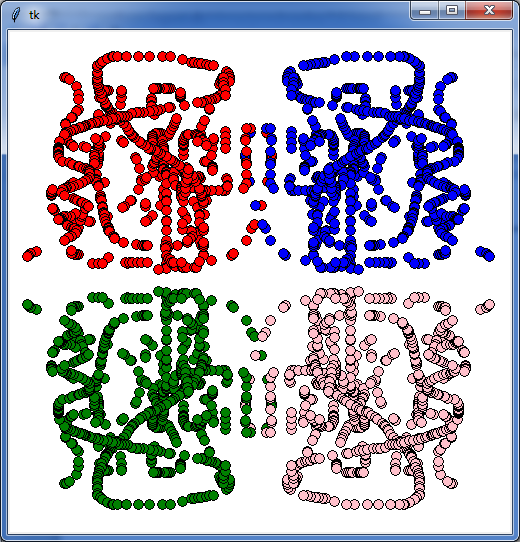
canvas.create\_line(300, 225, event.x, event.y)

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

canvas.bind('<B1-Motion>', tahanie)

Pri kliknutí (ešte bez ťahania) sa nakreslí červená úsečka, pri ťahaní sa kreslia čierne.

Pri kreslení útvarov môžeme využiť nielen súradnice myši, ale veľmi ľahko vieme vypočítať aj symetrické body: pre bod [x, y], sú to aj [sirka-x, y], [x, vyska-y], [sirka-x, vyska-y]. Vytvorte program, ktorý pri ťahaní myšou so zatlačeným ľavým tlačidlom bude kresliť farebné krúžky – každá symetrická časť kreslí krúžky inej farby.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

a = 5

def kruzok(event):

x, y = event.x, event.y

canvas.create\_oval(x-a, y-a, x+a, y+a, fill='red')

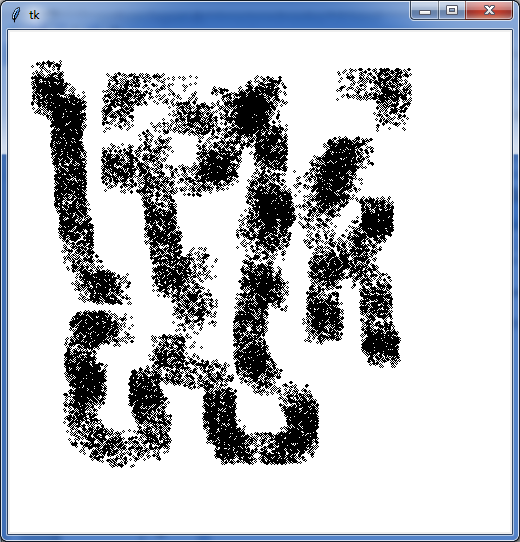
canvas.create\_oval(sirka-x-a, y-a, sirka-x+a, y+a, fill='blue')

canvas.create\_oval(x-a, vyska-y-a, x+a, vyska-y+a, fill='green')

canvas.create\_oval(sirka-x-a, vyska-y-a, sirka-x+a, vyska-y+a, fill='pink')

canvas.bind('<B1-Motion>', kruzok)

Vytvorte program, ktorý bude simulovať efekt kreslenia sprejom, nakreslí sa 50 náhodných bodiek v okolí pozície myši (okolie znamená, že x-ová aj y-ová súradnica nie je vo väčšej vzdialenosti ako napr. 15)

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

def sprej(event):

global xx, yy

x, y = event.x, event.y

for i in range(0, 51):

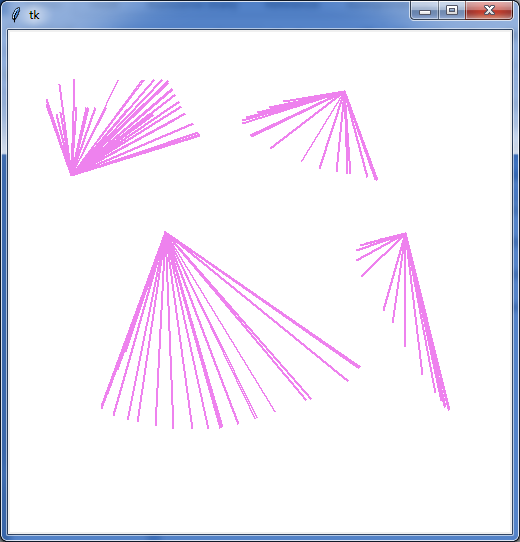
xs = random.randint(x-15, x+15)

ys = random.randint(y-15, y+15)

canvas.create\_oval(xs-1, ys-1, xs+1, ys+1)

canvas.bind('<B1-Motion>', sprej)

Nakreslite fialové vejáre. bod kde klikneme myšou si zapamätáme v globálnych súradniciach a pri ťahaní myšou budeme kresliť čiary, ktoré sú s týmto bodom spojené.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

xx, yy = 0, 0

def klik(event):

global xx, yy

x, y = event.x, event.y

xx, yy = x, y

def tahanie(event):

global xx, yy

x, y = event.x, event.y

canvas.create\_line(xx, yy, x, y, fill='violet', width=2)

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

canvas.bind('<B1-Motion>', tahanie)

**Reakcia na stláčanie klávesnice**

Na udalosti stlačenia klávesu reagujeme príkazom canvas.bind\_all('kláves', funkcia). Kláves je konkrétny znak na klávesnici, napr. 'a' alebo '**d**' alebo názov ovládacieho klávesu. Napríklad môžeme používať tieto klávesy:

'<Up>' - šípka hore

'<Down>' - šípka dole

'<Left>' - šípka vľavo

'<Right>' - šípka vpravo

'<space>' - medzera

Vytvorte program, ktorý bude zo stredu grafickej plochy kresliť úsečky v smere šípok po zatlačení na príslušný kláves z klávesnice.

import tkinter

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

xx, yy = sirka/2, vyska/2

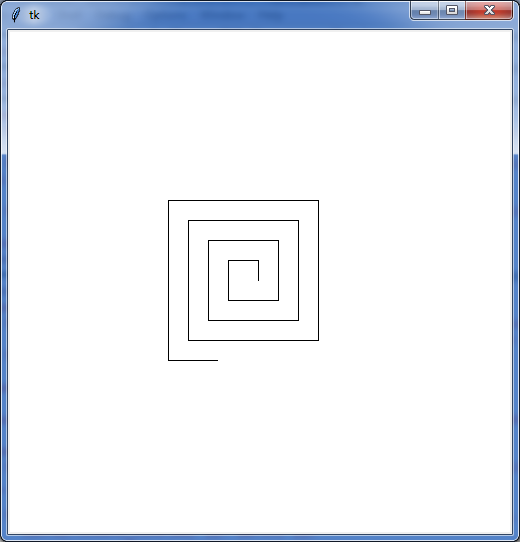
def hore(event):

global xx, yy

canvas.create\_line(xx, yy, xx, yy-10)

yy -= 10

def dole(event):

 global xx, yy

canvas.create\_line(xx, yy, xx, yy+10)

yy +=10

def vpravo(event):

global xx, yy

canvas.create\_line(xx, yy, xx+10, yy)

xx += 10

def vlavo(event):

global xx, yy

canvas.create\_line(xx, yy, xx-10, yy)

xx -= 10

canvas.bind\_all('<Up>', hore)

canvas.bind\_all('<Down>', dole)

canvas.bind\_all('<Right>', vpravo)

canvas.bind\_all('<Left>', vlavo)

**Opakovanie**

1. Napíšte procedúru stvorec(x, y, a, f), pričom parametre x, y určujú polohu štvorca, a je strana štvorca a f je farba štvorca.
2. Pomocou procedúry stvorec z predchádzajúceho zadania napíšte procedúru panelak(x, y, pp), pričom x, y určuje pozíciu paneláka pp počet poschodí.
3. Vytvorte Petržalské sídlisko na brehu Dunaja. V spodnej tretine canvasu nech sa vykreslí modrý obdĺžnik ako hladina Dunaja. Pri kliknutí zatlačeným ľavým tlačidlom myši kreslíte paneláky. Výška panelákov je od 1 do 13 poschodí. Pozor nesmú byť v Dunaji.
4. Napíšte skupinu príkazov, ktoré zabezpečia, že sa pri pohybe myšou nad grafickou plochou so stlačeným ľavým tlačidlom budú do grafickej plochy vykresľovať malé štvorčeky náhodnej farby, ktorá je vybraná z postupnosti farieb. Zabezpečte, aby sa štvorčeky vykresľovali len v Dunaji.

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=800, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

canvas.create\_rectangle(0, vyska/3\*2, sirka, vyska, fill='sky blue',

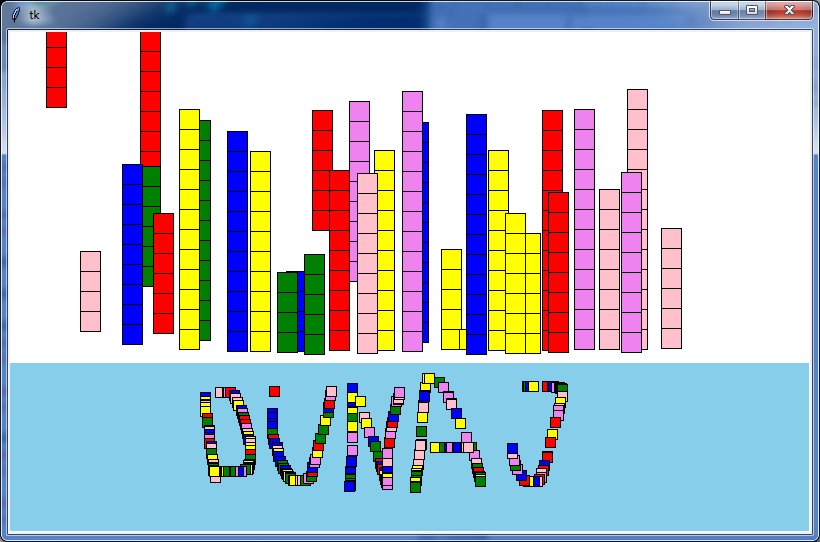
outline='sky blue')

def stvorec(x, y, a, f):

canvas.create\_rectangle(x, y, x+a, y-a, fill=f)

def panelak(x, y, pp):

a = 20

 f = random.choice(('green', 'blue', 'red', 'yellow', 'violet', 'pink'))

for i in range(1, pp+1):

stvorec(x, y, a, f)

y -= a

def klik(event):

x, y = event.x, event.y

pp = random.randint(1,13)

if y<vyska/3\*2:

panelak(x, y, pp)

def tahanie(event):

x, y = event.x, event.y

a = 10

#**pozor v choise 2 zatvorky**

f = random.choice**((**'green', 'blue', 'red', 'yellow', 'violet', 'pink'**))**

if vyska/3\*2<y<vyska:

stvorec(x, y, a, f)

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

canvas.bind('<B1-Motion>', tahanie)

**Podmienené príkazy – vysvetlenie**

Pri programovaní často riešime situácie, keď sa program má na základe nejakej podmienky rozhodnúť medzi viacerými možnosťami. Napr. program má vypísať, či zadaný počet bodov stačí na známku z predmetu. Preto si najprv vyžiada číslo - získaný počet bodov, **porovná** túto hodnotu s požadovanou hranicou, napr. 50 bodov a na základe toho vypíše, buď že je to dosť na známku, alebo nie je:

body = int(input('Zadaj získaný počet bodov: '))

**if** body >= 50:

**print**(body, 'bodov je dostačujúci počet na známku')

**else**:

**print**(body, 'bodov je málo na získanie známky')

Použili sme tu podmienený príkaz (príkaz vetvenia) if. Jeho zápis vyzerá takto:

**if** podmienka: *# ak podmienka platí, vykonaj 1. skupinu príkazov*

prikaz

prikaz

...

**else**: *# ak podmienka neplatí, vykonaj 2. skupinu príkazov*

prikaz

prikaz

...

V našom príklade sú v oboch skupinách príkazov len po jednom príkaze print(). Odsadenie skupiny príkazov (blok príkazov) má rovnaký význam ako vo for-cykle: budeme ich odsadzovať vždy presne o 4 medzery.

V pravidlách predmetu programovanie máme takéto kritériá na získanie známky:

* známka **A** za 90 a viac
* známka **B** za 80
* známka **C** za 70
* známka **D** za 60
* známka **E** za 50
* známka **Fx** za menej ako 50

Podmienka pre získanie známky **A**:

**if** body >= 90:

**print**('za', body, 'bodov získavaš známku A')

**else**:

...

Ak je bodov menej ako 90, už to môže byť len horšia známka: dopíšeme testovanie aj známky **B**:

**if** body >= 90:

**print**('za', body, 'bodov získavaš známku A')

**else**:

**if** body >= 80:

**print**('za', body, 'bodov získavaš známku B')

**else**:

...

Všetky riadky v druhej skupine príkazov (za else) musia byť odsadené o 4 medzery, preto napr. print(), ktorý vypisuje správu o známke **B** je odsunutý o 8 medzier. Podobným spôsobom zapíšeme všetky zvyšné podmienky:

body = int(input('Zadaj získaný počet bodov: '))

**if** body >= 90:

**print**('za', body, 'bodov získavaš známku A')

**else**:

**if** body >= 80:

**print**('za', body, 'bodov získavaš známku B')

**else**:

**if** body >= 70:

**print**('za', body, 'bodov získavaš známku C')

**else**:

**if** body >= 60:

**print**('za', body, 'bodov získavaš známku D')

**else**:

**if** body >= 50:

**print**('za', body, 'bodov získavaš známku E')

**else**:

**print**('za', body, 'bodov si nevyhovel a máš známku Fx')

Takéto odsadzovanie príkazov je v Pythone veľmi dôležité a musíme byť pritom veľmi presní. Príkaz if, ktorý sa nachádza vo vnútri niektorej vetvy iného if, sa nazýva **vnorený príkaz if**.

V pythone existuje konštrukcia, ktorá uľahčuje takúto 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 podmienka\_1 a podmienka\_2 neplatia, ale platí podmienka\_3, ...*

prikaz

...

**else**: *# ak žiadna z podmienok neplatí, ...*

prikaz

...

Predchádzajúci program môžeme zapísať aj takto:

body = int(input('Zadaj získaný počet bodov: '))

**if** body >= 90:

**print**('za', body, 'bodov získavaš známku A')

**elif** body >= 80:

**print**('za', body, 'bodov získavaš známku B')

**elif** body >= 70:

**print**('za', body, 'bodov získavaš známku C')

**elif** body >= 60:

**print**('za', body, 'bodov získavaš známku D')

**elif** body >= 50:

**print**('za', body, 'bodov získavaš známku E')

**else**:

**print**('za', body, 'bodov si nevyhovel a máš známku Fx')

Ukážme ešte jedno riešenie tejto úlohy - jednotlivé podmienky zapíšeme ako intervaly:

body = int(input('Zadaj získaný počet bodov: '))

**if** body >= 90:

**print**('za', body, 'bodov získavaš známku A')

**if** 80 <= body < 90:

**print**('za', body, 'bodov získavaš známku B')

**if** 70 <= body < 80:

**print**('za', body, 'bodov získavaš známku C')

**if** 60 <= body < 70:

**print**('za', body, 'bodov získavaš známku D')

**if** 50 <= body < 60:

**print**('za', body, 'bodov získavaš známku E')

**if** body < 50:

**print**('za', body, 'bodov si nevyhovel a máš známku Fx')

V tomto riešení využívame to, že else-vetva v príkaze if môže chýbať a teda pri neplatnej podmienke, sa nevykoná nič:

**if** podmienka: *# ak podmienka platí, vykonaj skupinu príkazov*

prikaz

prikaz

... *# ak podmienka neplatí, nevykonaj nič*

Zrejme každý if po kontrole podmienky (a prípadnom výpise správy) pokračuje na ďalšom príkaze, ktorý nasleduje za ním (a má rovnaké odsadenie ako if). Okrem toho vidíme, že teraz sú niektoré podmienky trochu zložitejšie, lebo testujeme, či sa hodnota nachádza v nejakom intervale. (podmienku 80 <= body < 90 sme mohli zapísať aj takto 90 > body >= 80)

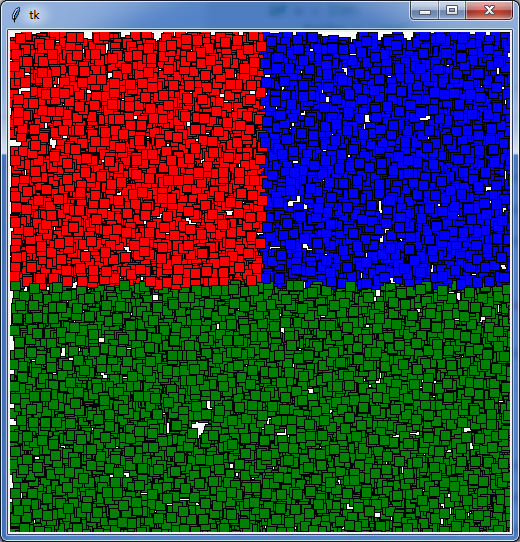
V Pythone môžeme zapisovať podmienky podobne, ako je to bežné v matematike:

|  |  |
| --- | --- |
| 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 |

Vytvorte program, ktorý na náhodné pozície nakreslí 1000 malých krúžkov, pričom tie, ktoré budú v spodnej polovici budú zelené a v hornej polovici vľavo budú červené a v pravej polovici budú modré.

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

def stvorec(x, y, farba):

canvas.create\_rectangle(x, y, x+10, y+10, fill=farba)

for i in range(10000):

x = random.randint(0, sirka)

y = random.randint(0, vyska)

if y < vyska/2:

if x < sirka/2:

farba = 'red'

else:

farba = 'blue'

else:

farba = 'green'

stvorec(x, y, farba)

Vytvorte program, ktorý zo stredu obrazovky nakreslí hrubú čiaru do miesta kliknutia myši. Čiaru kreslí farbou podľa miesta kliknutia. Obrazovka je rozdelená na štyri zóny. V ľavej hornej zóne kreslí červené čiary, v ľavej dolnej zóne modré čiary, v pravej hornej zóne zelené čiary a v pravej dolnej žlté čiary.

**Poznámka:**

Podmienky v Pythone môžu obsahovať logické operácie - majú obvyklý význam z matematiky:

* podmienka1 and podmienka2 ... (a súčasne) musia platiť obe podmienky
* podmienka1 or podmienka2 ... (alebo) musí platiť aspoň jedna z podmienok
* not podmienka ... (neplatí) podmienka neplatí

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=500, height=500)

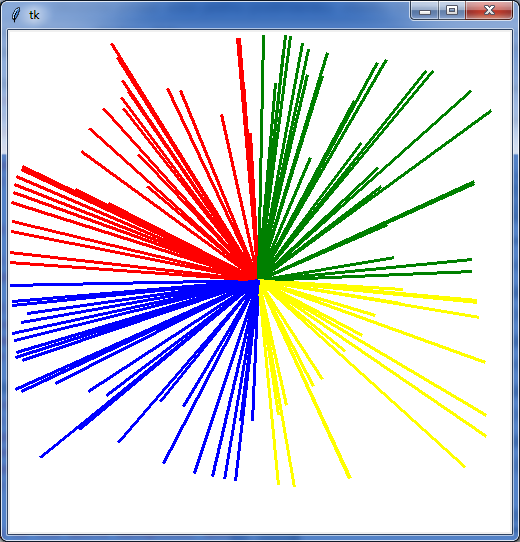
canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

def ciara(x, y, farba):

canvas.create\_line(sirka/2, vyska/2, x, y, fill=farba, width=3)



def klik(event):

x, y = event.x, event.y

if x < sirka/2 and y < vyska/2:

farba = 'red'

if x > sirka/2 and y < vyska/2:

farba = 'green'

if x < sirka/2 and y > vyska/2:

farba = 'blue'

if x > sirka/2 and y > vyska/2:

farba = 'yellow'

ciara(x, y, farba)

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

**Časovač**

V grafickej ploche môžeme kresliť krúžky na náhodné pozície s nejakým časovým pozdržaním (napr. 100 ms).

Metóda grafickej plochy after(), ktorá pozdrží výpočet o nejaký počet milisekúnd, je oveľa všestrannejšia: môžeme pomocou nej štartovať, tzv. **časovač**:

**metóda after()**

Metóda grafickej plochy, teda canvas, môže mať jeden z týchto tvarov:

canvas.after(milisekundy)

canvas.after(milisekundy, funkcia)

Prvý parameter milisekundy už poznáme: výpočet sa pozdrží o príslušný počet milisekúnd. Lenže, ak je metóda zavolaná aj s druhým parametrom funkcia, výpočet sa naozaj nepozdrží, ale pozdrží sa vyvolanie zadanej funkcie (skutočným parametrom musí byť referencia na funkciu, teda väčšinou bez okrúhlych zátvoriek). Táto vyvolaná funkcia musí byť definovaná bez parametrov.

S týmto druhým parametrom metóda after() naplánuje (niekedy v budúcnosti) spustenie nejakej funkcie a pritom výpočet pokračuje normálne ďalej na ďalšom príkaze za after() (bez pozdržania).

Tomuto mechanizmu hovoríme **časovač** (naplánovanie spustenia nejakej akcie), po anglicky **timer**. Najčastejšie sa používa takto:

**def** casovac():

*# príkazy*

canvas.after(cas, casovac)

Funkcia teda naplánuje spustenie samej seba po nejakom čase. 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.

Na zastavenie časovača nemáme žiaden príkaz. Časovač môžeme zastaviť len tak, že on sám v svojom tele na konci nezavolá metódu canvas.after() a tým aj skončí.

Nejakú udalosť môžeme spustiť aj klikaním (pozri program preteky dvoch bosoriek)

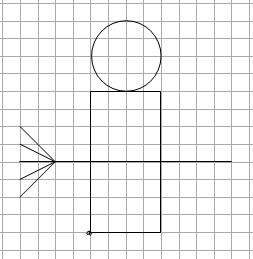
Bolo by dobre, keby sme vedeli počas behu časovača **zablokovať** klikanie a po skončení opäť povoliť. Využijeme metódu na zrušenie zviazania udalosti:

**metóda unbind()**

Metóda zruší zviazanie príslušnej udalosti:

canvas.unbind(meno\_udalosti)

**Preteky dvoch bosoriek – animácia**

Na začiatku sa vykreslia 2 bosorky, kliknutie tlačidlá spustí animáciu a na záver sa vypíše správa, ktorá bosorka vyhrala preteky. Znovu spustenie animácie spôsobí kliknutie do plochy.

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=800, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

a = 5

x1 = sirka/4

x2 = sirka/4\*3

def hlava(x, y):

canvas.create\_oval(x-2\*a, y-2\*a, x+2\*a, y+2\*a, fill='violet')

def telo(x, y, farba):

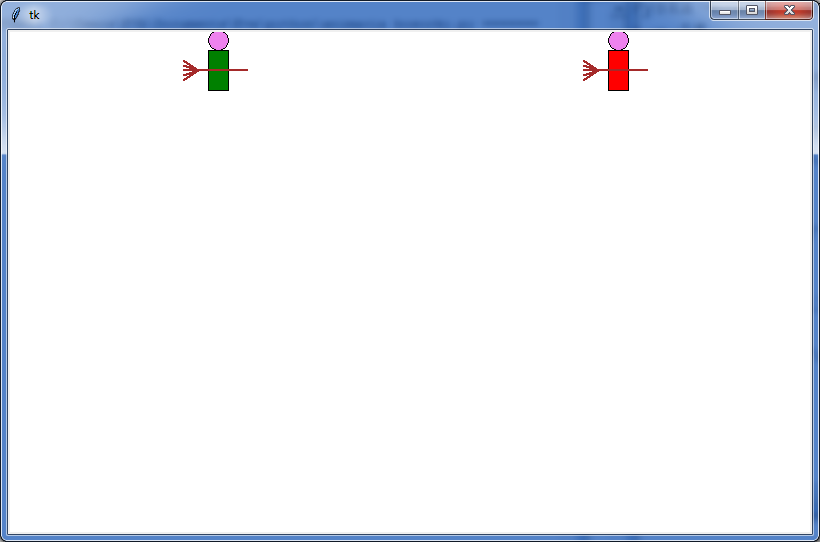
canvas.create\_rectangle(x, y, x+4\*a, y-8\*a, fill=farba)

def metla(x, y):

canvas.create\_line(x, y, x+10\*a, y, fill='brown', width=2)

for i in (2, 1, 0, -1, -2):

canvas.create\_line(x, y, x-3\*a, y-i\*a, fill='brown', width=2)

def bosorka(x, y, farba):

hlava(x+2\*a, y-10\*a)

telo(x, y, farba)

metla(x-2\*a, y-4\*a)

bosorka(x1, 60, 'green')

bosorka(x2, 60, 'red')

def start(event):

global y1, y2

y1, y2 = 60, 60

canvas.unbind('<Button-1>')

bosorka(x1, y1, 'green')

bosorka(x2, y2, 'red')

animacia()

def animacia():

global y1, y2

canvas.delete('all')

bosorka(x1, y1, 'green')

bosorka(x2, y2, 'red')

y1 += random.randint(1, 6)

y2 += random.randint(1, 6)

if y1 < vyska and y2 < vyska:

canvas.after(100, animacia)

else:

if y1 > y2:

canvas.create\_text(100,10, text='Vyhrala zelena bosorka',

font='Arial 10 bold', fill='green')

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

else:

canvas.create\_text(100,10, text='Vyhrala cervena bosorka',

font='Arial 10 bold', fill='red')

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

#animacia()

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

**Animácia kvapľožrúta**

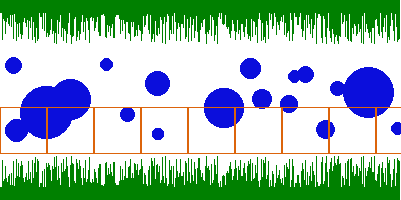
* **V tejto časti sa zopakuje for-cyklus**

1. Vytvorte program, ktorý do canvas-u nakreslí jaskyňu s kvapľami. Šírka každého kvapľa je jeden bod a dĺžka je náhodná, no minimálne 20 bodov a maximálne 69 bodov. Ako vidieť na obrázku, v našej jaskyni sú kvaple symetrické podľa vodorovnej osi, ktorá prechádza stredom jaskyne. Kvaple sa nachádzajú tesne pri hornom a dolnom okraji canvas-u a zaberajú šírku celého canvas-u.



* **V tejto časti sa zopakujú náhodné hodnoty, tlačidlá a vstup**

1. Doplňte do programu tlačidlo, ktoré nakreslí v jaskyni mláčky vody. Každá mláka má tvar kruhu, je nakreslená modrou farbou a polomer má v rozsahu od 10 do 50 bodov. Žiadna z mlák sa nemôže dotýkať kvapľov. Počet mlák zadávame počas behu programu z Entry. Aby sme mohli jaskyňou prechádzať, dokreslite do nej vodorovný rebrík. Rebrík je hnedý, široký 40 bodov a počet stupienkov je 10. Rebrík prechádza celým canvas-om.



Súradnice stredu mláky musia padnúť do vyšrafovanej oblasti:

Kvaple

Kvaple

r

r

r

r

* **V tejto časti sa zopakuje timer – časovač, podmienené príkazy**

1. Jaskyniari objavili v jaskyni kvapľožrúta, ktorý požiera kvaple. V programe sa, ktoré spustí animácia pohybu kvapľožrúta spustí kliknutím myši do canvas-u. Pohybuje sa z ľavého dolného okraja canvas-u do pravého dolného okraja. Kvapľožrút vymazáva kvaple, po ktorých prešiel.
2. ***Domáca úloha doplňte príkazy tak, aby ak žrút zožerie spodné kvaple, začal žrať horné zľava doprava***

import tkinter, random

#tento zapis vytvori okno aplikacie a do premennej okno ulozi referenciu nan

okno = tkinter.Tk()

canvas = tkinter.Canvas(bg='white', width=600, height=300)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

xk = 0

yk = vyska - 33

dx = 10

for i in range(sirka):

y = random.randint(20, 69)

canvas.create\_line(i, 0, i, y, fill='green')

canvas.create\_line(i, vyska, i, vyska-y, fill='green')

#xz, yz -suradnice stredu hlavy

def kvaplozrut(xz, yz, farba):

rz = 33

ro = 5

#hlava

canvas.create\_oval(xz-rz, yz-rz, xz+rz, yz+rz, fill=farba, outline=farba)

#oci

canvas.create\_oval(xz-4\*ro, yz-2\*ro, xz-2\*ro, yz+ro, fill='green')

canvas.create\_oval(xz+2\*ro, yz-2\*ro, xz+4\*ro, yz+ro, fill='green')

#usta

canvas.create\_text(xz, yz+5, text='(', angle=90, fill='red')

def animacia():

global xk, yk, dx

#premazavanie kvaplozruta

kvaplozrut(xk,yk, 'white')

xk += dx

kvaplozrut(xk, yk, 'yellow')

if xk-33 < sirka:

canvas.after(100, animacia)

if xk-30 >= sirka:

dx = -dx

yk = 33

def start(event):

global xk, yk

canvas.unbind('<Button-1>')

kvaplozrut(xk, yk, 'yellow')

animacia()

def tlacidlo1():

pocet = int(vstup.get())

canvas.create\_rectangle(0, 69, sirka, vyska-69, fill='white', outline='white')

id = canvas.create\_rectangle(0, 69, sirka, vyska-69, fill='white', outline='white')

canvas.delete(id)

for i in range(pocet):

r = random.randint(10, 50)

x = random.randint(r, sirka-r)

y = random.randint(r+69, vyska-r-69)

canvas.create\_oval(x-r, y-r, x+r, y+r, fill='navy', outline='navy')

canvas.create\_line(0, vyska-69, sirka, vyska-69, fill='brown', width=3)

canvas.create\_line(0, vyska-69-40, sirka, vyska-69-40, fill='brown', width=3)

stupen = sirka/10

x = 0

for i in range(10):

canvas.create\_line(x, vyska-69, x, vyska-69-40, fill='brown', width=3)

x += stupen

tlacidlo1 = tkinter.Button(okno, text='Kresli mláky', command=tlacidlo1)

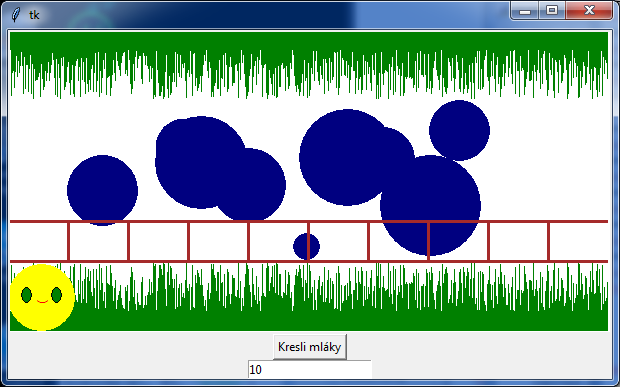
tlacidlo1.pack()

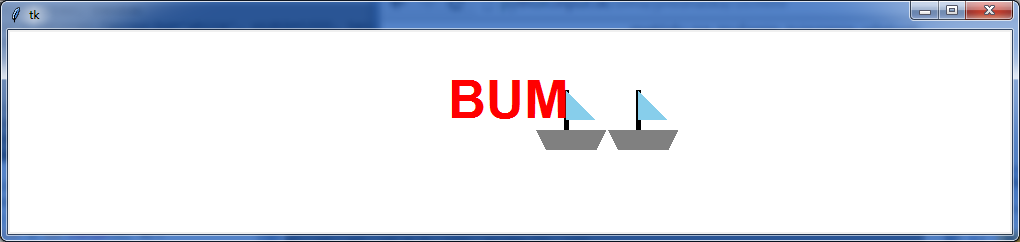
vstup = tkinter.StringVar()

entry = tkinter.Entry(okno, textvariable=vstup)

entry.pack()

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



**Simulácia pohybu dvoch lodičiek**

Vytvorte program, ktorý bude simulovať pohyb dvoch lodičiek oproti sebe. Pohyb sa začne kliknutím myši do plochy. Tesne pred zrazením lodičiek sa pohyb zastaví a vypíše sa BUM. Po kliknutí myšou sa pohyb lodičiek znova obnoví.

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=1000, height=200)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

x1, y1, x2, y2 = 0, 100, sirka-70, 100

def lodicka(x, y):

canvas.create\_polygon(x, y, x+10, y+20, x+60, y+20, x+70, y, fill='gray')

canvas.create\_line(x+30, y, x+30, y-40, width=5)

canvas.create\_polygon(x+30, y-10, x+60, y-10, x+30, y-40, fill='sky blue')

lodicka(x1, y1)

lodicka(x2, y2)

def start(event):

canvas.unbind('<Button-1>')

global x1, x2

x1 = 0

x2 = sirka - 70

lodicka(x1, y1)

lodicka(x2, y2)

animacia()

def animacia():

canvas.delete('all')

global x1, x2

lodicka(x1, y1)

lodicka(x2, y2)

x1 += 8

x2 -=5

if x1+70 < x2:

canvas.after(100, animacia)

else:

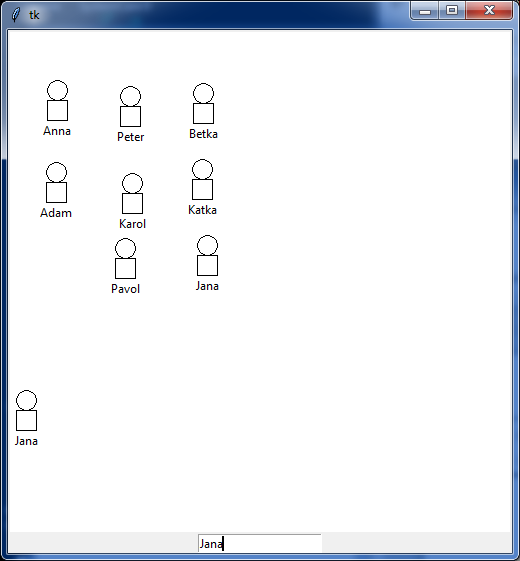
canvas.create\_text(sirka/2, 70, text='BUM', font='Arial 40 bold',

fill='red')

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

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

**Zasadací poriadok**

Vytvorte program – pomôcku pre učiteľa na vytvorenie zasadacieho poriadku. Do súčiastky Entry zadávame meno študenta, ktorého chceme usadiť. Kliknutím do plochy nakreslíme na mieste kliknutia toho študenta aj s menom.

import tkinter

okno = tkinter.Tk()

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

a = 20

def ziak(x, y):

canvas.create\_rectangle(x, y, x+a, y-a)

canvas.create\_oval(x, y-2\*a, x+a, y-a)

canvas.create\_text(x+a/2, y+a/2, text=vstup.get())

def klik(event):

x, y = event.x, event.y

ziak(x, y)

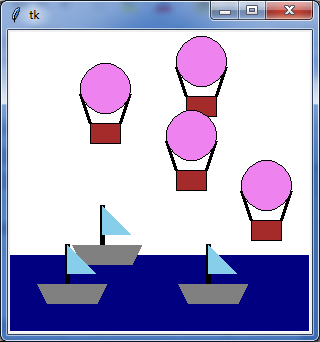
vstup = tkinter.StringVar()

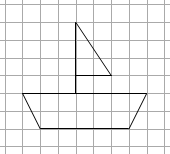
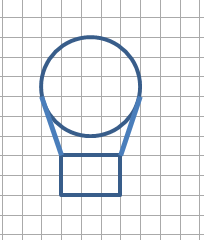
entry = tkinter.Entry(okno, textvariable=vstup)

entry.pack()

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

**Balóny a lodičky**

Vytvorte program, v ktorom si môžeme klikaním myši nakresliť lietajúce balóny na oblohe a plávajúce lodičky na mori. Program hneď v spodnej štvrtine nakreslí hladinu mora. Keď klikneme myšou nad morom, tak sa nakreslí balón. Ak klikneme na more tak, tak sa nakreslí lodička. Pri vytváraní funkcií môžete využiť obrázky.



import tkinter, random

canvas = tkinter.Canvas(bg='white', width=300, height=300)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

def lodicka(x, y):

canvas.create\_polygon(x, y, x+10, y+20, x+60, y+20, x+70, y, fill='gray')

canvas.create\_line(x+30, y, x+30, y-40, width=5)

canvas.create\_polygon(x+30, y-10, x+60, y-10, x+30, y-40, fill='sky blue')

def balon(x, y):

canvas.create\_rectangle(x, y, x+30, y-20, fill='brown')

canvas.create\_line(x, y-20, x-10, y-50, width=3)

canvas.create\_line(x+30, y-20, x+40, y-50, width=3)

canvas.create\_oval(x-10, y-80, x+40, y-30, fill='violet')

canvas.create\_rectangle(0, vyska/4\*3, sirka, vyska, fill='navy',

outline='navy')

def klik(event):

x, y = event.x, event.y

if y < vyska/4\*3-10:

balon(x, y)

else:

lodicka(x, y)

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

**Rovnaké kocky**

Vytvorte hru rovnaké kocky. V hre sa zobrazujú náhodné hodnoty dvoch hracích kociek (1 - 6). Hodnoty sa v pravidelných intervaloch menia. Úlohou hráča je zatlačiť tlačidlo „Rovnaké“, ak sú na oboch kockách rovnaké hodnoty. Ak stlačil tlačidlo v správnom okamihu, pripočítajú sa dva body. Keď sú hodnoty na kockách rôzne, hráč stratí jeden bod. V hornej časti obrazovky sa vypisuje aktuálny bodový stav.

import tkinter, random

okno = tkinter.Tk()

canvas = tkinter.Canvas(bg='white', width=800, height=400)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

body = 0

def animacia():

global cislo1, cislo2

canvas.delete('all')

cislo1 = random.randint(1, 6)

cislo2 = random.randint(1, 6)

canvas.create\_text(sirka/4, vyska/4, text=str(cislo1),

font='Arial 100 bold', fill='red')

canvas.create\_text(sirka/4\*3, vyska/4, text=str(cislo2),

font='Arial 100 bold', fill='red')

canvas.after(1000, animacia)

def porovnaj():

global cislo1, cislo2, rovnake, body

if cislo1 == cislo2:

body += 2

canvas.create\_text(100, 10, text='Získané body: '+

str(body), font='Arial, 10', fill='navy')

else:

body -= 1

canvas.create\_text(100, 10, text='Získané body: '+

str(body), font='Arial, 10', fill='navy')

animacia()

tlacidlo = tkinter.Button(okno, text='ROVNAKÉ', command=porovnaj)

tlacidlo.pack()

**Šetrič obrazovky1**

Najprv náhodne vygenerujte bod x, y a potom ho postupne spájajte s bodmi na spodnej hrane plochy, na hornej a bočných hrán plochy. Náhodne vygenerovaný bod sa neustále mení, nastavte aj rôznu farbu úsečkam, podľa toho s ktorými bodmi sú pospájané.

import tkinter, random

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

def animacia():

canvas.delete('all')

x = random.randint(0, sirka)

y = random.randint(0, vyska)

for i in range(0, sirka, 5):

canvas.create\_line(x, y, i, vyska, fill='green')

canvas.create\_line(x, y, i, 0, fill='violet')

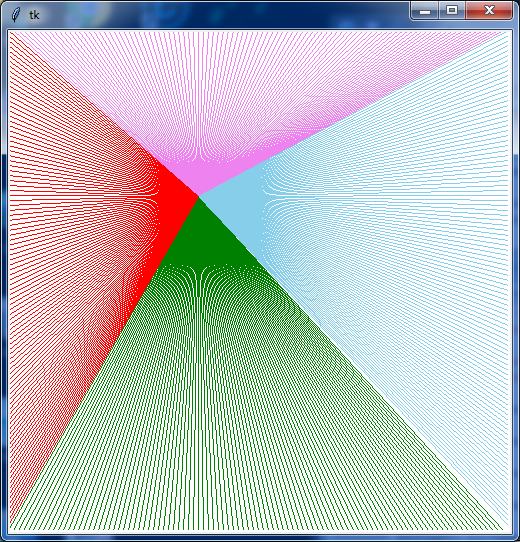
for i in range(0, vyska, 5):

canvas.create\_line(x, y, sirka, i, fill='sky blue')

canvas.create\_line(x, y, 0, i, fill='red')

canvas.after(1000, animacia)

animacia()



**Šetrič obrazovky2**

Vytvorte program – šetrič obrazovky, ktorý:

* nám umožňuje pomocou súčiastky zadávať ľubovoľný text,
* v pravidelných intervaloch zobrazuje zadaný text na náhodnom mieste a náhodnou farbou so zadaných možností,
* zadaný text v každom ďalšom zobrazení nakreslí otočený s väčším uhlom,
* text vykresľuje postupne iba s uhlami otočenia od 10 do 90 a potom znovu od začiatku s uhlom 10 až 90,
* pri kliknutí na ploch sa obsah obrazovky zmaže.

import tkinter, random

okno = tkinter.Tk()

canvas = tkinter.Canvas(bg='white', width=500, height=500)

canvas.pack()

uhol = 0

stop = 0

sirka = int(canvas['width'])

vyska = int(canvas['height'])

def start(event):

global stop, uhol

stop = 0

uhol = 0

canvas.unbind('<Button-1>')

#canvas.bind('<Button-3>', zmaz)

animacia()

def zmaz(event):

global stop

stop = 1

canvas.delete('all')

#canvas.unbind('<Button-3>')

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

def animacia():

global uhol

canvas.delete('all')

farba = random.choice(('navy', 'green', 'violet', 'red', 'yellow'))

canvas.create\_text(sirka/2, vyska/2, text=vstup.get(), angle=uhol, font=

'Arial 70 bold', fill=farba)

uhol += 10

if uhol > 90:

uhol = 0

if stop == 0:

canvas.after(1000, animacia)

vstup = tkinter.StringVar()

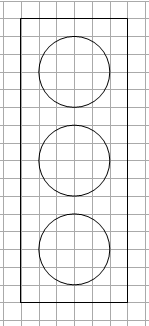
entry = tkinter.Entry(okno, textvariable=vstup)

entry.pack()

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

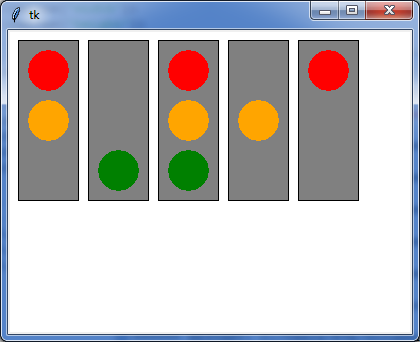
canvas.bind('<Button-3>', zmaz)

**Pokazený semafor**

Vytvorte program Pokazený semafor. Na križovatke sa pokazil semafor (obsahuje červenú, žltú a zelenú farbu). Teraz na ňom v pravidelných časových intervaloch svietia rôzne svetlá takto:

* niekedy svietia všetky tri svetlá,
* niekedy svieti len zelené alebo len žlté alebo len červené svetlo
* niekedy svieti naraz červené a žlté svetlo.

Ukážka všetkých možnosti je na obr.



import tkinter, random

canvas = tkinter.Canvas(bg='white', width=400, height=300)

canvas.pack()

sirka = int(canvas['width'])

vyska = int(canvas['height'])

pocet = 0

def kruh(x, y, farba):

canvas.create\_oval(x-20, y-20, x+20, y+20, fill=farba, outline=farba)

def semafor(x, y, f1, f2, f3):

canvas.create\_rectangle(x, y, x+60, y+160, fill='gray')

kruh(x+30, y+30, f1)

kruh(x+30, y+80, f2)

kruh(x+30, y+130, f3)

def animacia():

global pocet

if pocet%5==0:

pocet += 1

semafor(sirka/2-30, 10, 'red', 'orange', 'gray')

elif pocet%5==1:

pocet += 1

semafor(sirka/2-30, 10, 'gray', 'gray', 'green')

elif pocet%5==2:

pocet += 1

semafor(sirka/2-30, 10, 'red', 'orange', 'green')

elif pocet%5==3:

pocet += 1

semafor(sirka/2-30, 10, 'gray', 'orange', 'gray')

else:

pocet += 1

semafor(sirka/2-30, 10, 'red', 'gray', 'gray')

canvas.after(1000, animacia)

animacia()

Ďalšie štúdijné materiály:

<https://geo-inf.sk/tahak-prikazov/#t1>

<https://input.sk/python2018/03.html>