Základy programovania v Pythone

učebnica programovania v Pythone
pre stredné školy
a pre samoukov začiatočníkov

Pavol Daňo

Základy programovania v Pythone

učebnica programovania v Pythone pre stredné školy a pre samoukov začiatočníkov

Pavol Daňo

Základy programovania v Pythone

učebnica programovania v Pythone pre stredné školy a pre samoukov začiatočníkov

Autor:

Bc. Pavol Daňo

Recenzenti:

Mgr. Peter Pšenák, PhD.

Ing. Mgr. Peter Kuna, PhD.

Grafická úprava:

Bc. Pavol Daňo

Rukopis neprešiel jazykovou korektúrou.

Vydavateľské údaje:

ISBN 978-80-568-0598-5

Obsah

1 Úvodné slovo

Milí učitelia, milí žiaci,

dostáva sa vám do rúk učebnica, ktorá vám bude pomocníkom vo výučbe a osvojovaní si základov programovania v programovacom jazyku Python na hodinách informatiky a programovania na vašej strednej škole.

Hlavným cieľom tejto učebnice je pomôcť najmä školám, na ktorých sa výučbe programovania nevenuje až taká veľká pozornosť, akú si vyžaduje súčasnosť, vám, učiteľom informatiky, ktorí by ste chceli výučbu programovania zefektívniť alebo zaviesť a spopularizovať medzi vašimi žiakmi, no potrebujete „podať pomocnú ruku“ a vám, žiaci, ktorí možno aj uvažujete nad maturitou z informatiky alebo prácou programátora, no „bojíte“ sa toho.

Učebnica obsahuje viaceré tematické celky, na úvode ktorých sú vytýčené špecifické vzdelávacie ciele, ktoré sa snažia zahŕňať kognitívnu, afektívnu aj psychomotorickú oblasť – teda osvojenie si vedomostí, získanie kladného postoja a zručností využiť vedomosti pri riešení konkrétnych úloh. Za úvodom nasleduje stručne vysvetlená podstata tematického celku, podľa možností vždy na sérii ľahších aj ťažších úloh a príkladov. Na záver každej kapitoly nasleduje zhrnutie kľúčových častí učiva a otázky na zopakovanie.

Žiak posledného ročníka by mal úspešným absolvovaním a osvojením si všetkého učiva v rozsahu, v akom je prezentovaný v učebnici, dosiahnuť patričnú úroveň maturanta z programátorskej časti[1] predmetu informatika a získať pevné programátorské základy a algoritmické myslenie, ktoré uplatní nielen ako programátor v jazyku Python.

Niektoré tematické celky a časti učiva odporúčame popri inom učive vyučovať aj v nižších ročníkoch na informatike, a to aspoň tretinu vyučovacieho času,[2] aby žiaci získali aspoň nejakú predstavu o programovaní a aby im to pomohlo pri výbere povinne voliteľných predmetov do maturitného ročníka. Odporúčané tematické celky alebo ich časti pre prvý alebo druhý ročník sú označené hviezdičkou (*) pri nadpise. V maturitnom ročníku ich odporúčame potom preberať opäť – na zopakovanie.

Možno sa vám bude zdať učivo tejto učebnice, takpovediac, rozhádzané – veď prečo spomíname zásady programovania a algoritmizácie až kdesi v strede a nie na začiatku? Prečo rozoberáme typy výrazov až po cykloch a podmienkach? Napriek tomu odporúčame vyučovať a osvojovať si jednotlivé kapitoly tak, ako postupne nasledujú za sebou. Cieľom výučby programovania nie je a ani nemá byť zahrnutie žiaka informáciami o teórii programovania hneď v úvode, ale pristupovať induktívnou metódou a generalizovaním poznatkov, čím sa motivácia a záujem žiakov výrazne zvýši. Odporúčame tiež riadiť sa metodickými poznámkami pod čiarou, na ktoré hlavný text odkazuje horným číselným indexom.

Prajeme vám veľa motivácie pri výučbe a učení sa programovania a príjemnú prácu s touto učebnicou.

Textové súbory k príkladom a úlohám, na ktoré sa odkazujeme v kapitole 11 a multimediálne súbory k hre Logik v kapitole 13 sa spolu s elektronickou verziou tejto učebnice nachádzajú na nasledujúcich odkazoch:

- **odkaz 1:** https://pdfweb.truni.sk/e-ucebnice/python/
- **odkaz 2:** https://drive.google.com/drive/folders/1zvB4K3wNaxZq4lVZz2lOB8I_f-UoX8l7?usp=sharing

V prípade akýchkoľvek otázok k učebnici kontaktujte autora prostredníctvom e-mailovej adresy:

pavoldano321@gmail.com

[1] Odporúčanie platí pre gymnáziá – maturitná skúška z predmetu informatika, podobne ako obsahová náplň informatiky na gymnáziách, by mala pozostávať okrem programovania aj z iných oblastí definovaných v katalógu cieľových požiadaviek na maturitnú skúšku. Tým sa však táto učebnica nezaoberá.

[2] Odporúčanie platí pre gymnáziá – ak vaša škola využíva aspoň jednu disponibilnú hodinu na informatiku, odporúčame programovanie zaviesť v tom ročníku, kde je časová dotácia 2 hodiny týždenne a informatiku vyučovať v dvojhodinovom bloku.

2 Programovanie*

Ciele

Cieľom tejto kapitoly, podobne ako prvej hodiny programovania, je okrem vytýčenia a oboznámenia žiakov s cieľmi predmetu, priebehom výučby a spôsobom priebežnej a záverečnej klasifikácie aj otvorenie diskusie, kde žiaci vyjadria svoj názor na programovanie, ako ho vnímajú, kde a či sa s ním stretli, skúmať ich postoje k programovaniu, či a prečo si myslia, že je programovanie dôležité a prečo má pred ním veľa ľudí rešpekt. Žiak sa oboznámi s inštaláciou programovacieho prostredia IDLE Python, otváraním, ukladaním, spúšťaním a zastavovaním programu a tiež so základnými nastaveniami a režimami programovacieho prostredia.

2.1 Čo je to programovanie? Prečo je v súčasnosti tak veľmi žiadané? Prečo sa ho ľudia boja?

Programovaním možno označiť celý proces od zadania nejakej úlohy programátorovi, analýzy úlohy (s pomocou odborníka v danej oblasti, ktorý nemusí byť programátor) až po zostavenie algoritmu (t. j. konkrétneho postupu) a jeho realizáciu prostredníctvom ľubovoľného programovacieho jazyka, teda **vytvorenie** takého **postupu (t. j. programu), ktorý je schopný počítač vykonať, a tým automatizovane a samostatne riešiť predložený problém,** ďalej tiež jeho testovanie, vylepšovanie a aktualizáciu. Úlohou programátora je aj tvorba dokumentácie, údržba, poskytovanie licencie pre ďalšie zariadenia a podobne.

Nástupom nových technológií a neustálou modernizáciou elektroniky nastáva zavádzanie programovateľných čipov aj do zariadení, v ktorých sa ešte pred pár desaťročiami nenachádzali a v ktorých, takpovediac, vytlačili analógové mechanizmy (hodinky, chladničky, práčky, automobily, rozhlasové a televízne prijímače, kuchynské roboty), tiež nové smartfóny a iné inteligentné zariadenia obsahujú nespočetné množstvo aplikácií, teda enormne narástla potreba tieto zariadenia (hardvér) a aplikácie (softvér) vyrábať, vyvíjať – teda programovať a programátori sú tak čoraz viac „nedostatkovou komoditou".

A prečo sa ľudia programovania tak boja? V spoločnosti je stále vžitých veľmi veľa mýtov, že programátori sú introverti bez zmyslu pre humor, že sú to ľudia, ktorí musia mať vysokú školu, musia byť dobrí v matematike alebo to, že programovanie nie je nič pre ženy. Ľudia sa tiež boja neúspechu a myslia si, že program je spleť „nezmyselných" symbolov, ktoré programátorovi svietia zelenožltým písmom na tmavej obrazovke a myslia si snáď nebodaj aj to, že programátori celý program vedia naspamäť.

Pripomeňme si však, že prvá programátorka bola žena – grófka Ada Lovelace a programátori ako Bill Gates, Steve Jobs či Mark Zuckerberg nemajú dokončenú vysokú školu. Programovanie a informatika (resp. kybernetika) bola síce kedysi súčasťou matematiky, neznamená to však, že je nevyhnutné ovládať diferenciálny a integrálny počet či abstraktnú algebru, aby ste mohli byť programátorom. Takmer žiadny programátor nepracuje so strojovým kódom, ale so **zdrojovým kódom**[3] v programovacom prostredí, ktorý je prehľadný, intuitívny a pre človeka rýchlo osvojiteľný. Žiaden programátor neovláda celý programovací jazyk a ani to nie je potrebné – totiž – **nástrojom číslo jeden pre každého programátora je „googlenie".**[4]

2.2 Programovací jazyk Python

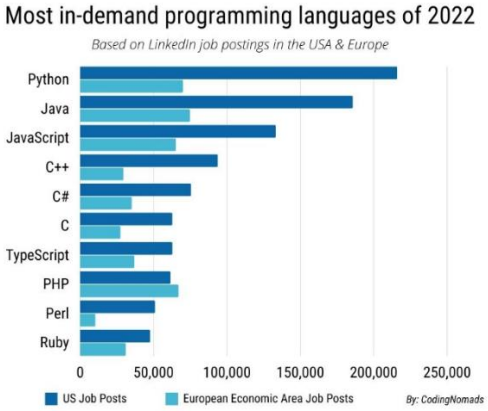
Prečo práve Python? Na mnohých školách, na ktorých sa programovanie vyučuje (možno aj na tej vašej), sa „stále" používa programovací jazyk Pascal a programuje sa v prostrediach Lazarus alebo Delphi. Tento jazyk bol vyvinutý najmä na edukačné účely, a to ešte v roku 1970, jeho modifikácie sa však neskôr ujali aj v programovaní bežných aplikácií. Stal sa z neho didakticky veľmi dobre premyslený programovací jazyk, ktorý výrazne zjednodušil syntaxe svojich predchodcov (ALGOL, COBOL), z ktorých vychádzal.

Mnohé školy na Slovensku či vo svete však postupne prechádzajú na jazyk Python, ktorý sa za pár rokov od jeho vzniku v roku 1989 stal azda najžiadanejším a najpoužívanejším programovacím jazykom v praxi

[3] *Vysvetlením, čo je to zdrojový a strojový kód, sa budeme zaoberať v kapitole venovanej algoritmizácii.*
[4] *Googlenie odporúčame dovoliť používať žiakom aj pri previerkach z programovania, aj na maturitnej skúške.*

a v programátorských firmách. Svoje aplikácie v ňom vytvárajú firmy ako Google, Facebook, Netflix, Spotify, Pinterest, Instagram a podobné. Tiež ho považujeme za oveľa vhodnejší na edukačné účely, **pretože hlavnou príčinou,** aj z vlastnej skúsenosti, **prečo si žiaci vytvoria odpor k programovaniu hneď na začiatku, je zložitá syntax jazyka** (napríklad aj jazyka Pascal).

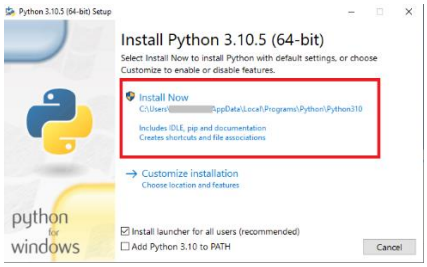
Python je navyše takzvane **multiplatformový,** čo znamená, že je kompatibilný s viacerými operačnými systémami počítačov, smartfónov a iných zariadení. Jazyk Python má síce mnoho kritikov, ktorí mu vyčítajú hlavne benevolenciu syntaxe a v edukácii žiakov prirovnávajú k učeniu sa jazdy na aute s automatickou prevodovkou v tom zmysle, že keď potom žiak nasadne do „staršieho" vozidla s manuálnou prevodovkou (začne sa učiť iný programovací jazyk – ťažší/s ťažšou syntaxou), bude si na to dlho zvykať. Napriek tomu považujeme zavedenie výučby programovania v Pythone za prínosné a potrebné – nielen spojiť školu s praxou a „pokročiť s dobou", ale **hlavne neodradiť žiakov od programovania** – na mnohých špičkových univerzitách sa tiež učí ako úvodný jazyk.[5]



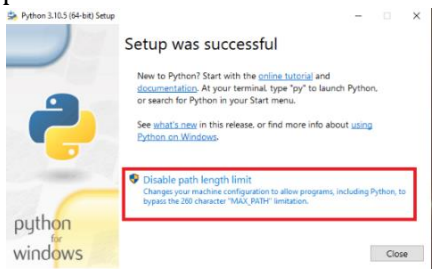
Graf najžiadanejších programovacích jazykov v Európe a USA [1].

2.3 Inštalujeme Python

1. Najnovšia verzia Pythona je vždy dostupná zadarmo na stránke https://www.python.org/downloads/.
2. Spustite inštalačný program. Odporúčame inštalovať Python s predvolenými funkciami/nastaveniami, t. j. **Install Now:**



3. Na záver odporúčame zrušiť limit cesty k súboru 260 znakov (napr. cesta súboru C:\Users\User\AppData\Local\Programs\Python310 má 46 znakov). Limity považujeme za zbytočné a akurát môžu viesť k neočakávanej chybe, keby sme mali napríklad súbor, s ktorým pracujeme, uložený vo viackrát vnorenom priečinku, s dlhším názvom a pod. Limit 260 znakov sa tak dá veľmi ľahko prekročiť.[6]



[5] *Napriek tomu odporúčame poukázať na hodinách programovania v maturitnom ročníku aj na príkazy a prvky, bez ktorých sa v iných jazykoch nezaobídeme (deklarácie premenných, oddeľovače príkazov, zátvorky a podobne), odporúčame však týmto nezaťažovať nižšie ročníky.*

[6] *Mali by sme však podotknúť, že tento limit je dôležitý z dôvodu kompatibility s rôznymi systémami. (Pred chvíľou sme tvrdili, že Python je multiplatformový a zrušením tejto voľby sa táto „multiplatformovosť", čiže kompatibilita, stráca...)*

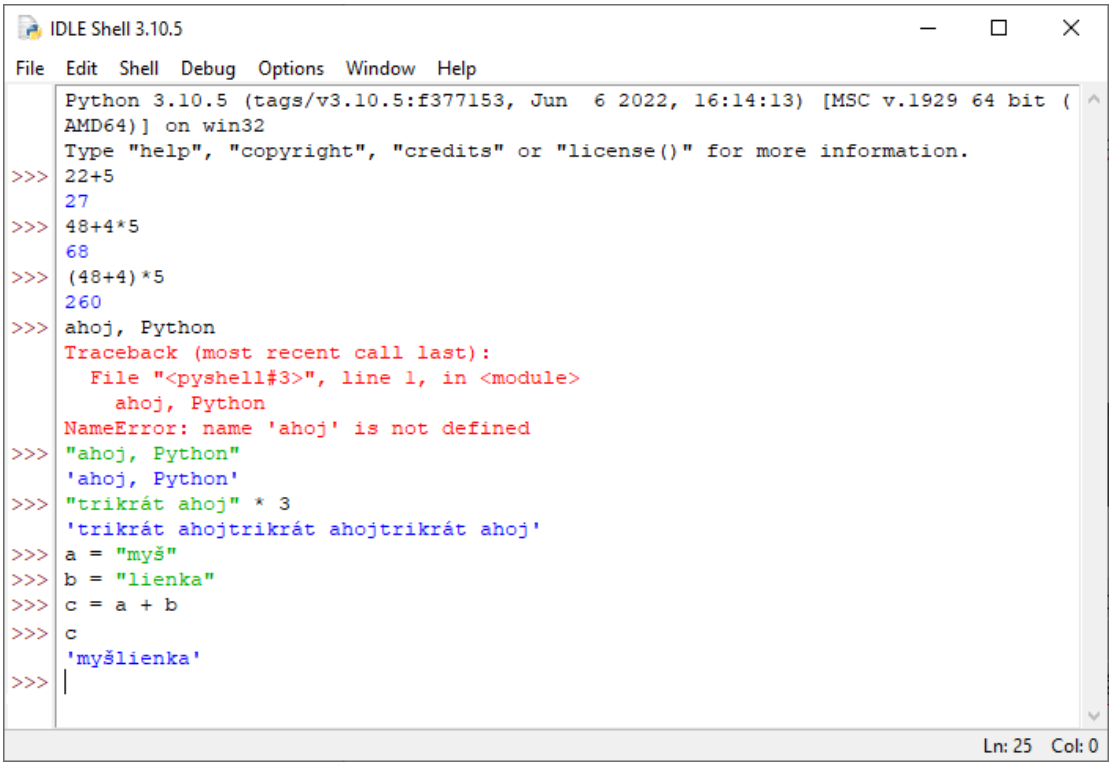
4. Po vypísaní hlásenia **Setup was successful** zatvoríme inštalačné okno.

2.4 Ovládame Python

Budeme pracovať vo **vývojovom prostredí jazyka Python,** tzv. **IDLE,**[7] teda vždy budeme spúšťať túto položku:[8]

Python pracuje **v niekoľkých režimoch.** Ten, ktorý sa otvorí kliknutím na uvedenú ikonu, sa volá **interaktívny režim.** Očakáva zadávanie textových príkazov do riadka za znaky **>>>** a 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,** prípadne **konzola.**[9]

Poďme si ho vyskúšať. Vidíme, že shell zvláda aritmetické operácie, dokonca ich vyhodnocuje podľa priority (najskôr násobenie, potom sčítanie). Keď však napíšeme text, vyhodnotí to ako chybu, pretože Python to berie ako názov premennej, ktorú ešte nepozná. Ak chceme pracovať s textom, musíme ho dať do **strojopisných**[10] úvodzoviek (`"text"`) alebo apostrofov[11] (`'text'`). Skúšali sme text uložiť do premennej a premenné sčítať, čo znamená v prípade textu spojiť.[12]

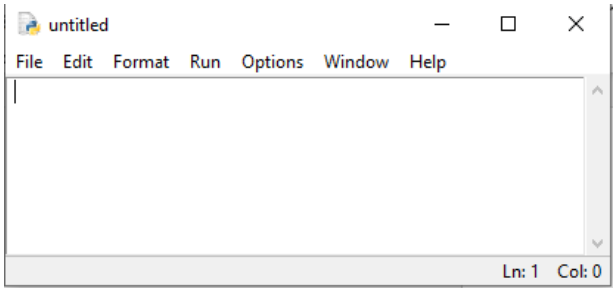


Programovanie ako také však nebude prebiehať v interaktívnom režime, ale v **programovacom režime.** Ten spustíme týmto spôsobom: **File → New File alebo Ctrl + N.** Otvorí sa nové okno, do ktorého budeme vpisovať program. Vyskúšať si môžete rôzne nastavenia, ktoré ale vôbec nie je potrebné vykonávať, hlavne odporúčame **nemeniť preddefinované farby textu kódu ani typ písma.**[13]

---

[7] IDLE = Integrated Development and Learning Environment, teda integrované vývojárske a výučbové prostredie.
[8] Samozrejme, verzia aj typ systému sa môžu u každého líšiť.
[9] Konzola je všeobecný termín; shell je v podstate názov konzoly iba niektorých programátorských prostredí, napr. IDLE Python.
[10] Strojopisné úvodzovky sú ", strojopisný apostrof ', pozor, v programovaní nepoužívame oblé „ “ úvodzovky ani oblý apostrof ‘.
[11] Apostrof napíšeme na slovenskej klávesnici stlačením klávesovej kombinácie Alt Gr + P (Alt Gr je pravý Alt; na niektorých klávesniciach je totiž označený iba ako Alt).
[12] Nateraz to bol však len test, k premenným a typom sa ešte vrátime a podrobnejšie ich rozoberieme.
[13] Ak ide o proporcionalitu – nevoľte si neproporcionálne písmo (také, ktoré nemá pevnú šírku znakov); Python je založený na odsadzovaní textu pod seba, čo použitím neproporcionálneho písma takmer nie je možné.

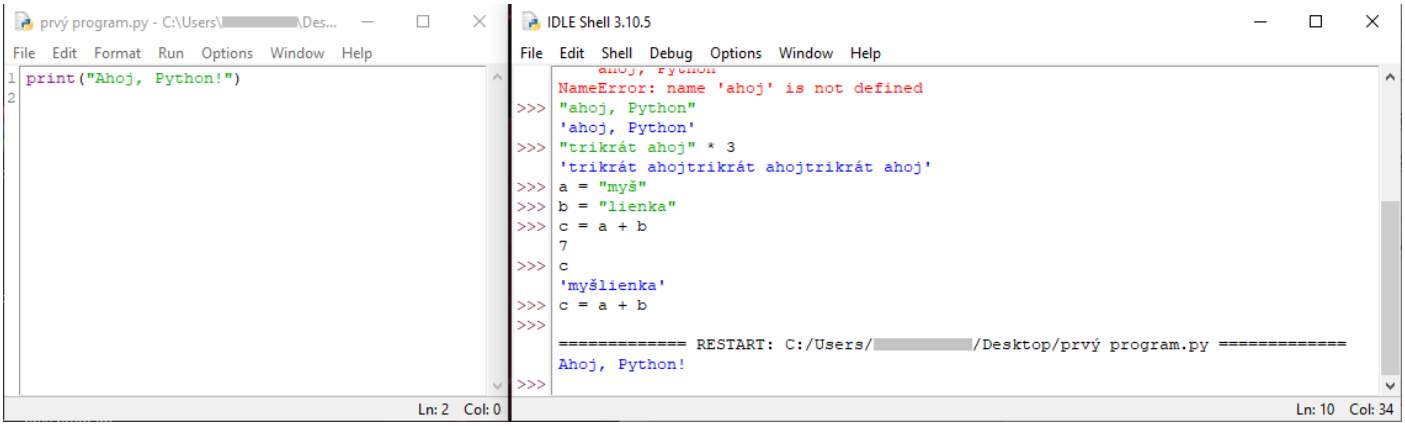
Čo odporúčame, pre toho, komu sú príjemnejšie inverzné farby a má takým štýlom nastavené aj farby operačného systému, je **nastavenie tmavého režimu:** Options → Configure IDLE → Highlights → IDLE Dark → Apply a rovnako **nastavenie zobrazovania čísel riadkov:** Options → Configure IDLE → Shell/Ed → Show line numbers in new windows → Apply.



Poďme si to vyskúšať. Napíšme jednoriadkový program, ktorý vypíše na konzolu (na shell) `Ahoj, Python!` Napíšeme takýto príkaz: `print("Ahoj, Python!")`. Vidíme, že slovo **print** aj text v úvodzovkách zmenilo svoju farbu. To, prečo to tak je a aj to, čo znamená `print` a prečo sa text, ktorý chceme vypísať, dáva do zátvoriek a ešte navyše aj do úvodzoviek, si vysvetlíme neskôr.

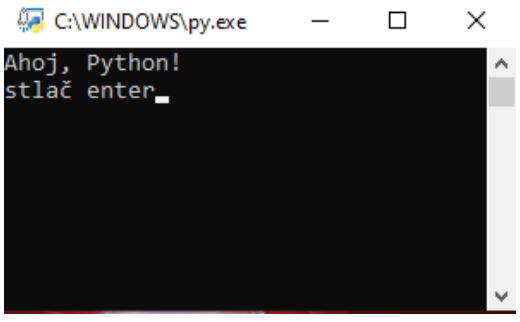
Projekt (program) si vhodne uložíme **(File → Save As...)** a spustíme **(Run → Run Module),** prípadne stlačením klávesu **F5.** Pamätajme na to, že **program musí byť vždy najskôr uložený a potom spustený.** Ak zabudneme, nič sa nedeje, informačné okno nás na to upozorní.

Po spustení program reštartuje shell, vykoná, čo má a ukončí svoju činnosť. To, že program ukončil svoju činnosť, vidíme na tom, že sa v shelli **vypíšu znaky >>> na posledný riadok:**

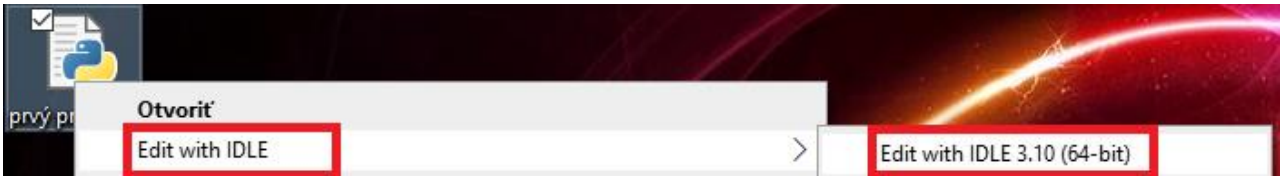


Ako opätovne otvoriť program? Povedali by ste, že nie je nič jednoduchšie ako otvoriť program v programovacom prostredí. Je to predsa tak, ako keď dvakrát klikneme na wordovský súbor a on sa vám otvorí v programe Word. Áno aj nie. Keď dvakrát poklikáme na ikonu programu **prvý program.py,** na obrazovke blikne akési čierne okno a vzápätí zmizne. Čo sa stalo? **Program sa spustil priamo zo systému, vykonal, čo mal** (vypísal text `Ahoj, Python!`) **a ukončil sa.**

Ak chceme, aby sa okno hneď nezatvorilo, je potrebné na koniec programu napísať `input("niečo")`. „Niečo“ môže byť napríklad `"stlač enter"`. Ono to vlastne nie je príkaz, ale špeciálna funkcia, podobne ako `print(...)`, k tomu sa však dostaneme neskôr. `input(...)` je funkcia, ktorá čaká dovtedy, kým nezadáme cez klávesnicu nejaký (požadovaný) text a nestlačíme **enter.**



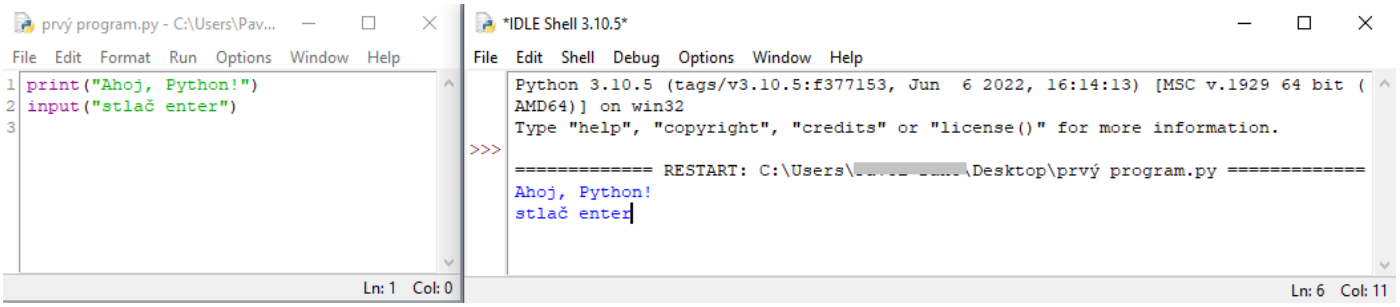
My však **chceme vidieť kód programu, ktorý sme uložili a zatvorili.** Urobíme to jedine tak, že na súbor klikneme pravým tlačidlom myši a vyberieme možnosť **Edit with IDLE → Edit with IDLE x.xx** (podľa toho, ktorú verziu máme nainštalovanú). Tým otvoríme program v programovacom režime.



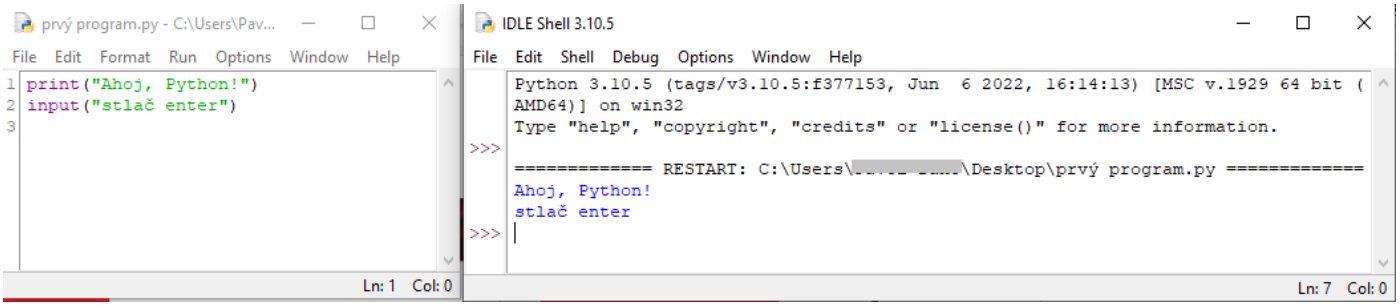
Shell sa otvorí **až vtedy, keď vykonáme spustenie programu** (cez **Run → Run Module** alebo **F5**) a môžeme, ale nemusíme ho po každom zastavení programu zatvoriť. Vo fáze písania a úpravy samotného kódu ho však nepotrebujeme.

Ak chceme **prerušiť činnosť** aktuálne spusteného programu, stlačíme kombináciu **Ctrl + C.**

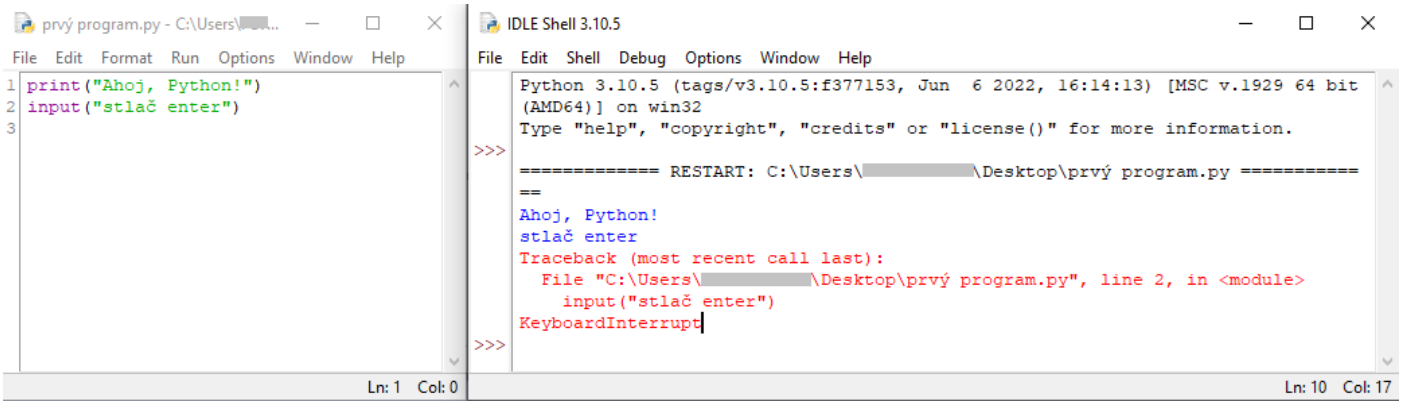
Spustený program (čaká na enter; na poslednom riadku v shelli nie sú znaky **>>>**):



Program ukončený (stlačením klávesu enter):



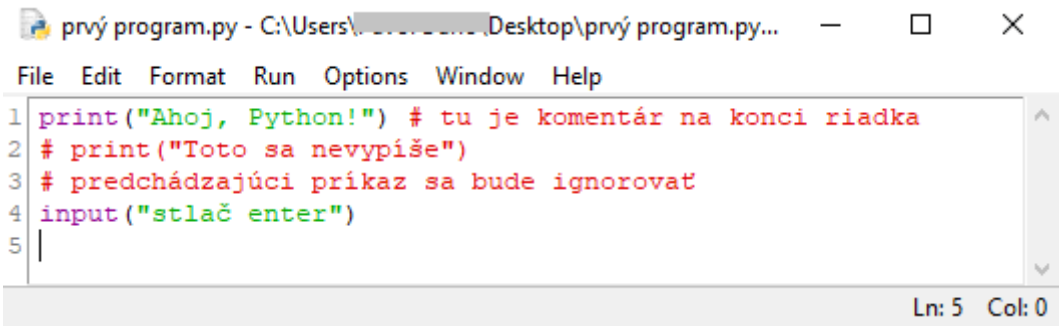
Program ukončený na riadku 2 stlačením **Ctrl + C** (`input(...)` nebol vykonaný = vykonávanie programu bolo prerušené):

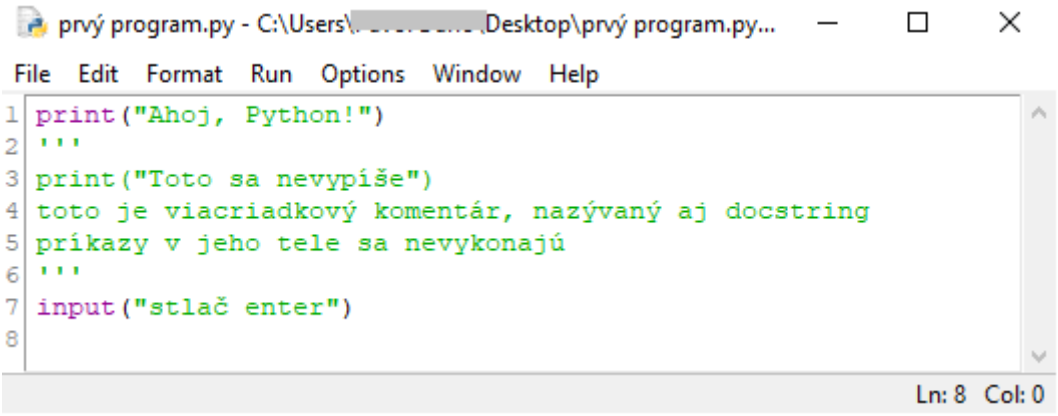


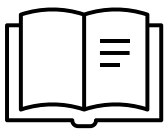
Ukážeme si teraz ešte jednu užitočnú vec – vkladanie komentárov.

Pri programovaní budeme veľmi často chcieť skúsiť napríklad napísať niečo inak a zachovať si pôvodný príkaz, ktorý nechceme vymazať, ale nechceme, aby sa vykonal. Prípadne si potrebujeme k niečomu priamo v programe spraviť poznámky či vysvetlenie. **Na to slúži symbol mriežky (#), ktorý napíšeme stlačením Alt Gr + X.** Symbol mriežky môžeme napísať na začiatku riadka, a tým „povieme“ programu, aby tento riadok „odignoroval“, prípadne hocikde v riadku – vždy platí, že **to, čo sa nachádza za #, sa už nevykoná a nazývame to**

komentár. Mnohé programovacie jazyky, vrátane jazyka Python, povoľujú aj viacriadkové komentáre, tzv. **docstring.** Tie sa v Pythone začínajú a aj končia trojicou strojopisných apostrofov `'''`.







ZHRNUTIE

- ✓ Programovanie je vytvorenie takého postupu, ktorý je schopný počítač vykonať, a tým automatizovane a samostatne riešiť predložený problém.
- ✓ Programovací jazyk, ktorý budeme používať, je Python.
- ✓ Programovať budeme v programovacom režime otvorením interaktívneho režimu IDLE a v ňom File → New File alebo Ctrl + N.
- ✓ Program sa ukladá cez File → Save As… a spúšťa cez Run → Run Module, prípadne stlačením klávesu F5.
- ✓ Program musí byť vždy najskôr uložený a potom spustený.
- ✓ Ukončený program je vtedy, keď sa v shelli vypíšu znaky `>>>` na posledný riadok.
- ✓ Program opätovne otvoríme tak, že na súbor klikneme pravým tlačidlom myši a vyberieme možnosť Edit with IDLE → Edit with IDLE x.xx.
- ✓ Činnosť aktuálne spusteného programu vieme prerušiť (teda ukončiť) stlačením kombinácie Ctrl + C.
- ✓ Na vkladanie jednoriadkových komentárov do programu používame symbol mriežky – `#`, viacriadkové komentáre sa začínajú aj končia znakmi `'''`.

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem programovanie.
2. Ako sa nazývajú dva programovacie režimy, ktoré poskytuje (napríklad) IDLE Python?
3. Ako správne uložíme program?
4. Ako otvoríme program s cieľom jeho úpravy?
5. Ako spustíme program?
6. Ako spoznáme, že program ukončil svoju činnosť?
7. Ako vieme ukončiť činnosť programu?

3 Kreslenie*

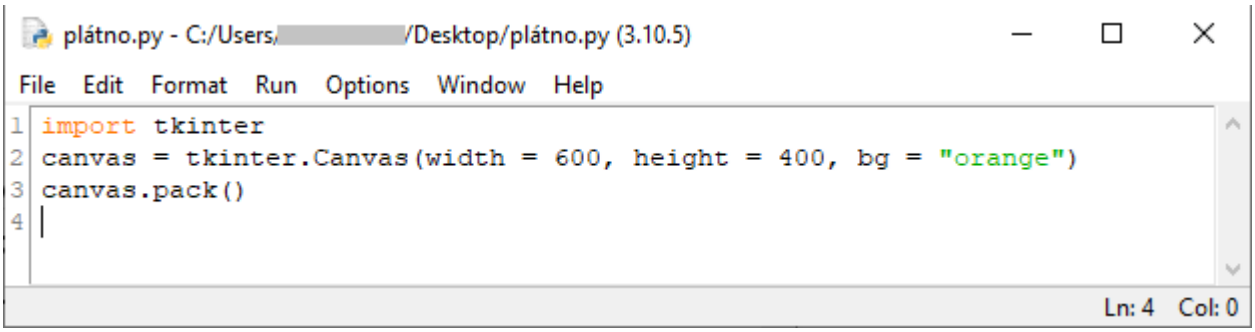
Ciele

*Cieľom tejto kapitoly má byť oboznámenie žiakov s tzv. grafickým rozhraním **TK inter** (tkinter), so súradnicovým systémom, ktorý toto rozhranie používa a tiež s kreslením základných geometrických útvarov, ktoré toto rozhranie podporuje, na plátno. Žiak bude po prebratí tejto kapitoly vedieť importovať rozhranie tkinter, nastaviť základné atribúty kresliaceho plátna, vedieť nakresliť na požadované miesta čiaru, lomenú čiaru, štvorec, kruh, elipsu, mnohouholník, kruhový výsek, text nastavením rôznych fontov, rezov a iných atribútov a vytvoriť jednoduchú animáciu.*

3.1 Rozhranie TK inter

Budeme pracovať s modulom – rozhraním **tkinter** (interface), ktorý slúži na vytváranie grafických aplikácií. My ho budeme využívať prakticky len na kreslenie na tzv. plátno, hoci tento modul zvláda aj iné typy grafických prvkov.

Základná séria príkazov potrebná na vykreslenie plátna, do ktorého budeme neskôr kresliť, je táto:



- `import tkinter` – importuje (tzn. zavolá, privedie, pripraví na používanie) rozhranie tkinter,[14]
- `canvas = tkinter.Canvas(width = 600, height = 400, bg = "orange")` – z balíčka `tkinter` vyberie funkciu `Canvas(…)`, ktorá vytvorí kresliace plátno, ktorému môžeme, ale nemusíme (odporúčame však robiť tak vždy) nastaviť rozmery a farbu pozadia – tu konkrétne výška (`width`) má 600 pixelov (obrazových bodov), šírka (`height`) 400 pixelov a `bg` (= background, čo je farba pozadia) je oranžová. Toto všetko je priradené do premennej `canvas`, pomocou ktorej budeme mať prístup k všetkým funkciám plátna.[15]
- `canvas.pack()` – umiestni plátno do grafickej aplikácie (do samostatného okna) – teraz je plátno pripravené, aby sme doň mohli kresliť.

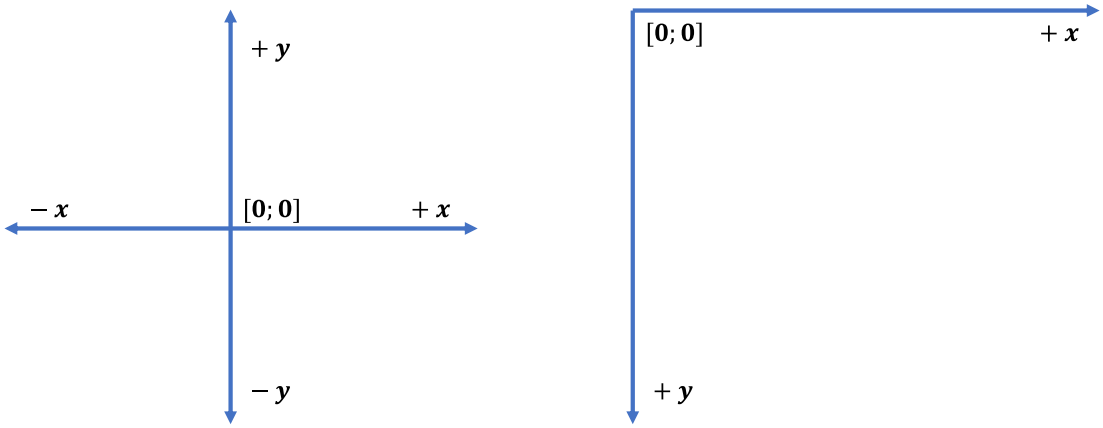
Za túto trojicu príkazov už budú nasledovať príkazy kreslenia tzv. geometrických primitív (čiara, kruh, elipsa, obdĺžnik, mnohouholník, text).

[14] *Je to súprava rôznych vopred naprogramovaných funkcií, ktoré spolu súvisia a sú uložené v balíčku. To, ako to približne funguje, si vysvetlíme neskôr (nematuritné ročníky sa tým však nemusia trápiť).*

[15] *Premenná sa mohla volať hocijako inak, avšak taká je „nepísaná dohoda“ medzi programátormi a bude sa s tým takto lepšie pracovať.*

3.2 Súradnicový systém

Z matematiky poznáme karteziánsky súradnicový systém, ktorého bod **[0; 0]** je v „pomyselnom strede“ v rovine. V počítačovej grafike sa však oveľa častejšie používa systém, **ktorého stred [0; 0] sa nachádza v ľavom hornom rohu obrazovky,** dialógového okna aplikácie alebo rôznych modulov, čo je aj tento prípad. Zľava doprava sa teda zvyšuje hodnota x, zhora nadol potom hodnota y. Záporné hodnoty súradníc sa väčšinou nepoužívajú a ak áno, znamená to, že sa bod nachádza mimo zobrazovanej plochy. Čísla súradníc sú vyjadrené v pixeloch (obrazových bodoch).

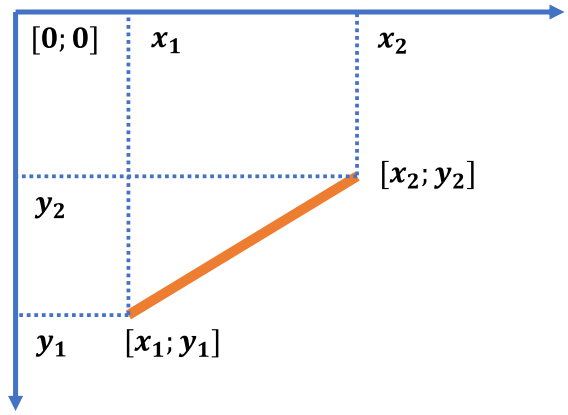


3.3 Geometrické primitíva

Ukážeme si príkazy na kreslenie geometrických primitív a to, ako sú zakotvené v systéme súradníc.

Čiara:

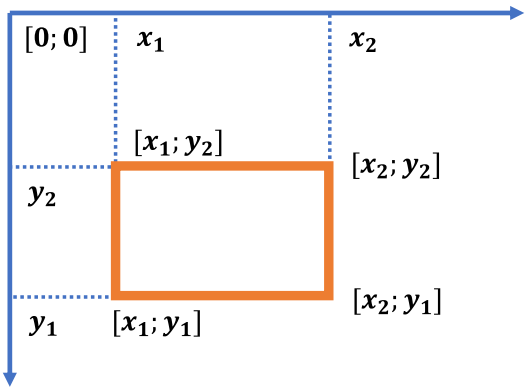
- `canvas.create_line(x1, y1, x2, y2, fill = "green", width = 5)`
- vyžaduje súradnice **koncových bodov** čiary, voliteľné parametre[16] sú výplň a hrúbka v pixeloch
- čiara môže mať aj viacero dvojíc súradníc (vrcholov), potom sa z nej stane **lomená čiara**
- parametre `fill` a `width` sú nepovinné, ak ich nenapíšeme, čiara (alebo hocijaký útvar nižšie spomenutý) má predvolene čiernu farbu, hrúbku 1 pixel a je bez výplne



Štvorec:

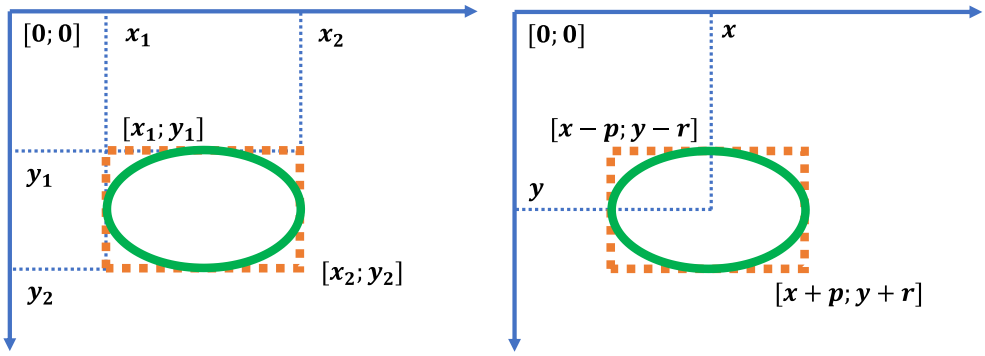
- `canvas.create_rectangle(x1, y1, x2, y2, outline = "green", fill = "red", width = 5)`
- vyžaduje súradnice koncových bodov jednej z uhlopriečok
- ten istý štvorec by sa teda vykreslil aj napísaním príkazu `canvas.create_rectangle(x1, y2, x2, y1, outline = "green", fill = "red", width = 5)`
- oproti čiare parameter `fill` znamená farba výplne útvaru a `outline` farba obrysu

[16] *Parameter je v programovaní označenie pre vstupné údaje podprogramu. Budeme sa tomu bližšie venovať v kapitole o podprogramoch.*



Elipsa:

- `canvas.create_oval(x1, y1, x2, y2, outline = "green", fill = "red", width = 5)`
- kreslí sa podobne ako obdĺžnik, s tým rozdielom, že elipsa, ktorá takto vznikne, je v podstate **vpísaná do „pomocného“ obdĺžnika**
- ak by sme chceli nakresliť elipsu alebo kruh so stredom v konkrétnom bode $[x; y]$, od prvých súradníc odpočítame a k druhým súradniciam pripočítame polomer
- podľa obrázka nižšie, ak $p = r$, dostaneme **kruh**, resp. kružnicu

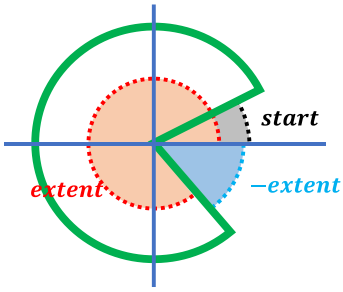


Mnohouholník:

- `canvas.create_polygon(x1, y1, x2, y2, x3, y3, xn, yn, outline = "green", fill = "red", width = 5)`
- už sme spomínali, že keď `line` môže mať viacero dvojíc súradníc, stane sa z nej lomená čiara, útvar `polygon`, na rozdiel od lomenej čiary, automaticky uzatvára tvar úsečkou z prvej a poslednej dvojice súradníc a umožňuje tiež takýto polygón vyplniť, dajú sa vytvárať aj nekonvexné polygóny

Kruhový výsek:

- `canvas.create_arc(x1, y1, x2, y2, outline = "green", fill = "red", width = 5, start = 0, extent = -50)`
- vykresľuje sa rovnakým spôsobom ako kruh, pričom parametre `start` a `extent` vyjadrujú v stupňoch, odkiaľ a pokiaľ sa má vytvoriť výsek (môžu byť aj záporné čísla) tak, ako v geometrii – proti smeru hodinových ručičiek (ako na obrázku):



Text:

- `canvas.create_text(x, y, text = "Python", font = "Times 45 bold")`
- vykreslí zadaný textový reťazec, pričom súradnice x a y určujú polohu stredu textu
- formátovanie textu (typ písma, veľkosť, rez) sa dá zapísať dvomi rôznymi spôsobmi:

```
font = ('Times New Roman', 45, 'bold italic')
font = 'Times 45 bold italic'
```

Problém však nastáva pri druhom spôsobe zápisu, ktorý síce rozpozná písmo podľa jeho prvého slova názvu `Times`, skúsme však napísať `font = 'Times New Roman 45 bold italic'` a program vypíše chybu. Mnohé písma s viacslovným názvom takto nevykreslíme (napr. Comic Sans MS; pri ňom nestačí napísať napr. `'Comic 45 bold italic'` a je potrebné použiť prvý spôsob s oddeľovaním parametrov čiarkou).

Textu sa dá, samozrejme, meniť napríklad aj farba, pridaním už známeho parametra `fill`, tiež sa s textom dá pohrať a vygoogliť si (už len zo zaujímavosti) ďalšie parametre, myslíme si však, že vyššie spomenuté bude bohato stačiť.

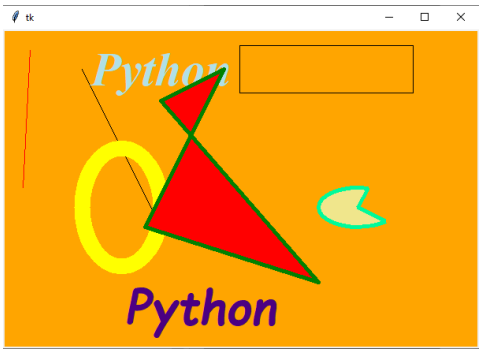
3.4 Farby

Skôr, než si v ďalšej kapitole vysvetlíme, ako a prečo sa dajú miešať farby, aké len chceme, základné farby, ktoré sú vopred pomenované a používa ich aj jazyk HTML a CSS, nájdeme napríklad tu: https://www.w3schools.com/colors/colors_names.asp [8]. Napriek tomu, že Python je tzv. case sensitive,[17] môžeme farby písať s malými alebo veľkými písmenami.

PRÍKLAD 1

Pokúste sa identifikovať, ktorý príkaz patrí ktorému útvaru a uvedomiť si, ako vyzerajú jednotlivé príkazy. Vezmite si ceruzku a papier, načrtnite si osi, naneste na ne hodnoty súradníc a na základe toho si načrtnite jednotlivé útvary, aby ste si to uvedomili a zvykli na systém súradníc.

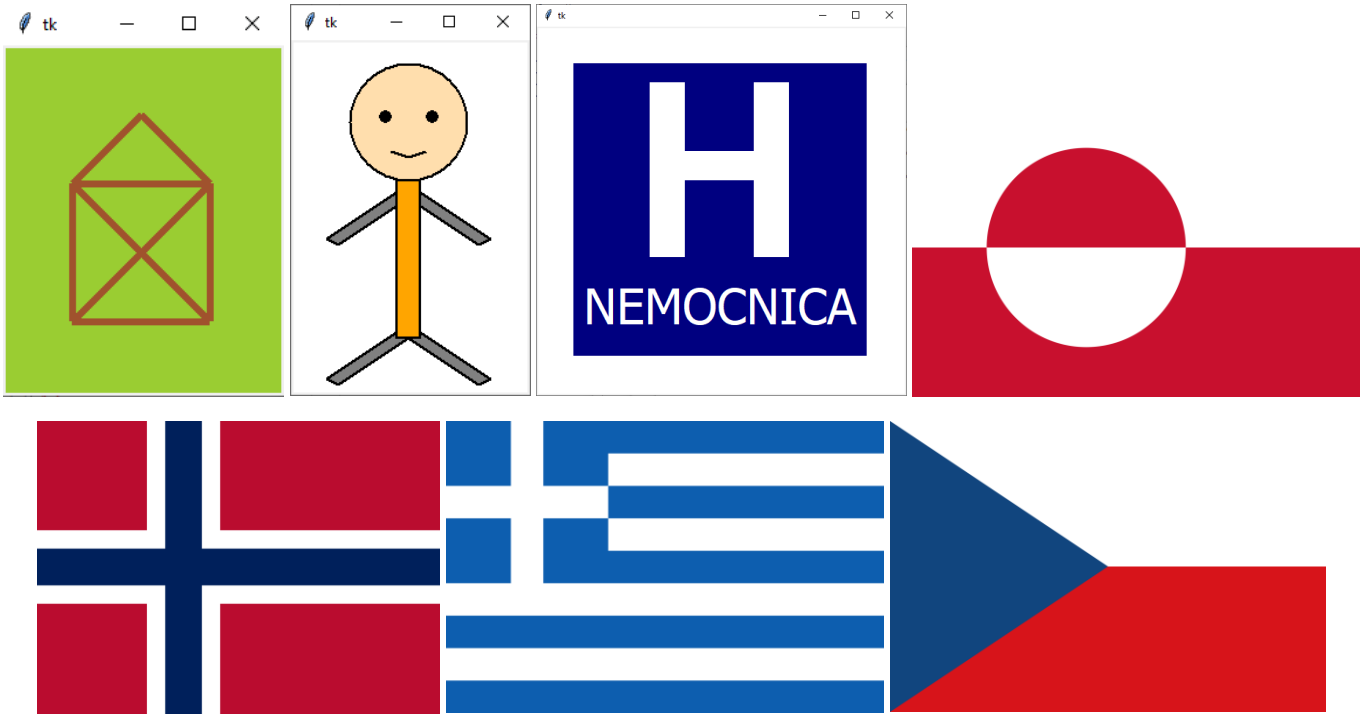
```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 600, height = 400, bg = "orange")
3.  canvas.pack()
4.
5.  canvas.create_line(100, 50, 200, 250)
6.
7.  canvas.create_text(250, 350, text = "Python", font = ('Comic Sans MS', 45, 'bold italic'), fill =
    "indigo")
8.
9.  canvas.create_rectangle(300, 20, 520, 80)
10.
11. canvas.create_oval(200, 150, 100, 300, outline = "yellow", width = 20)
12.
13. canvas.create_text(200, 50, text = "Python", font = 'Times 45 bold italic', fill = "PowderBlue")
14.
15. canvas.create_polygon(280, 50, 180, 250, 400, 320, 200, 90, fill="red", outline = "green", width = 5)
16.
17. canvas.create_line(25, 200, 34, 25, fill = "red")
18.
19. canvas.create_arc(400, 200, 500, 250, outline = "MediumSpringGreen", fill = "khaki", width = 5, start
    = 75, extent = 240)
```



[17] *Keď je niečo case sensitive, znamená to, že záleží na veľkosti písmen, a teda napr.* `Slovo ≠ slovo ≠ SLOVO`.

Úloha 1

Pokúste sa nakresliť panáčika, domček, informačnú tabuľu, vlajku Grónska a jednoduchšie vlajky iných krajín (a pohrajte sa aj s rozmermi plátna): [2]



3.5 Animácie

Aby bolo kreslenie zaujímavejšie, vyskúšame použiť funkcie **update** a **after.** Funkcia **`update()`** aktualizuje zmenu, ktorá sa na plátne udeje. Funkcia **`after(...)`** slúži na pozdržanie vykonávania príkazov, do jej zátvorky sa píše číslo, ktoré predstavuje, koľko milisekúnd má program čakať (1000 teda znamená jednu sekundu).

Príklad 2

Pokúste sa najskôr sami pomocou týchto príkazov nasimulovať animáciu kreslenia domčeka jedným ťahom.

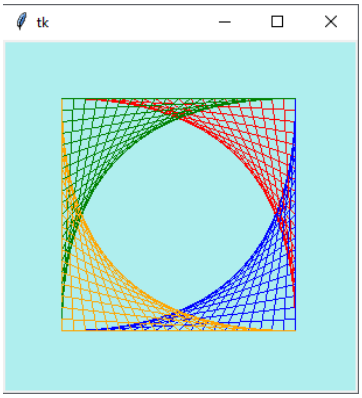
Ak ste vyskúšali, pozrime sa spoločne na to. Ak to máte podobne, spustite program a sledujte, čo robí.

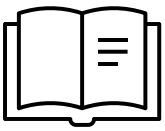
```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 200, height = 250, bg = "yellowgreen")
3.  canvas.pack()
4.
5.  # 1
6.  canvas.create_line(50, 200, 150, 100, fill = "sienna", width = 5)
7.  canvas.update()
8.  canvas.after(500)
9.
10. # 2
11. canvas.create_line(150, 100, 150, 200, fill = "sienna", width = 5)
12. canvas.update()
13. canvas.after(500)
14.
15. # 3
16. canvas.create_line(150, 200, 50, 100, fill = "sienna", width = 5)
17. canvas.update()
18. canvas.after(500)
19.
20. # 4
21. canvas.create_line(50, 100, 150, 100, fill = "sienna", width = 5)
22. canvas.update()
23. canvas.after(500)
```

```
24.
25. # 5
26. canvas.create_line(150, 100, 100, 50, fill = "sienna", width = 5)
27. canvas.update()
28. canvas.after(500)
29.
30. # 6
31. canvas.create_line(100, 50, 50, 100, fill = "sienna", width = 5)
32. canvas.update()
33. canvas.after(500)
34.
35. # 7
36. canvas.create_line(50, 100, 50, 200, fill = "sienna", width = 5)
37. canvas.update()
38. canvas.after(500)
39.
40. # 8
41. canvas.create_line(50, 200, 150, 200, fill = "sienna", width = 5)
42. canvas.update()
43. canvas.after(500)
```

Úloha 2

Skúste sa ešte „potrápiť“ (na základe toho, čo sme sa doteraz naučili) kreslením nasledujúceho obrazca. V ďalšej kapitole si ukážeme, že na to existuje iný, jednoduchší spôsob. Možno vám nejaký napadá už teraz (prosím, nepozerajte sa zatiaľ do nasledujúcich kapitol) [3].





Zhrnutie

- ✓ Python používa grafické rozhranie TK inter, v ktorom sa okrem iného dajú kresliť základné geometrické útvary, tzv. geometrické primitíva.
- ✓ Súradnicový systém je v počítačovej grafike so „stredom“ [0; 0] v ľavom hornom rohu obrazovky, hodnoty x-ovej osi stúpajú zľava doprava, hodnoty y-ovej osi zhora nadol.
- ✓ Z definície kreslenia geometrických primitív vyplýva, že vieme nakresliť štvorec (obdĺžnik) s uhlopriečkou a vpísanou kružnicou (elipsou) použitím tých istých súradníc.
- ✓ Mnohouholník je vlastne uzavretá lomená čiara.
- ✓ Funkcia **`update()`** aktualizuje zmenu, ktorá sa na plátne udeje a funkcia **`after(...)`** slúži na pozdržanie vykonávania príkazov.

Otázky na zopakovanie

1. Opíšte základnú sériu príkazov potrebných na vytvorenie kresliaceho plátna.
2. Charakterizujte rozdiel medzi karteziánskou súradnicovou sústavou a počítačovou (grafickou) sústavou.
3. Čo sa stane, ak neposkytneme geometrickým primitívom voliteľné parametre (napr. **`fill`**, **`width`**)?
4. Ako sa textu definuje jeho font (veľkosť, rez, typ písma)?
5. Ako vieme vytvoriť jednoduchú animáciu pri kreslení na plátne?

4 Premenná, priradenie, vstup a výstup, základné údajové typy

Ciele

Cieľom tejto kapitoly je, aby žiak vedel vysvetliť, čo je premenná, na čo sa používa, ako ju môže a ako ju nesmie pomenovať, čo je priraďovací príkaz, ako funguje funkcia **`print()`** *a* **`input()`** *(teda výstup a vstup), aby vedel správne používať priraďovací príkaz, pochopil princíp priraďovania a uvedomil si postupnosť príkazov. Žiak sa tiež oboznámi so základnými údajovými typmi – celé číslo, desatinné číslo, textový reťazec, vie s nimi vykonávať rôzne operácie a podľa potreby ich pretypovať.*

4.1 Premenná*

Doteraz sme sa len „hrali“ s „pevným“ kreslením na plátno. Programovanie sa však nezaobíde bez takej dôležitej veci, akou sú premenné, operácie s nimi, priraďovanie a modifikácia hodnôt premenných. Ukážeme si, ako sa pomocou premenných a funkcie náhodného generovania čísel dajú „oživiť“ veci, ktoré sme v predchádzajúcej kapitole kreslili na plátno. Vysvetlíme si tiež, ako sa dá sprehľadniť kód programu uložením funkcie so zložitým zápisom (syntaxou) do premennej.

Premenná slúži na **zapamätanie nejakej hodnoty tak, aby sme ju mohli použiť aj neskôr.**[18] Premenná[19] **pomenováva existujúcu hodnotu v pamäti.** Premenná, ak ešte predtým nevznikla, **vzniká priamo vykonaním priraďovacieho príkazu,**[20] **tzv. dynamickým priradením.**

Maturanti by mali vedieť, že existujú programovacie jazyky, ktoré sú orientované viac staticky ako dynamicky, a tam sa o premennej nedá tak celkom povedať, že slúži „len“ na zapamätanie hodnoty. Totiž, v jazykoch ako Pascal alebo C rozlišujeme medzi **deklaráciou a definíciou premennej.**

- **Deklarácia**[21] je zavedenie premennej („vyhlásenie“, že existuje) a jej reálna úloha je **pomenovanie konkrétneho vyhradeného** (tzv. alokovaného) **pamäťového miesta konkrétnej veľkosti a typu.**
- **Definícia** je už samotné **priradenie hodnoty k premennej**, teda jej uloženie do vopred vyhradeného – rezervovaného pamäťového miesta.

4.2 Priradenie hodnôt premennej*

Priraďovací príkaz v Pythone je napríklad: **`číslo = 85`**, kde:

- **`číslo`** je **identifikátor**[22] premennej,
- **`=`** je priraďovací znak,[23] pozor však, **nie je komutatívny!**[24]
- **`85`** je **hodnota** premennej,

a teda všeobecne: **`premenná = hodnota`**

Keďže priraďovací znak **nie je komutatívny,** najskôr na pravej strane:

- zistí, aká je tam hodnota,
- ak je tam operácia, vypočíta ju (zistí výslednú hodnotu),
- ak je tam operácia s vopred definovanou premennou alebo premennými, najprv zistí ich hodnotu a potom vykoná operáciu (zistí výslednú hodnotu)

a až potom tú hodnotu referuje (odkáže) na premennú z ľavej strany. Práve vďaka tomu môže hodnota obsahovať aj premennú (teda akýsi odkaz na inú hodnotu v pamäti), ale len takú, **ktorú sme vopred zadefinovali.**

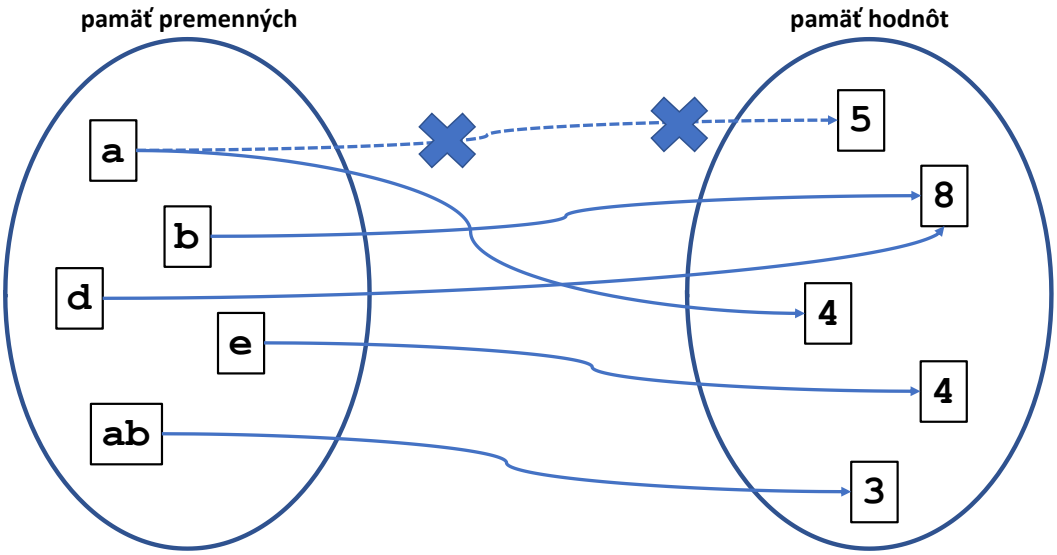
---

[18] *Premennú môžeme aj zmeniť a nielen použiť v stave, v akom práve je.*
[19] *V dynamicky orientovaných jazykoch, akým je aj Python.*
[20] *Ale iba v niektorých jazykoch. Vysvetlenie nájdete hneď v ďalšom odseku.*
[21] *z angl. declaration = vyhlásenie*
[22] *Niekedy identifikátoru premennej hovoríme aj názov alebo meno premennej.*
[23] *Pozor, nie je to znamienko rovnosti (to vyzerá inak)!*
[24] *komutatívnosť: ak $a = b$, tak aj $b = a$*

Ako to funguje v pamäti počítača? Mali by sme vedieť, že **spúšťanie každého programu** (aj operačného systému) **je vlastne akýsi proces načítavania potrebných častí kódu z pevného disku do operačnej** („pracovnej“) **pamäte**[25] **počítača.** Práve v operačnej pamäti sú vyhradené dve také miesta, ktorým hovoríme **pamäť identifikátorov a pamäť hodnôt,** ktorú si obhospodaruje program. Po vykonaní nasledujúcich príkazov:

```
1.  a = 5    # definícia premennej a (jej vytvorenie)
2.  b = 8    # definícia premennej b
3.  a = 4    # zmena hodnoty premennej a
4.  ab = 3   # definícia premennej ab, pozor, nie je to súčin a * b
5.  d = b    # definícia premennej d
6.  e = 4    # definícia premennej e
```

vyzerá pamäť premenných a pamäť hodnôt asi takto:



Poďme si to rozobrať:

- ako prvé sme priradili hodnotu **5** premennej **a**, v pamäti premenných tak vzniklo **a** a do pamäte hodnôt sa uložila hodnota **5**, premenná **a** teda odkazuje šípkou na hodnotu **5**
- ako druhé sme priradili hodnotu **8** premennej **b**, teda **b** z pamäte premenných odkazuje na **8** z pamäte hodnôt
- ďalej sme zmenili hodnotu premennej **a** na hodnotu **4**, **referencia** na hodnotu **5** teda **zanikla** a premenná **a** odteraz referuje na hodnotu **4** v pamäti hodnôt, hodnota **5** je teraz v pamäti „len tak“ odvisnutá a zanikne až vtedy, keby v operačnej pamäti nebolo žiadne miesto (prepíše sa inou hodnotou), alebo po zatvorení celého programu, alebo po reštarte počítača – záleží od operačného systému, procesora a podobne
- ďalej sme premennej **ab** priradili hodnotu **3**, princíp je už známy
- ďalej sme premennej **d** priradili hodnotu, na ktorú odkazuje premenná **b**, teda premenná **d** odkazuje odteraz **na tú istú hodnotu 8** z pamäte hodnôt **ako premenná b** – POZOR – to však neznamená, že ak neskôr zmeníme hodnotu jednej premennej, zmení sa automaticky aj hodnota druhej premennej[26]
- nakoniec sme premennej **e** priradili hodnotu **4**, prečo sa teda nereferovala na „štvorku“, ktorá už pamäti hodnôt existuje? Pretože sme sa **neodkázali na žiadnu hodnotu** (ako v prípade **d = b**, kde povieme, že priraď **d**-čku takú hodnotu, akú má **b**-čko), ale priradili sme premennej konkrétne číslo. Ak by sme tak chceli, museli by sme napísať **e = a**

Proces priraďovania hodnôt premennej je **veľmi dôležité si uvedomiť, bez neho sa v programovaní nikam nepohneme.**

[25] *V slovenčine rozlišujeme „ľudskú“ pamäť a pamäť počítača, napr.: dnešný deň sa mi uložil do **pamäti** (vzor kosť); ale: premennú **a** uložíme do operačnej **pamäte** (vzor dlaň).*

[26] *Toto však neplatí pri poliach. K tomu sa však dostaneme, zatiaľ to žiakom netreba vedieť, aby sa nemýlili.*

Úloha 3

Dokreslite do schémy, čo by sa stalo vykonaním nasledujúcich príkazov – pozor – zohľadnite to, že nasledujú za sebou!

To, aké hodnoty nakoniec premenné nadobudli, si môžete vyskúšať sami. Napíšte ich do programu, uložte ho, spustite ho. Nepoužívajte výpis (`print()`). Na shell sa z toho dôvodu síce nič nevypíše, program však svoju úlohu vykoná – jeho úlohou je vlastne iba vytvorenie premenných a priradenie hodnôt premenným. Program teda skončil s tým, že síce „nič viditeľné“ neurobil, ale **nechal po sebe v operačnej pamäti tieto hodnoty uložené.** Keď v shelli potom zadáte konkrétnu premennú a stlačíte enter, vypíše vám to jej hodnotu. Ak zadáte premennú, ktorá nebola vytvorená, teda v pamäti neexistuje, vypíše chybu. Hodnoty premenných môžete tiež v shelli priraďovacím príkazom meniť.

```
7.  e = e + 3
8.  ab = ab + ab
9.  a = b + 5
10. f = -5 + d
11. d = 8 + c
12. c = 14
```

4.3 Hromadné priradenie*

Spomeňme ešte zopár užitočných príkazov, ktoré fungujú v Pythone.

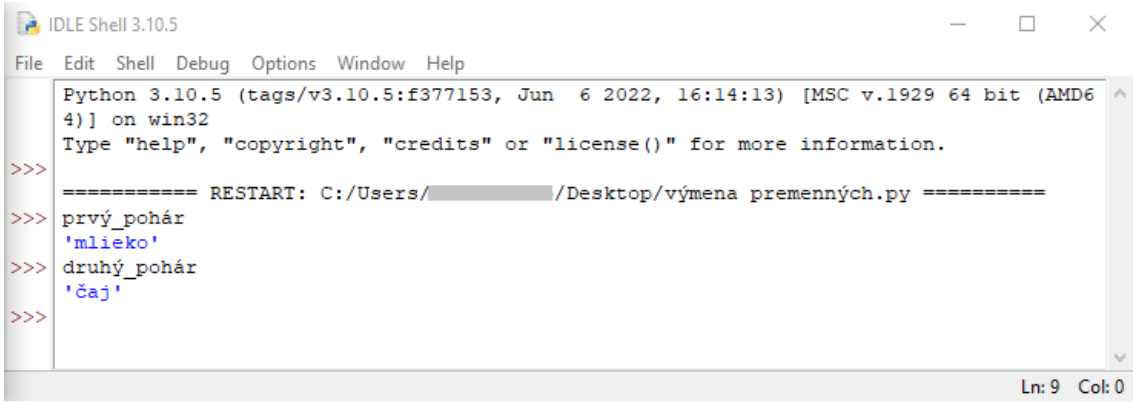
Príklad 3

Máme dva poháre, v prvom je čaj a v druhom mlieko. Ako prelejete čaj z prvého pohára do druhého a mlieko z druhého do prvého? Ak potrebujete, môžete použiť aj ďalší pomocný pohár.

Analogicky si to preveďme na premenné a pozrime sa na to:

```
1.  prvý_pohár = "čaj"
2.  druhý_pohár = "mlieko"
3.
4.  tretí_pohár = prvý_pohár
5.  prvý_pohár = druhý_pohár
6.  druhý_pohár = tretí_pohár
```

Čaj z prvého pohára sme preliali do pomocného – tretieho (v skutočnosti sme nič „nepreliali“, ale rovnakú hodnotu priradili ďalšej premennej, teda teraz máme v prvom a treťom pohári čaj). „Uvoľnil“ sa tým prvý pohár, do ktorého môžeme „preliať“ mlieko z druhého pohára (teda „premazať“ premennú (presnejšie – zmeniť jej referenciu) „mliekom“). Nakoniec z tretieho pohára „prelejeme“ čaj do druhého (čím druhému poháru zrušíme referenciu na mlieko a vytvoríme referenciu na čaj). Nakoniec klikneme do okna shell a zistíme (overíme si), čo sa nachádza v jednotlivých pohároch:



Ale Python ponúka jednoduchšie a elegantnejšie riešenie, ako „vymeniť obsah“ premenných a volá sa to **hromadné priradenie.** Vyzerá to takto:

```
1.  prvý_pohár = "čaj"
2.  druhý_pohár = "mlieko"
3.
4.  prvý_pohár, druhý_pohár = druhý_pohár, prvý_pohár
```

Princíp je jednoduchý – na pravej strane sa vytvorí **usporiadaná dvojica hodnôt,**[27] ktorá sa priradí usporiadanej dvojici premenných na ľavej strane. Rovnako sa to dá s tromi, štyrmi, …, *n* premennými naraz.

ÚLOHA 4

Určte, v ktorom pohári je aký nápoj po ich vzájomnej výmene

```
1.  prvý_pohár = "čaj"
2.  druhý_pohár = "mlieko"
3.  tretí_pohár = "voda"
4.  štvrtý_pohár = "džús"
5.
6.  tretí_pohár, prvý_pohár, druhý_pohár, štvrtý_pohár = druhý_pohár, prvý_pohár, štvrtý_pohár,
    tretí_pohár
```

Ďalšie zjednodušenie a sprehľadnenie zápisu sa dá vykonať aj vtedy, ak vytvárame viac premenných naraz:

```
1.  # toto:
2.  a = 5
3.  b = 8
4.  c = 9
5.  d = 1
6.  e = 0
7.  f = 4
8.  g = 4
9.
10. # je to isté ako toto:
11. a, b, c, d, e, f, g = 5, 8, 9, 1, 0, 4, 4
```

alebo ak vytvárame viac premenných s rovnakou hodnotou:

```
1.  # toto:
2.  a = 5
3.  b = 5
4.  c = 5
5.  d = 4
6.  e = 5
7.  f = 4
8.  g = 4
9.
10. # je "to isté" ako toto:
11. a = b = c = e = 5
12. d = f = g = 4
```

ÚLOHA 5

V druhom prípade je to „to isté“ len z hľadiska programátora. Pamäť premenných a/alebo pamäť hodnôt však bude zrejme vyzerať inak, keď použijeme jeden a keď použijeme druhý spôsob definície premenných. Skúste si to uvedomiť a načrtnite si to.

4.4 Identifikátor premennej*

Už sme si povedali a ukázali, ako funguje premenná a priradenie. Povieme si teraz však, aké **pravidlá** platia pri pomenovávaní premenných:

- identifikátor má každá premenná, je to **názov jej hodnoty,**[28]
- v Pythone môže obsahovať čísla, písmená, podčiarkovník (_) aj diakritiku,
- **nemôže sa začínať číslom,**

[27] *Dvojica, kde záleží na poradí členov, v Pythone známa ako* `tuple`*, k tomu sa však neskôr dostaneme.*

[28] *Dôležitá terminologická poznámka – nehovorme v súvislosti spomínania názvu premennej, že sa premenná volá, ale že sa nazýva. Volanie je v programovaní niečo úplne iné a budeme sa tým zaoberať neskôr.*

- identifikátormi premennej nemôžu byť tzv. **dohovorené slová** (napr. `for`, `if`, `in`), ale ani názvy funkcií – aj keď to už nie sú dohovorené slová, ale referencie na funkcie, napríklad `input` alebo `print`. K dohovoreným slovám Python ani nedovolí nič priradiť, pozor však, nepoužívajme názvy funkcií ako identifikátory, lebo keď niečo priradíme napríklad do `print` alebo `input`, zrušíme si funkciu! Ale môžeme si napr. zmeniť názov funkcie `input`, napr. `vstup = input`, potom už používame funkciu `vstup`, čo však tiež neodporúčame, je to tu len zo zaujímavosti v zmysle „aj to sa dá“:

```
1.  vstup = input
2.  výstup = print
3.
4.  input = "ahoj"  # toto sa dá, nesmie sa to však robiť
5.
6.  #if = 84  # toto by sa ani nedalo - program hneď vypíše chybu
7.  #for = 5
8.
9.  a = vstup("Zadaj text: ")
10.
11. výstup("Zadané a: " + a + input)
12.
13. b = input("Zadaj znova: ")      # toto sa už nedá, pretože program už považuje input
14.                                 # za premennú a nie za referenciu na funkciu, čo je zle!
15.                                 # vypíše: TypeError: 'str' object is not callable,
16.                                 # teda, že nedá sa zavolať (použiť) vstup (teda input)
```

- nesmie tiež obsahovať žiadne **operátory (`+ - * /`), úvodzovky, apostrofy** ani **medzeru,**
- identifikátory sú **case sensitive**, t. j. rozlišujeme malé a veľké písmená – premenná **`a`** je iná ako premenná **`A`**,
- nech vás nepomýli, že ak v matematike *abc* by znamenal súčin troch premenných ($a \cdot b \cdot c$), v programovaní **`abc`** je názov (teda identifikátor) jednej premennej; ak by sme chceli vyjadriť ich súčin, musíme zapísať: **`a*b*c`**,
- pomenovávajte premenné vždy zrozumiteľne – tak, aby ste neskôr vedeli, na akú hodnotu odkazujú, keď ich budete mať viac.

4.5 Vstup a výstup*

Už sme párkrát použili príkaz (funkciu) `print()` a `input()`, no ešte sme si nepovedali, čo presne robia a na čo sa dajú použiť, aj keď už možno tušíte. V nasledujúcich programoch ich totiž budeme hojne využívať, pretože **každý program je písaný pre koncového používateľa,[29] od ktorého program vyžaduje zadávanie rôznych údajov klávesnicou a od ktorého zas používateľ vyžaduje reakciu – informačný výpis.**

V Pythone na získavanie vstupných údajov slúži funkcia `input()` s voliteľným parametrom – vkladaním informačného textu:

- Je to tzv. funkcia s návratovou hodnotou,[30] ktorej úlohou je **pozastaviť vykonávanie programu a pýtať si vstupný text v shelli od používateľa ukončený enterom, ktorého obsah uloží do danej premennej.**
- zápis: **`a = input("Zadaj text: ")`**
- Hodnota, ktorú zadáme do klávesnice, sa uloží ako **hodnota identifikátora.** Do zátvorky medzi úvodzovky alebo apostrofy zadávame inštrukciu (otázku) pre používateľa. Ak nezadáme do zátvorky nič, v shelli sa iba čaká na vstup. Je však vždy vhodné zadať medzi úvodzovky, čo konkrétne sa pýtame, pretože to by bolo ako čakať od niekoho odpoveď na nepoloženú otázku.
- Ak dáme na konci programu `input()`, program sa **neukončí,** kým nestlačíme enter – ale to už vieme.

[29] *Dôležitá terminologická poznámka: **správne je používateľ, nie užívateľ.** Počítač ani program „nejeme,“ ale používame (use = použiť; take = užiť, požiť; je to z češtiny zle preložené slovo – **uživatel = používateľ,** poživatel = užívateľ). Je to veľmi častá a hrubá chyba, bohužiaľ, hlavne v médiách.*

[30] *K tomu, čo to znamená, sa dostaneme v ďalšej kapitole, zatiaľ len stačí vedieť to používať.*

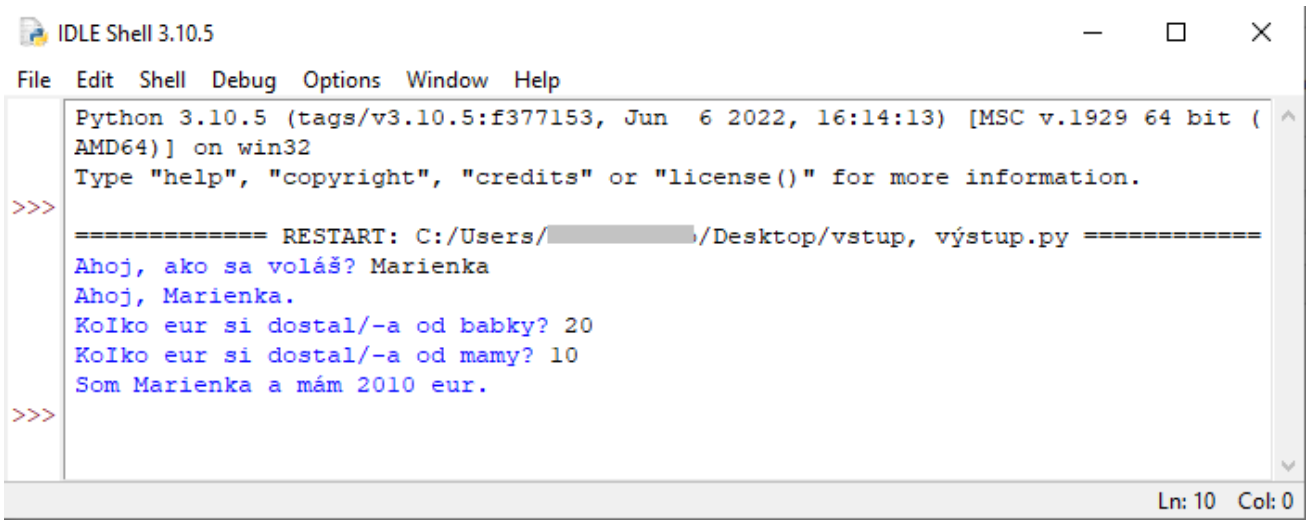
Na zobrazovanie rôznych hodnôt (informačných výpisov alebo sledovanie hodnôt premenných) počas vykonávania programu slúži funkcia **`print()`**

- Je to tzv. funkcia bez návratovej hodnoty, čo znamená, že jej výsledok sa nedá uložiť do premennej.
- **Zabezpečuje výpis hodnôt** na shell.
- zápis: **`print("Text", end = ", ")`**
- Hodnôt premenných môžeme zobrazovať aj **viac naraz,** napr. **`print(a, b)`**.
- Ak zadáme voliteľný parameter **`end`,** medzi hodnoty výpisu sa vsunie to, čo je v úvodzovkách (v našom prípade konkrétne čiarka a medzera), ak ho nezadáme, vypisuje sa **každá hodnota do nového riadka.**

Poďme si to vyskúšať naprogramovaním krátkeho rozhovoru:

```
1. meno = input("Ahoj, ako sa voláš? ")
2. print("Ahoj, " + meno + ".")
3. babka = input("Koľko eur si dostal/-a od babky? ")
4. mama = input("Koľko eur si dostal/-a od mamy? ")
5. print("Som " + meno + " a mám " + babka + mama + " eur.")
```

po spustení:



Zdá sa, že rozhovor funguje, ale tú sumu vypočítalo nejako čudne, však? Ono to v podstate iba spojilo dokopy 20 a 10. Prečo? Hneď v ďalšej kapitole si to ideme vysvetliť.

4.6 Údajové typy a operácie s údajovými typmi*

Python poskytuje niekoľko rôznych typov údajov, na začiatok sa zoznámime s tromi základnými typmi: **celými číslami, desatinnými číslami a znakovými reťazcami.**

celé čísla

- v Pythone sa nazývajú **`int`** (z angl. integer = celé číslo)
- majú rovnaký význam, ako ich poznáme z matematiky: zapisujú sa v desiatkovej sústave a môžu sa začínať znamienkom plus alebo mínus
- ich veľkosť (počet cifier) je obmedzená len kapacitou pracovnej pamäte Pythona (hoci aj niekoľko miliónov cifier)

desatinné čísla

- v Pythone sa nazývajú **`float`** (z angl. float = číslo s plávajúcou/pohyblivou desatinnou čiarkou)
- obsahujú desatinnú bodku alebo exponenciálnu časť (napr. **`1e+15`** znamená $1 \cdot 10^{15}$)
- môže vzniknúť aj ako výsledok niektorých operácií (napr. delením dvoch celých čísel)
- majú obmedzenú presnosť (približne 16-17 platných cifier)

znakové reťazce

- v Pythone sa nazývajú **`str`** (z angl. string = reťazec)

- je to postupnosť znakov (neskôr sa tomu budeme venovať hlbšie)
- ich dĺžka (počet znakov) je obmedzená len kapacitou pracovnej pamäte Pythona
- **uzatvárame ich medzi apostrofy alebo úvodzovky,** oba zápisy sú rovnocenné – reťazec sa musí končiť tým istým znakom, akým sa začal (apostrof alebo úvodzovka) a takto zadaný reťazec nesmie presiahnuť jeden riadok
- môže obsahovať aj písmená s diakritikou
- prázdny reťazec má dĺžku 0 a zapisujeme ho ako `""` alebo `''`

S uvedenými údajovými typmi môžeme ďalej pracovať. Na to sú definované nasledujúce operácie:

celočíselné operácie – oba operandy (čísla) musia byť celočíselného typu

`+`	sčítanie	`1 + 2` má hodnotu `3`
`-`	odčítanie	`2 - 5` má hodnotu `-3`
`*`	násobenie	`3 * 37` má hodnotu `111`
`//`	celočíselné delenie	`22 // 7` má hodnotu `3`
`%`	zvyšok po delení (tzv. modulo)	`22 % 7` má hodnotu `1`
`**`	umocňovanie	`2 ** 8` má hodnotu `256`

operácie s desatinnými číslami – aspoň jeden z operandov musí byť desatinného typu (okrem delenia `/`)

`+`	sčítanie	`1 + 0.2` má hodnotu `1.2`
`-`	odčítanie	`6 - 2.86` má hodnotu `3.14`
`*`	násobenie	`1.5 * 2.5` má hodnotu `3.75`
`/`	delenie	`23 / 3` má hodnotu `7.666666666666667`
`//`	delenie zaokrúhlené nadol	`23.0 // 3` má hodnotu `7.0`
`%`	zvyšok po delení	`23.0 % 3` má hodnotu `2.0`
`**`	umocňovanie	`3 ** 3.0` má hodnotu `27.0`

operácie so znakovými reťazcami

`+`	zreťazenie (spojenie dvoch reťazcov)	`'a' + 'b'` má hodnotu `'ab'`
`*`	viacnásobné zreťazenie reťazca	`3 * 'x'` má hodnotu `'xxx'`

Zreťazenie dvoch reťazcov je bežné aj v iných programovacích jazykoch. Viacnásobné zreťazenie je dosť výnimočné.

Vráťme sa späť k problému sčítavania čísel. Už vieme, ako sa zo sumy 20 a 10 zrazu stala 2010 – bol to textový reťazec. Ale ako to opraviť?

Medzi rozdielnymi údajovými typmi sa nedajú robiť operácie. Na konverziu údajových typov sa používa tzv. **pretypovanie** – teda **premena jedného údajového typu na iný údajový typ.**

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

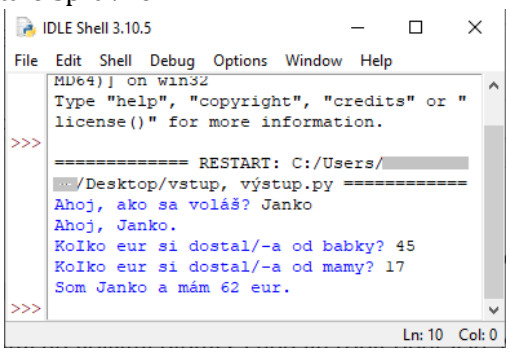
- **`int(hodnota)`** z danej hodnoty vyrobí celé číslo, napr.:
  - **`int(3.14)`** → **`3`**
  - **`int('37')`** → **`37`**
- **`float(hodnota)`** z danej hodnoty vyrobí desatinné číslo, napr.:
  - **`float(333)`** → **`333.0`**
  - **`float('3.14')`** → **`3.14`**
- **`str(hodnota)`** z danej hodnoty vyrobí znakový reťazec, napr.:
  - **`str(356)`** → **`'356'`**
  - **`str(3.14)`** → **`'3.14'`**

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

Poďme opraviť program:

```
1.  meno = input("Ahoj, ako sa voláš? ")
2.  print("Ahoj, " + meno + ".")
3.  babka = int(input("Koľko eur si dostal/-a od babky? "))
4.        # vstup sme museli pretypovať na celé číslo, aby sme
5.        # vedeli vykonať operáciu sčítania
6.  mama = int(input("Koľko eur si dostal/-a od mamy? "))
7.        # takisto
8.  print("Som " + meno + " a mám " + str(babka + mama) + " eur.")
9.        # vykonáme operáciu sčítania súm a nakoniec nesmieme
10.       # zabudnúť opäť zmeniť číslo na reťazec, ktorý vypisujeme
```

po oprave vidíme, že sumy už spočítalo správne:



Keďže už vieme, ako fungujú premenné, vstup a výstup, ako a kedy je potrebné pretypovať údajové typy, poďme si to vyskúšať aj v nám už známom grafickom rozhraní. Vašou úlohou bolo nakresliť si panáčika. Poďme skúsiť meniť jeho veľkosť pomocou vstupu a premenných. Váš kód vyzerá zrejme takto:

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 200, height = 300, bg = "white")
3.  canvas.pack()
4.
5.  canvas.create_polygon(90,130,  30,170,  40,175,  100,135, fill="grey",width=2, outline="black")
6.  canvas.create_polygon(110,130, 170,170, 160,175, 100,135, fill="grey",width=2, outline="black")
7.  canvas.create_polygon(90,250,  30,290,  40,295,  100,255, fill="grey",width=2, outline="black")
8.  canvas.create_polygon(110,250, 170,290, 160,295, 100,255, fill="grey",width=2, outline="black")
9.  canvas.create_rectangle(90, 120, 110, 255, fill = "orange", width = 2)
10. canvas.create_oval(50, 20, 150, 120, fill = "navajowhite", width = 2)
11. canvas.create_oval(75, 60, 85, 70, fill = "black", width = 1)
12. canvas.create_oval(115, 60, 125, 70, fill = "black", width = 1)
13. canvas.create_line(85, 95, 100, 100, 115, 95, fill = "black", width = 2)
```

Čo teraz vlastne ideme robiť? Musíme každú súradnicu nechať násobiť nejakým číslom (ak nechceme panáčika zdeformovať, každú súradnicu musíme násobiť vždy tým istým číslom), teda koeficientom. Pomenujme ho **`k`**. Nech program po spustení toto **`k`** vyžaduje od používateľa. Nezabudnime, že **`k` musí byť číslo, nesmie to byť reťazec.** Musíme ho pretypovať. Ale na aký typ? Celočíselný alebo desatinný? Logicky by vám asi napadlo, že keď pracujeme s pixelmi, neexistuje také niečo ako pol pixela, teda to musí byť celé číslo (**`integer`**). Áno, to je pravda, avšak rozhranie **`tkinter`** má tento problém vyriešený a súradnice zaokrúhľuje vždy na najbližšie celé číslo. Teda použijeme pretypovanie na **`float()`**, aby sme mohli panáčika nielen zväčšovať, ale aj zmenšovať.

Program upravíme do takejto podoby:

```
1.  #koeficient - aby sa zbytočne neroztiahol kód, použijeme len názov k
2.  k = float(input("Zadaj koeficient zväčšenia: "))
3.
4.  import tkinter
5.  canvas = tkinter.Canvas(width = k*200, height = k*300, bg = "white")
6.  canvas.pack()
7.
8.  canvas.create_polygon(k*90,  k*130, k*30,  k*170, k*40,  k*175, k*100, k*135,
                          fill = "grey", width = 2, outline = "black")
9.  canvas.create_polygon(k*110, k*130, k*170, k*170, k*160, k*175, k*100, k*135,
                          fill = "grey", width = 2, outline = "black")
10. canvas.create_polygon(k*90,  k*250, k*30,  k*290, k*40,  k*295, k*100, k*255,
                          fill = "grey", width = 2, outline = "black")
11. canvas.create_polygon(k*110, k*250, k*170, k*290, k*160, k*295, k*100, k*255,
                          fill = "grey", width = 2, outline = "black")
12. canvas.create_rectangle(k*90, k*120, k*110, k*255, fill = "orange", width = 2)
13. canvas.create_oval(k*50, k*20, k*150, k*120, fill = "navajowhite", width = 2)
14. canvas.create_oval(k*75, k*60, k*85, k*70, fill = "black", width = 1)
15. canvas.create_oval(k*115, k*60, k*125, k*70, fill = "black", width = 1)
16. canvas.create_line(k*85, k*95, k*100, k*100, k*115, k*95, fill = "black", width = 2)
```

Vyskúšame spustiť stlačením **F5** alebo **Run → Run Module.** Po spustení od nás shell požaduje zadanie koeficienta. Skúsme napríklad `3`. Po stlačení tlačidla enter vidíme, že panáčik je zrazu trojnásobne väčší. Zatvorme okno s plátnom a spustime program opäť. Skúsme tentoraz zadať koeficient `0.5`.[31] Vidíme, že panáčik je o polovicu menší ako pôvodný.

Úloha 6

Skúste takto „oživiť“ aj vlajku Grónska.

Príklad 4

Vytvorte takúto jednoduchú značku rezervácie parkovacieho miesta.

Možné riešenie:

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 500, height = 700, bg = "green")
3.  canvas.pack()
4.
5.  a = input("zadaj EČV: ")
6.
7.  canvas.create_rectangle(100, 100, 400, 500, fill = "blue", width = 6)
8.  canvas.create_text(250, 300, text = "P", font = "arial 200 bold", fill = "white")
9.  canvas.create_rectangle(100, 550, 400, 600, fill = "white", width = 6)
10. canvas.create_text(250, 575, text = a, font = "arial 30 bold", fill = "black")
```

Úloha 7

Nakreslite nasledujúce značky tak, aby sa dala meniť ich veľkosť na plátne aj maximálna povolená rýchlosť podľa hodnôt zadaných používateľom.

4.7 Kreslenie pravidelných n-uholníkov a hviezd[32]

Už poznáme niektoré, tzv. **štandardné funkcie,** ktoré sú zadefinované už pri štarte Pythona – napríklad funkcie `int()`, `float()`, `str()` pretypujú zadanú hodnotu na iný typ alebo funkcie `print()` a `input()` sú určené na výpis textov a prečítanie textu zo vstupu. Hovorí sa im inak aj zabudované funkcie (tzv. **built-in functions**) a po ich použití sa ich názvy vždy zafarbia nafialovo.[33] Štandardných funkcií je oveľa viac a s mnohými z nich sa zoznámime neskôr. Teraz sa zoznámime postupne s dvoma novými modulmi – `math` a `random` (predstavme si ich ako knižnice – súhrn užitočných funkcií), ktoré, hoci nie sú štandardne zabudované, my ich budeme často potrebovať. Ak potrebujeme pracovať s nejakým modulom, musíme to najskôr Pythonu oznámiť. Slúži na to príkaz `import`. Už ste si možno všimli, že keď sme kreslili útvary, importované funkcie na ich kreslenie fialové

[31] *Pozor, používame desatinnú bodku, nie čiarku!*
[32] *Zaradenie tejto podkapitoly 4.7 aj pre žiakov stredných škôl nižších ročníkov nechávame na zvážení vyučujúceho, keďže sa tu pasívne pracuje s goniometrickými funkciami sínus a kosínus – používajú sa však len na vyčíslenie hodnôt súradníc.*
[33] *Ak teda nemáte nastavené inak a používate svetlý režim farieb a IDLE Python.*

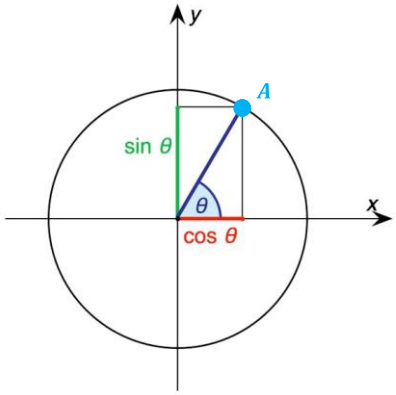
neboli – to preto, **lebo nie sú štandardné** (zabudované). A ako ste si isto teraz všimli, `import` je ďalšie dohovorené (rezervované) slovo – oranžovo zafarbené.

Pomocou zápisu `import math` umožníme našim programom pracovať s knižnicou matematických funkcií. V skutočnosti týmto príkazom Python vytvorí novú premennú `math`. Knižnica v tomto module obsahuje napríklad tieto matematické funkcie: `sin()`, `cos()`, `sqrt()`. Lenže s takýmito funkciami nemôžeme pracovať priamo. Python nepozná ich mená, pozná jediné meno, a to meno modulu `math`. Keďže tieto funkcie sa nachádzajú priamo v tomto module, budeme k nim pristupovať tzv. **bodkovou notáciou,** t. j. za meno modulu uvedieme prvok (v tomto prípade funkciu) z daného modulu – napríklad `math.sin()` označuje výpočet hodnoty sínus a `math.sqrt()` výpočet druhej odmocniny čísla.

Nám však je takáto bodková notácia už známa a už sme ju použili – bez toho, aby sme si to uvedomovali. Tak sme predsa importovali celý grafický modul `tkinter` a používali sme jeho funkcie – napr. `canvas.create_text()`, `canvas.create_line()`, `canvas.update()` alebo `canvas.after()`.

PRÍKLAD 5

Ostaňme pri kreslení vlajok. Mnohé vlajky obsahujú pravidelné hviezdy. Ako takú pravidelnú hviezdu (povedzme päťcípu) nakresliť, keď vieme, že všetky vrcholy sa nachádzajú na kružnici a sú od seba rovnomerne vzdialené? Asi to už použitím „obyčajných čísel“ ako súradníc x a y nebude vyzerať tak pekne. Musíme si pomôcť zrejme inak – rozdeliť kružnicu na 5 rovnakých kruhových výsekov, čo znamená 360 stupňov rozdeliť na pätiny – po 72°. Teraz už len stačí spomenúť si na matematiku a na to, ako sa dá vypočítať súradnica napríklad tohto bodu *A* na kružnici, ak poznáme uhol [4]:



Ako vidíme, x-ová súradnica má hodnotu cos (θ) a y-ová má hodnotu sin (θ). Také funkcie knižnica `math` pozná. Poďme ich vyskúšať.

Ako prvé, importujeme `tkinter` a `math`, už vieme, prečo. Budeme využívať funkcie na sínus, kosínus a na prevod stupňov na radiány, pretože funkcie sínus a kosínus neprijímajú hodnoty v stupňoch, ale v radiánoch. Ich zápis je však na použitie v súradniciach zbytočne zdĺhavý, a preto sme si dovolili uložiť mená funkcií do nových, kratších premenných.

Samotný polygón dostáva dvojice súradníc takéhoto druhu:

```
sx + cos(rad(stupne))*k
sy + sin(rad(stupne))*k
```

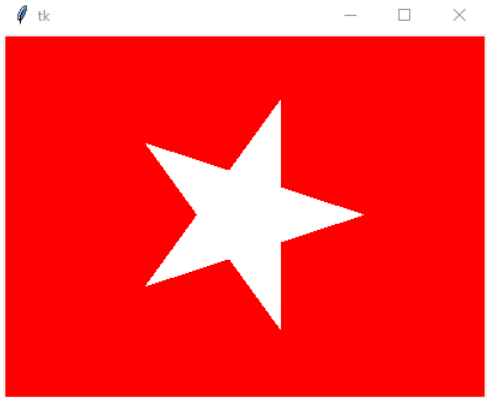
Poďme sa pozrieť na kód:

```
1.  import tkinter
2.  import math
3.
4.  # premenné umožňujú zjednodušiť a sprehľadniť zápis funkcií:
5.  cos = math.cos
6.  sin = math.sin
7.  rad = math.radians
8.
9.  šírka, výška = 400, 300
10.
11. canvas = tkinter.Canvas(width = šírka, height = výška, bg = "red")
12. canvas.pack()
13.
```

```
14. k = 100
15. sx = šírka // 2
16. sy = výška // 2
17.
18. # kosínus a sínus prijímajú hodnoty v radiánoch, preto sme museli
19. # použiť funkciu prevodu stupňov na radiány
20.
21. # funkcie kosínus a sínus sme vynásobili konštantou, aby sa dali
22. # použiť ako súradnice
23.
24. canvas.create_polygon(sx + cos(rad(0))*k,   sy + sin(rad(0))*k,
25.                       sx + cos(rad(144))*k, sy + sin(rad(144))*k,
26.                       sx + cos(rad(288))*k, sy + sin(rad(288))*k,
27.                       sx + cos(rad(72))*k,  sy + sin(rad(72))*k,
28.                       sx + cos(rad(216))*k, sy + sin(rad(216))*k,
29.                       fill = "white", outline = "white")
```

Prečo sme ale vynásobili hodnoty sínusu a kosínusu takou veľkou premennou,[34] v našom prípade 100? Samotné funkcie sínus a kosínus dávajú veľmi malé čísla – predsa hodnoty oboch funkcií v goniometrii spadajú do intervalu $\langle -1; 1 \rangle$, čo po premietnutí do grafiky dáva nula, nanajvýš jeden pixel na obrazovke, čo by sa ani nezobrazilo na plátne.

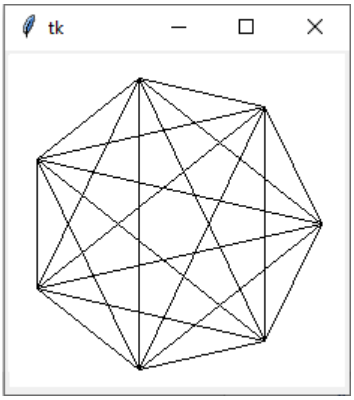
Súradnice sme ešte posunuli o **`sx`** vodorovne a o **`sy`** zvislo – v našom konkrétnom prípade do stredu plátna. Samotný bod $[sx; sy]$ predstavuje teraz stred hviezdy a od tohto stredu odpočítavame, resp. pripočítavame hodnoty vrcholov. Vodorovnú súradnicu však ešte treba posunúť viac doprava, keďže turecká vlajka má ešte aj polmesiac. Tiež treba opraviť rotáciu hviezdy, na tureckej vlajke je oproti tejto nakreslenej pootočená ešte o 36 stupňov. Ale to už ľahko zvládnete.



Ak má funkcia, ktorú používame, veľa parametrov,[35] môžeme si to sprehľadniť tým, že za niektorým parametrom dáme enter a pokračujeme s písaním ďalších parametrov automaticky pod zátvorku (ako z riadka 24 na riadok 25).

Úloha 8

Nakreslite takýto pravidelný 7-uholník – buď použitím funkcie **`create_polygon()`** alebo **`create_line()`**. Neskôr si ukážeme, ako sa to dá oveľa jednoduchšie.



*[34] Aj keď z matematického uhla pohľadu je to konštanta, v programovaní sa tomu hovorí premenná, aj keď sa nebude meniť – ide o to, že **sa dá meniť**. Konštanty sú len napr. e alebo π, na ktoré existujú samostatné funkcie, ktoré ich aj vyčísľujú, jednu z nich si neskôr aj ukážeme.*

[35] Neskôr si vysvetlíme, čo je funkcia a aj ako fungujú parametre – zatiaľ stačí vedieť, že to je to, čo píšeme do zátvoriek a oddeľujeme čiarkami, v našom prípade tých desať súradníc, výplň a obrys.

4.8 Generovanie náhodnej veľkosti a pozície*

Pomocou ďalšieho modulu, ktorý si teraz ukážeme, sa naučíme náhodne meniť pozíciu, veľkosť nakreslených útvarov aj farbu. Tento modul obsahujúci funkcie, ktoré zabezpečujú náhodný výber, sa nazýva **`random`**.

My z tejto knižnice využijeme najmä tieto dve funkcie: **`randrange()`** – vyberie náhodné číslo z postupnosti celých čísel a **`choice()`** – vyberie náhodný prvok z nejakej postupnosti, napríklad zo znakového reťazca – postupnosti znakov. Aby sme mohli pracovať s týmito funkciami, nesmieme zabudnúť napísať **`import random`**.

A ako určíme rozsah náhodne generovaných hodnôt? Funkcia **`randrange()`** prijíma jeden, dva alebo tri parametre, pričom povinný je aspoň jeden. Dôležité je uvedomiť si, že sa generujú len celé čísla. Teda:

- **prvý parameter (ak je iba jeden)** určuje hornú otvorenú hranicu maximálnej náhodne vygenerovanej hodnoty, pričom spodná hranica sa predpokladá automaticky nula (vrátane nej),
- **ak sú parametre dva alebo tri,** tak prvý parameter určuje dolnú hranicu minimálnej náhodne vygenerovanej hodnoty **vrátane** nej, druhý parameter potom určuje maximálnu hodnotu **okrem** nej,
- **tretí parameter** určuje **krok** – rozostup medzi hodnotami.

Príklady:

príklad	*množina všetkých možných vygenerovaných hodnôt*
`randrange(10)`	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
`randrange(0, 10)`	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
`randrange(0, 10, 1)`	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
`randrange(3, 10)`	{3, 4, 5, 6, 7, 8, 9}
`randrange(3, 10, 2)`	{3, 5, 7, 9}
`randrange(10, 100, 10)`	{10, 20, 30, 40, 50, 60, 70, 80, 90}
`randrange(10, 1)`	{ } (nič)
`randrange(1, 1)`	{ } (nič)

Zápis teda môže vyzerať napríklad takto: **`náhodné_číslo = random.randrange(4, 101, 3)`**.

PRÍKLAD 6

Vyskúšajme teraz pomocou tejto funkcie generovať náhodnú veľkosť panáčika z úlohy 1. Vždy, keď spustíme program, vykreslí sa na plátne panáčik inej veľkosti bez toho, aby sme koeficient zmenšenia alebo zväčšenia museli zadávať.

```
1.  import tkinter
2.  import random # pridali sme import knižnice
3.
4.  šírka, výška = 900, 900 # trošku sme zväčšili rozmery plátna
5.  canvas = tkinter.Canvas(width = šírka, height = výška, bg = "white")
6.  canvas.pack()
7.
8.  k = random.randrange(30)/10 # koeficient sme nechali vytvárať náhodne a keďže chceme aj
    desatinné čísla (zmenšenie), podelíme výsledok 10 a dostaneme tak interval od 0 do 3
9.
10. canvas.create_polygon(k*90,  k*130, k*30,  k*170, k*40,  k*175, k*100, k*135,
                          fill = "grey", width = 2, outline = "black")
11. canvas.create_polygon(k*110, k*130, k*170, k*170, k*160, k*175, k*100, k*135,
                          fill = "grey", width = 2, outline = "black")
12. canvas.create_polygon(k*90,  k*250, k*30,  k*290, k*40,  k*295, k*100, k*255,
                          fill = "grey", width = 2, outline = "black")
13. canvas.create_polygon(k*110, k*250, k*170, k*290, k*160, k*295, k*100, k*255,
                          fill = "grey", width = 2, outline = "black")
14. canvas.create_rectangle(k*90, k*120, k*110, k*255, fill = "orange", width = 2)
15. canvas.create_oval(k*50, k*20, k*150, k*120, fill = "navajowhite", width = 2)
16. canvas.create_oval(k*75, k*60, k*85, k*70, fill = "black", width = 1)
17. canvas.create_oval(k*115, k*60, k*125, k*70, fill = "black", width = 1)
18. canvas.create_line(k*85, k*95, k*100, k*100, k*115, k*95, fill = "black", width = 2)
```

4.9 Pomôcka – tlačidlo*

Aby sme nemuseli pri testovaní náhodnosti stále spúšťať a zastavovať program, pomôžeme si tak, že na spodnú časť plátna dáme tlačidlo, ktoré toto prekreslenie urobí. To, čo teraz spravíme, súvisí s tvorbou podprogramov. K tomu sa dostaneme v inej kapitole. Teraz sa preto budeme držať iba „vysvetlenia“ „lebo to tak je“.

Chceme vytvoriť tlačidlo tak, aby niečo prekresľovalo alebo aktualizovalo. Musíme si uvedomiť, ktoré príkazy stačí vykonať raz (to sú vytvorenie plátna s rozmermi, farbou pozadia, niektoré premenné, ktoré sa nemenia) a čo potrebujeme vykonávať opakovane.

To, čo potrebujeme vykonávať opakovane, to takzvane „zabalíme“ do funkcie. Ešte sme si síce nepovedali, čo je to funkcia (k tomu sa dostaneme neskôr), vedzme teraz aspoň, ako sa to robí. Zhromaždime si všetky príkazy, ktoré sa budú meniť. Do riadka **pred ne** napíšme **`def názov_funkcie():`** (nezabudnite prázdne zátvorky a dvojbodku), označíme kurzorom všetky požadované príkazy a stlačíme tabulátor, aby sa odsadili od okraja, čím vyjadríme, že patria tej funkcii. Tým sme príkazy „zabalili“ do funkcie – takto:

```
1.  import tkinter
2.  import random
3.
4.  šírka, výška = 900, 900
5.  canvas = tkinter.Canvas(width = šírka, height = výška, bg = "white")
6.  canvas.pack()
7.
8.  def kresli_panáčika():   # všetko kreslenie bude odteraz vložené v takejto
9.                           # funkcii a odsadené tabulátorom zľava
10.     canvas.delete("all") # pridali sme príkaz na vymazanie plátna
11.     k = random.randrange(30)/10
12.     canvas.create_polygon(k*90,  k*130, k*30,  k*170, k*40,  k*175, k*100, k*135,
                              fill = "grey", width = 2, outline = "black")
13.     canvas.create_polygon(k*110, k*130, k*170, k*170, k*160, k*175, k*100, k*135,
                              fill = "grey", width = 2, outline = "black")
14.     canvas.create_polygon(k*90,  k*250, k*30,  k*290, k*40,  k*295, k*100, k*255,
                              fill = "grey", width = 2, outline = "black")
15.     canvas.create_polygon(k*110, k*250, k*170, k*290, k*160, k*295, k*100, k*255,
                              fill = "grey", width = 2, outline = "black")
16.     canvas.create_rectangle(k*90, k*120, k*110, k*255, fill = "orange", width = 2)
17.     canvas.create_oval(k*50, k*20, k*150, k*120, fill = "navajowhite", width = 2)
18.     canvas.create_oval(k*75, k*60, k*85, k*70, fill = "black", width = 1)
19.     canvas.create_oval(k*115, k*60, k*125, k*70, fill = "black", width = 1)
20.     canvas.create_line(k*85, k*95, k*100, k*100, k*115, k*95, fill = "black", width = 2)
21.
22. # tu sú príkazy na vytvorenie tlačidla a priradenie jeho významu
23. tlačidlo = tkinter.Button(text = "Kreslenie", command = kresli_panáčika)
24. tlačidlo.pack(side = tkinter.BOTTOM)
```

Za touto funkciou (pokojne aj na konci programu) definujeme nové tlačidlo takto:

```
tlačidlo = tkinter.Button(text = "Váš text", command = názov_funkcie)
tlačidlo.pack(side = tkinter.BOTTOM)
```

čím mu povieme, aby bolo pomenované **`Váš text`**, aby vykonávalo príkaz **`názov_funkcie`** a bolo umiestnené na spodku plátna. Otestujme, že jeho opätovným klikaním sa mení veľkosť panáčika bez toho, aby sme plátno museli opakovane zatvárať a potom zas spúšťať tlačidlom **F5.**

Pre prehľadnosť uvádzame vzor:[36]

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 150, height = 150, bg = "white")
3.  canvas.pack()
4.
5.  # tu budú príkazy, ktoré stačí vykonať raz
6.
7.  def akcia_tlačidla():
8.      # tu budú príkazy takto odsadené zľava, napríklad:
9.      canvas.delete("all") # na vymazanie plátna
10.     print("Fungujem")
11.
12. tlačidlo = tkinter.Button(text = "Váš text", command = akcia_tlačidla)
13. tlačidlo.pack(side = tkinter.BOTTOM)
```

[36] *Odporúčame dovoliť ho žiakom používať aj na previerkach ako „ťahák“.*

4.10 Generovanie a vykresľovanie iných náhodných javov*

Druhou užitočnou funkciou z modulu `random`, ktorú budeme často používať, je funkcia **`choice()`**. Táto má len jeden parameter, ktorým je **postupnosť hodnôt.** My do nej vyskúšame vložiť zopár farieb, z ktorých chceme, aby náhodne vyberala. Skúsme to hneď s náhodne vybratými písmenami, nech je to zaujímavejšie.

Ako však „povedať" funkcii, z čoho má vyberať? Funkcia prijíma **len jeden parameter,** teda, ak chceme vybrať napríklad jedno písmeno abecedy, zapíšeme:

```
random.choice("abcdefghijklmnopqrstuvwxyz")
```

Keďže už vieme, že znakový reťazec je postupnosť znakov, funkcia vyberie náhodnú hodnotu z tejto postupnosti, teda náhodný znak.

Čo však názvy farieb? Tie asi bude treba od seba nejako oddeliť. Ak chceme vybrať takúto náhodnú hodnotu, musíme použiť tzv. **n-ticu (tuple) alebo pole (array).** Sú to pre nás nové údajové typy, ktorými sa budeme bližšie zaoberať v kapitole o poliach, vedzme však zatiaľ, že sú to akési **usporiadané n-tice,** teda opäť je to istým spôsobom postupnosť a svojím spôsobom jedna hodnota (aj keď obsahuje viac položiek – hodnôt). Zapisujú sa do oblých `()` alebo hranatých `[]` zátvoriek a ich členy sa oddeľujú čiarkou.[37] Pole čísel teda napríklad vyzerá takto: `[2, 4, 8, 1, 2, 4, 0]` a pole textových reťazcov napríklad takto: `["ahoj", "vitaj", "dobrý deň"]`. N-tica vyzerá rovnako, avšak uzatvára sa do oblých zátvoriek. Zápis bude napríklad:

```
farba = random.choice(("red", "green", "blue"))
```

alebo `farba = random.choice(["red", "green", "blue"])`.

PRÍKLAD 7

Napíšte program, ktorý takto náhodne vykresľuje písmenká. Tie sú rôznych typov písma, rôznej veľkosti, rezu, farby, pozície. Skúste najskôr sami a potom sa pozrite na riešenie.

Riešenie:

```
1.  import tkinter
2.  import random
3.
4.  canvas = tkinter.Canvas(width = 500, height = 500, bg =
    "white")
5.  canvas.pack()
6.
7.  def akcia_tlačidla():
8.      farba     = random.choice(("red", "green", "blue", "purple", "black",
9.                                 "khaki", "brown", "cyan"))
10.     písmeno   = random.choice("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
11.     písmo     = random.choice(("Times New Roman", "Arial", "Calibri",
12.                                "Comic Sans MS", "Tahoma", "Courier New"))
13.     veľkosť   = random.randrange(10, 50)
14.     poloha_x  = random.randrange(25, 475)
15.     poloha_y  = random.randrange(25, 475)
16.     rez       = random.choice(("normal", "bold", "italic", "bold italic"))
17.
18.     canvas.create_text(poloha_x, poloha_y, text = písmeno,
19.                        font = (písmo, veľkosť, rez), fill = farba)
20.
21. tlačidlo = tkinter.Button(text = "Generuj písmenko", command = akcia_tlačidla)
22. tlačidlo.pack(side = tkinter.BOTTOM)
```

PRÍKLAD 8

Napíšte program, ktorý po stlačení tlačidla nakreslí vždy iný trojuholník (na náhodnú pozíciu, náhodnej veľkosti, tvaru aj farby) a aby mal označené vrcholy A, B, C – tiež náhodne zvolenými farbami.

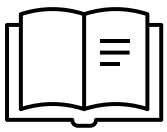
[37] *Hranaté zátvorky napíšeme na slovenskej klávesnici stlačením Alt Gr + F, Alt Gr + G.*

Riešenie:

```
1.  import tkinter
2.  import random
3.
4.  canvas = tkinter.Canvas(width = 700, height = 700, bg = "white")
5.  canvas.pack()
6.
7.  def akcia_tlačidla():
8.      canvas.delete("all")
9.
10.     a = random.randrange(50, 650)
11.     b = random.randrange(50, 650)
12.     c = random.randrange(50, 650)
13.     d = random.randrange(50, 650)
14.     e = random.randrange(50, 650)
15.     f = random.randrange(50, 650)
16.
17.     f1 = random.choice(("red", "green", "blue", "orange", "brown"))
18.     f2 = random.choice(("red", "green", "blue", "orange", "brown"))
19.     f3 = random.choice(("red", "green", "blue", "orange", "brown"))
20.     f4 = random.choice(("red", "green", "blue", "orange", "brown"))
21.
22.     canvas.create_polygon(a, b, c, d, e, f, fill = f1, width = 0)
23.     canvas.create_text(a, b, text = "A", fill = f2, font = "Arial 30 bold")
24.     canvas.create_text(c, d, text = "B", fill = f3, font = "Arial 30 bold")
25.     canvas.create_text(e, f, text = "C", fill = f4, font = "Arial 30 bold")
26.
27.
28. tlačidlo = tkinter.Button(text = "Trojuholník", command = akcia_tlačidla)
29. tlačidlo.pack(side = tkinter.BOTTOM)
```

Úloha 9

Keďže ste si už precvičili kreslenie základných geometrických útvarov na canvas, pohrajte sa s nakreslením záhrady (dom, kvety, plot, strom, slnko atď.), lesa, zimnej krajiny, prípadne si vymyslite iné, komplexnejšie zadania – napríklad, po stlačení tlačidla nechajte generovať niekoľko snehových vločiek, kvetov, stromov – rôznej veľkosti, rôznej pozície, farby a iných vlastností.



Zhrnutie

- ✓ Premenná slúži na zapamätanie hodnoty tak, aby sme ju mohli použiť aj neskôr. Pomenováva existujúcu hodnotu v pamäti.
- ✓ V Pythone vzniká tzv. dynamickým priradením.
- ✓ V iných jazykoch rozlišujeme medzi deklaráciou a definíciou premennej.
- ✓ Priraďovací príkaz sa skladá z identifikátora, priraďovacieho znaku a hodnoty premennej.
- ✓ Po priradení hodnoty premennej identifikátoru sa vytvorí záznam v pamäti identifikátorov aj v pamäti hodnôt operačnej pamäte, kde identifikátor referuje (čiže odkazuje) na hodnotu.
- ✓ Python ponúka aj možnosť tzv. hromadného priraďovania.
- ✓ Identifikátor premennej v Pythone môže obsahovať diakritiku, čísla, podčiarkovník, nesmie sa však začínať číslom, obsahovať operátory, úvodzovky, apostrof ani medzeru; nesmú to byť dohovorené slová a nemali by to byť referencie (nielen vstavaných) funkcií.
- ✓ Vstup zabezpečuje funkcia **`input()`**, pozastavuje vykonávanie programu a pýta si text od používateľa.
- ✓ Výstup zabezpečuje funkcia **`print()`**, vypisuje danú hodnotu na shell.
- ✓ Základné údajové typy, ktoré poznáme, sú celé čísla, desatinné čísla a textové (znakové) reťazce.
- ✓ Znakový reťazec je postupnosť znakov a uzatvára sa medzi strojopisné apostrofy alebo úvodzovky.
- ✓ Keď chceme vykonávať operácie medzi dvoma hodnotami, musia byť rovnakého údajového typu, ak nie sú, musíme ich pretypovať.
- ✓ Knižnica **`math`** umožňuje používať matematické funkcie na sínus, kosínus, radiány a podobne.
- ✓ Knižnica **`random`** umožňuje používať funkcie na náhodné generovanie čísel alebo náhodný výber.

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem premenná.
2. Ako vzniká premenná v Pythone a ako v iných programovacích jazykoch?
3. Vysvetlite, čo znamená, že priraďovací znak nie je komutatívny a čo vďaka tomu môžeme a čo z tohto dôvodu nemôžeme robiť?
4. Zdôvodnite, prečo nebude fungovať tento príkaz, ak je ako prvý v programe: `a = a + 5`?
5. Ako sa ukladajú údaje do pamäte hodnôt a pamäte premenných a akým spôsobom vznikajú a zanikajú referencie? Vytvorte vhodný príklad a vysvetlite to na ňom.
6. Akým spôsobom funguje hromadné priraďovanie?
7. Ako môžeme a ako nesmieme pomenovať identifikátor premennej?
8. Ako má používateľ vedieť, čo (aký vstupný údaj) od neho počítač chce? Ako to má program používateľovi „oznámiť“?
9. Ako, naopak, má program používateľovi oznámiť to, čo v danej chvíli používateľ od programu vyžaduje?
10. Zdôvodnite, prečo sa niekedy môže stať, že 8-krát 15 nie je 120, ale 1515151515151515? Čo je potrebné spraviť, aby sme tomu predišli?
11. Akým spôsobom vypočítame druhú odmocninu z kosínusu sínusu 120 stupňov (teda $\sqrt{\cos(\sin 120\,°)}$)?
12. Ako predchádzajúci príklad vylepšiť tak, aby sa táto hodnota počítala vždy po jeho spustení z ľubovoľného uhla od 0 po 360 stupňov? Zdôvodnite všetko, čo ste pre to museli urobiť (naprogramovať), aby program fungoval.

5 Cyklus

Ciele

Cieľom kapitoly venovanej cyklom je, aby žiak precvičovaním rôznych zadaní získal schopnosť usúdiť, kedy a ako je potrebné v zadaní použiť cyklus, aby si uvedomil, ako sa dá využiť identifikátor cyklu, aby si osvojil funkcie generujúce postupnosť a uvedomil si, ako sa nimi takáto postupnosť špecifikuje. Žiak maturitného ročníka by mal byť schopný pomocou cyklu s vopred známym počtom opakovaní naprogramovať ľubovoľný sumatívny alebo multiplikatívny matematický vzorec (nechať vyčísliť jeho hodnotu na určenú presnosť) a tiež tvoriť obrazce a semigrafiku z cyklov (trojuholníky, grafy jednoduchých funkcií).

Doteraz sme vytvárali programy, kde sa príkazy vykonávali postupne jeden za druhým. Lenže pri programovaní reálnych programov budeme potrebovať, aby sme isté časti programov mohli vykonávať viackrát za sebou bez toho, aby sme to museli viackrát rozpísať.

5.1 Cyklus s vopred známym počtom opakovaní*

V Pythone poznáme **dva typy cyklov,**[38] predstavíme si prvý z nich, tzv. **for cyklus.** Táto programová konštrukcia má všeobecne takýto tvar:

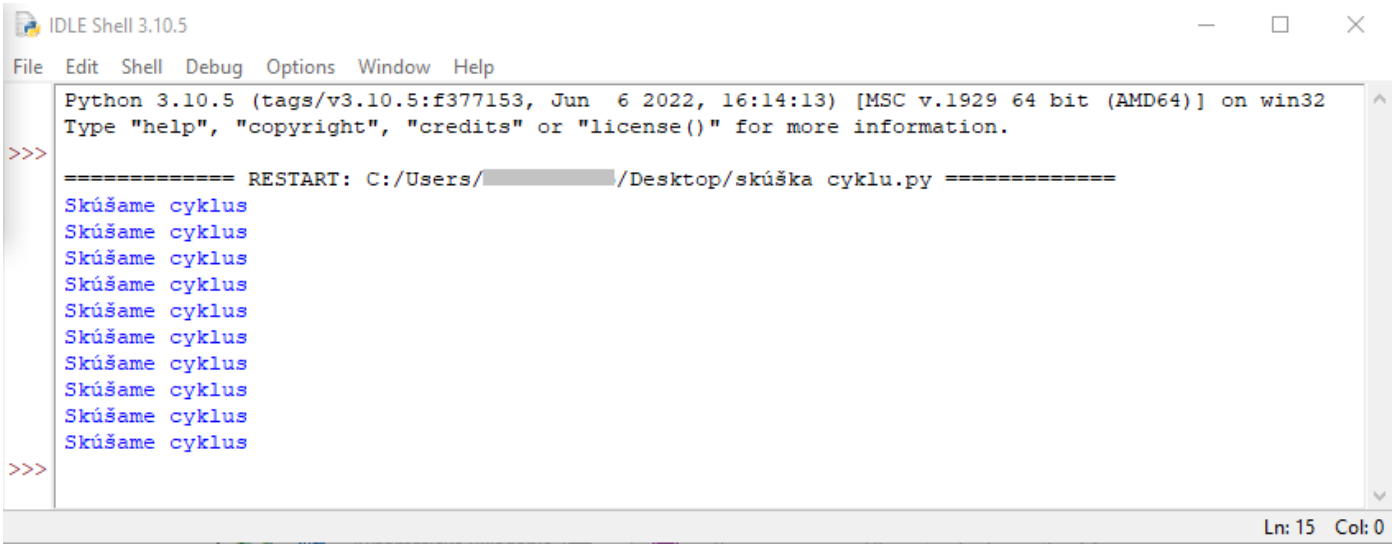
```
for premenná in postupnosť:
    príkaz_1
    príkaz_2
    ...
```

a **opakuje zadaný počet krát príkazy odsadeného bloku príkazov.** Samotný riadok konštrukcie `for` obsahuje meno premennej (**identifikátor cyklu**) a **postupnosť, ktorej dĺžka vyjadruje** rozsah (**počet opakovaní**).[39] Za ním nasleduje **dvojbodka** a do ďalších riadkov tela cyklu príkazy, ktoré musia byť **odsadené od kraja** o rovnaký počet medzier (odporúčame používať tabulátor, ktorý odsadí riadky o 4 medzery).

Poďme si to vyskúšať na veľmi jednoduchom príklade.

```
1.  for premenná in range(10):
2.      print("Skúšame cyklus")
```

Po spustení, ako môžeme vidieť, na shell desaťkrát vypíše „Skúšame cyklus“:

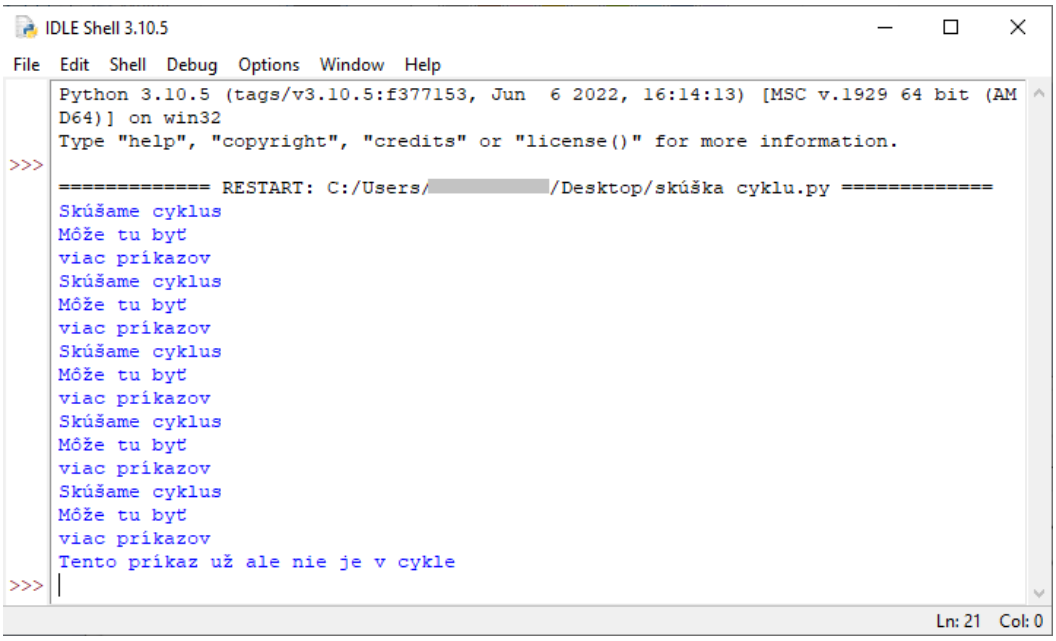


[38] *Po anglicky loop – slučka, niečo, čo sa opakuje.*

[39] *Na generovanie postupnosti hneď ako prvé použijeme funkciu* `range()`*, je to najčastejší spôsob, ako v Pythone zostaviť cyklus for.*

Cyklus môže mať, samozrejme, viac riadkov a **patrí mu len to, čo je odsadené na rovnakú úroveň.** Príkaz, ktorý je už odsadený na rovnakú úroveň ako riadok konštrukcie cyklu, sa **vykoná iba raz.** Vyskúšajme si to:

```
1.  for premenná in range(5):
2.      print("Skúšame cyklus")
3.      print("Môže tu byť")
4.      print("viac príkazov")
5.  print("Tento príkaz už ale nie je v cykle")
```



5.2 Identifikátor cyklu

V konštrukcii cyklu však máme „akúsi“ premennú, ideme zistiť, na čo všetko sa dá využiť a čo vlastne vyjadruje. V cykle sa jej hovorí **počítadlo,** niekedy aj **akumulátor, identifikátor** cyklu alebo aj **riadiaca premenná.** Python tejto premennej automaticky nastavuje hodnotu podľa toho, ktorý **člen postupnosti** práve nasleduje – v našom prípade, keďže sme použili **`range(n)`** – koľký raz sa už cyklus vykonal. Teda zápis:

```
1.  for premenná in range(n):
2.      príkazy
```

v skutočnosti znamená

```
1.  premenná = 0
2.  príkazy
3.  premenná = 1
4.  príkazy
5.  premenná = 2
6.  príkazy
7.  …
8.  premenná = n-1
9.  príkazy
```

Ako je to s tou postupnosťou, ktorá má byť v konštrukcii cyklu **for** a čo to znamená? Postupne si preberieme (zopakujeme), čo tie postupnosti vlastne v Pythone sú alebo čo nimi môže byť. Najčastejšie sa ako postupnosť v cykle používa **číselný rozsah, ktorý zabezpečuje funkcia `range(n)`.**[40] Ako už bolo naznačené, tento **`range(n)` mení hodnotu počítadla** podľa celočíselnej postupnosti $0, 1, 2, 3, \ldots, n-1$. My už sme takéto čosi mali, keď sme generovali náhodné čísla. Tam sme používali funkciu **`randrange()`** a uvádzali sme aj tabuľku prípustných hodnôt. Princíp je **úplne rovnaký aj tu,** akurát tu nejde o všetky prípustné náhodne vygenerované hodnoty, ale o **postupnosti celých čísel.** Teda **`range(n)`** je postupnosť celých čísel s jedným povinným a dvoma voliteľnými parametrami, ktorých funkcia je opísaná v kapitole zaoberajúcej sa náhodne generovanými prvkami, ale pripomeňme si ju pozmenenú a doplnenú:

[40] *Môžete si všimnúť, ako sme spomínali, že funkcia je tzv. vstavaná (built-in) – teda netreba nič importovať a je zafarbená nafialovo.*

- **prvý parameter (ak je iba jeden)** určuje hornú otvorenú hranicu maximálnej hodnoty postupnosti, pričom spodná hranica sa predpokladá automaticky nula (vrátane nej),
- **ak sú parametre dva alebo tri,** tak prvý parameter určuje dolnú hranicu minimálnej hodnoty postupnosti **vrátane** nej, druhý parameter potom určuje maximálnu hodnotu **okrem** nej.
- **tretí parameter** určuje **krok** – rozostup medzi hodnotami.

Príklady:

volanie funkcie	*generovaná postupnosť*
`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, 80, 90
`range(10, 1)`	prázdna postupnosť
`range(1, 1)`	prázdna postupnosť

Okrem toho tu funguje aj záporné krokovanie tretieho parametra. Napríklad:

`range(15, 5, -1)`	15, 14, 13, 12, 11, 10, 9, 8, 7, 6
`range(9, -1, -1)`	9, 8, 7, 6, 5, 4, 3, 2, 1, 0

Uvedomte si jednu zaujímavú a dôležitú vlastnosť – prvý parameter vždy obsahuje číslo, ktorým sa postupnosť začína[41] a druhý parameter číslo, ktorým sa postupnosť končí, ale okrem neho.[42] Ak by sme sa aj v budúcnosti stretli s podobnými funkciami vyžadujúcimi **rozsah** (interval), tak ten funguje **práve na takomto princípe.**

Spomeňme si, čo všetko ešte považujeme v programovaní za postupnosť. Je ňou **textový reťazec** (postupnosť znakov) a aj **usporiadaná n-tica** (tuple), prípadne **polia a zoznamy** (všetko postupnosti nejakých hodnôt). Do konštrukcie cyklu teda vieme zapísať aj tie.[43] Všetkým takýmto údajovým typom, ktoré „sú schopné" tvoriť postupnosť (a teda byť argumentom cyklu), hovoríme, že sú tzv. **iterovateľné.**[44] Iterovateľný údajový typ znamená taký, **po ktorého zložkách možno prechádzať cyklom.**

PRÍKLAD 9

Poďme si ukázať pár príkladov. Do všetkých cyklov dáme zatiaľ jediný príkaz, a tým bude `print("počítadlo má hodnotu", i)`,[45] ktorý bude vypisovať hodnotu počítadla. Vyskúšajte si sami vytvoriť niekoľko takýchto príkladov, spustiť cyklus a pozorovať hodnoty identifikátorov. Napríklad:

```
1.  for i in range(5):
2.      print("počítadlo má hodnotu", i)
3.
4.  for i in range(3, 45, 8):
5.      print("počítadlo má hodnotu", i)
6.
7.  for j in "abeceda":
8.      print("počítadlo má hodnotu", j)
9.
10. for názov_počítadla in ("red", "orange", "yellow", "green", "black", "blue"):
11.     print("počítadlo má hodnotu", názov_počítadla)
12.
13. for i in "red", "orange", "yellow", "green", "black", "blue":
14.     print("počítadlo má hodnotu", i)
15.
16. postupnosť = (1, 5, 7, 0, 8, 9, 44)
17. # postupnosť = 1, 5, 7, 0, 8, 9, 44      # dá sa aj takto bez zátvoriek
18.
```

[41] *Môžeme matematicky povedať, že range má spodnú hranicu intervalu* ***uzavretú.***
[42] *range má hornú hranicu intervalu* ***otvorenú.***
[43] *Postupnosti sa tu dajú písať aj bez zátvoriek, odporúčame ich tam však napriek tomu vždy dať.*
[44] *iterovateľné = „opakovateľné"; iterácia = opakovanie*
[45] *Vecou nepísaného pravidla je označovať identifikátory v cykle zvyčajne písmenami* `i`, `j` *alebo* `k`.

```
for i in postupnosť:
    print("počítadlo má hodnotu", i)

for k in 1, 5, 7, 0, 8, 9, 44:
    print("počítadlo má hodnotu", k)
```

Postupnosť môže v Pythone znamenať aj postupnosť rôznych údajových typov, nielen čísel alebo textových reťazcov. Vyskúšajme do programu dopísať a spustiť:

```

postupnosť_2 = 1, 5, "ahoj", (3, 5, 2), 'C', 3.95, postupnosť

for k in postupnosť_2:
    print("počítadlo má hodnotu", k)
```

5.3 Kreslenie pomocou cyklov*

Keďže sme si ukázali, ako sa zapisuje cyklus a vieme, ako fungujú počítadlá, poďme to využiť a pomocou cyklu kresliť.

PRÍKLAD 10

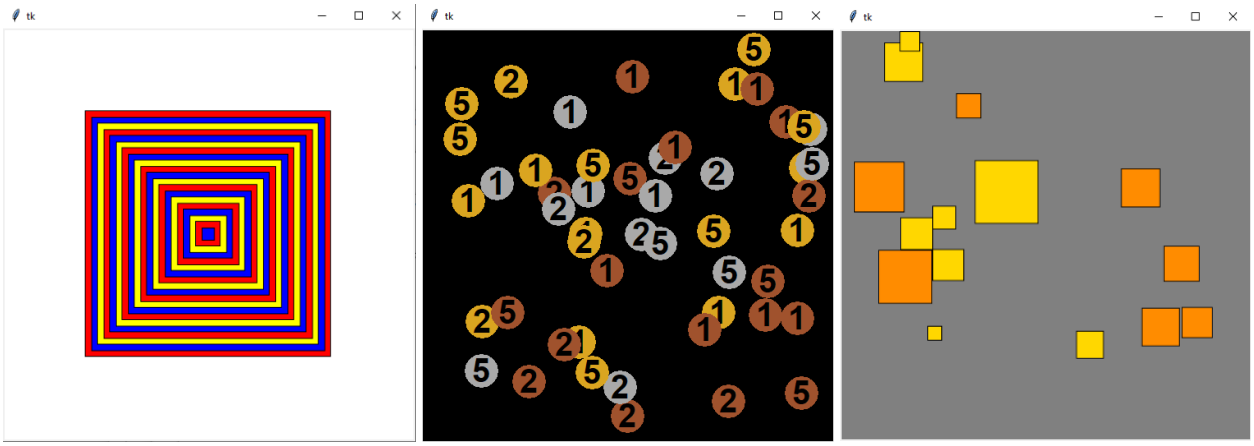
Nakreslime si takýto terč. Program si na začiatku vypýta od používateľa, koľko kruhov má nakresliť, pričom vonkajší kruh má vždy rovnakú veľkosť (na obrázku pre **`počet = 15`** a **`počet = 8`**). Na základe toho, čo už vieme, pokúste sa najskôr sami a potom sa pozrite na riešenie.

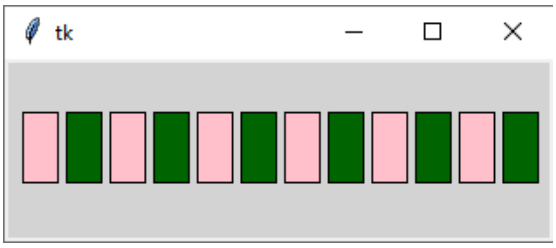
Riešenie:

```
import tkinter
canvas = tkinter.Canvas(width = 500, height = 500, bg = "white")
canvas.pack()

počet = int(input("Zadaj počet kruhov: "))
x1 = 100      # chceme terč 100 pixelov od všetkých okrajov
x2 = 400      # aj tu, keďže 100 od okraja = 500 - 100 = 400
f1 = "red"    # dve farby, ktoré sa budú striedať
f2 = "white"

for i in range(počet):
    canvas.create_oval(x1, x1, x2, x2, fill = f1)
    x1 = x1 + (150/počet) # hodnotu jedných súradníc (tých zhora) zväčšíme
    x2 = x2 - (150/počet) # hodnotu druhých (tých zdola) zmenšíme, aby bol
                          # nasledujúci kruh menší
    # Rozdeľujeme polomer kruhu (rozmer plátna 500 px, z každej strany plátna
    # uberáme 100 px, čiže ostane kruh s priemerom 300 px, polomerom
    # 150 px) na toľko častí, koľko zadal používateľ.
    f1, f2 = f2, f1   # hromadným priradením vymeníme farby
    canvas.update()   # keď chceme vidieť priebeh kreslenia
    canvas.after(500)
```

Úloha 10

Podobne nakreslite aj tento „hranatý terč“ a skúste vykresliť n mincí s nominálnymi hodnotami 1, 2 a 5 a farbami pripomínajúcimi zlatú, striebornú a medenú (to, ktorá minca má byť akej farby – napr. 1 = medená, 2 = strieborná, 5 = zlatá, riešiť nemusíte, tým sa budeme zaoberať neskôr), náhodne rozmiestnené štvorce alebo obdĺžniky za sebou, prípadne si vymyslite iné.

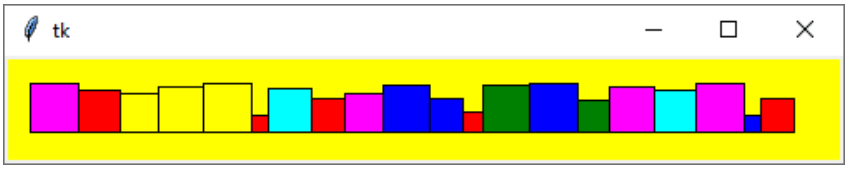




Príklad 11

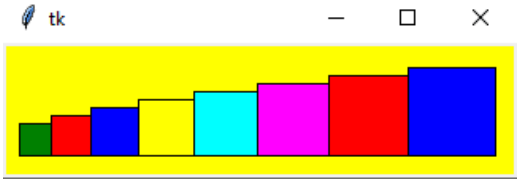
Riešenia nasledujúcich dvoch zadaní si ukážeme, avšak bez bližšieho komentára. Kód je intuitívny a určite na to prídete aj sami.

V prvom prípade vytvorme niekoľko (napríklad 20) štvorcov náhodnej veľkosti a farby a nechajme ich umiestňovať tesne za sebou.




```
1.  import tkinter
2.  import random
3.
4.  canvas = tkinter.Canvas(width = 500, height = 60, bg = "yellow")
5.  canvas.pack()
6.
7.  x, y = 15, 45
8.
9.  for i in range(20):
10.     veľkosť = random.randrange(10, 31)
11.     farba = random.choice(("red", "green", "blue", "cyan", "magenta", "yellow"))
12.     canvas.create_rectangle(x, y, x + veľkosť, y - veľkosť, fill = farba, width = 1)
13.     x = x + veľkosť
```

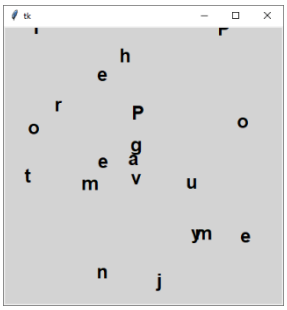
V druhom prípade vytvorme niekoľko štvorcov, ktorým necháme postupne narastať veľkosť vždy o rovnakú hodnotu.

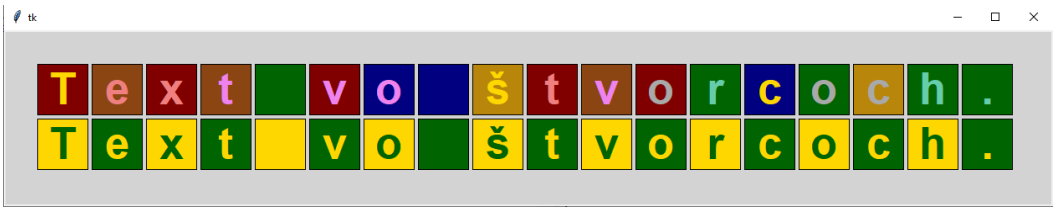


```
1.  import tkinter
2.  import random
3.
4.  canvas = tkinter.Canvas(height = 80, width = 320, bg = "yellow")
5.  canvas.pack()
6.
7.  x, y, a, b = 10, 70, 20, 5
8.
9.  for i in range(8):
10.     farba = random.choice(("red", "green", "blue", "cyan", "magenta", "yellow"))
11.     canvas.create_rectangle(x, y, x + a, y - a, fill = farba)
12.     x = x + a
13.     a = a + b
```

ÚLOHA 11

Nasledujúce dve úlohy vyriešte sami. Napíšte program, ktorý po zadaní vety používateľom jednotlivé písmenká rozhádže náhodne na plátno. Potom napíšte program, ktorý každé písmenko vety umiestni do náhodne farebného štvorca.





5.4 Vnorený cyklus

Niekedy potrebujeme vykonávať aj **opakovanie v rámci opakovania.** V tom nám nič nebráni, vieme totiž do tela cyklu vložiť ďalší cyklus – vznikne tým tzv. **vnorený cyklus** alebo cyklus v cykle. Vtedy už je vhodné použiť pre každý cyklus iný názov počítadla.[46]

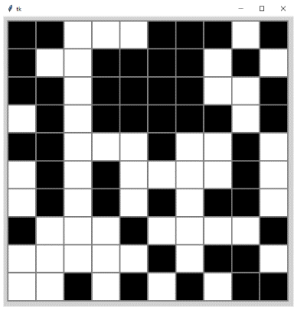
Vyskúšajme si na jednoduchom príklade, ako sa taký cyklus v cykle správa.

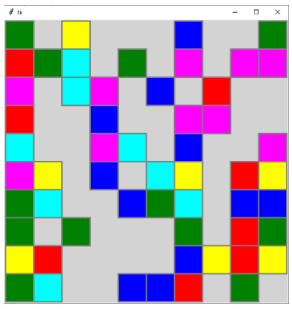
```
1.  for i in range(5):
2.      print("cyklus prvej úrovne", i)
3.      for j in range(3):
4.          print("    cyklus druhej úrovne", j)
```

Cyklus prvej úrovne vykoná prvé opakovanie, v rámci ktorého vnorený cyklus vykoná svoj *n* počet opakovaní. Potom cyklus prvej úrovne vykoná druhé opakovanie, v rámci ktorého vnorený cyklus vykoná opäť všetkých svojich *n* opakovaní.

PRÍKLAD 12

Pomocou takéhoto vnoreného cyklu a všetkého, čo sme sa doteraz učili, skúste najskôr sami vytvoriť nasledujúce šachovnice. Potom sa pozrite na riešenie oboch:





Riešenie prvej šachovnice:

```
1.  import tkinter
2.  import random
3.
4.  canvas = tkinter.Canvas(width = 620, height = 620, bg = "lightgrey")
5.  canvas.pack()
6.
```

[46] *Ale ak s počítadlom priamo nepracujeme, nie je to nevyhnutné, iba odporúčané.*

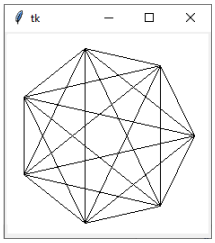
```
y = -50

for i in range(10):
    x = 10
    y = y + 60

    for i in range(10):
        farba = random.choice(("white", "black"))
        canvas.create_rectangle(x, y, x + 60, y + 60,
                                fill = farba, outline = "grey", width = 3)
        x = x + 60
```

Riešenie druhej šachovnice:

```
import tkinter
import random

canvas = tkinter.Canvas(width = 603, height = 603, bg = "lightgrey")
canvas.pack()

y = -57

for i in range(10):
    x = 0
    y = y + 60

    for i in range(10):
        farba = random.choice(("red", "green", "blue", "cyan", "magenta", "yellow"))
        a = random.choice((6, 12, 18))
        canvas.create_rectangle(x*a + 3, y, x*a + 60+3, y + 60,
                                fill=farba, outline="grey", width="3")
        x = x + 10
```

PRÍKLAD 13

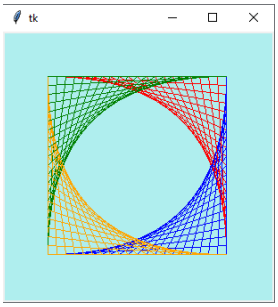
Spomeňte si na predchádzajúce úlohy – kreslenie n-uholníka. Ako vidíme, obrázok sa dá takto pekne vykresliť pomocou cyklu v cykle.




```
import tkinter
import math
canvas = tkinter.Canvas(width = 500, height = 500, bg = "white")
canvas.pack()

n = int(input("Zadaj počet uhlov: "))
k = 2 * math.pi / n

a, b = 0, k

for i in range(n):
    canvas.create_line(250 + math.cos(a)*100, 250 + math.sin(a)*100,
                       250 + math.cos(b)*100, 250 + math.sin(b)*100)
    a, b = b, b+k

    for i in range(n):
        canvas.create_line(250 + math.cos(i*k)*100, 250 + math.sin(i*k)*100,
                           250 + math.cos(b)*100,   250 + math.sin(b)*100)
```

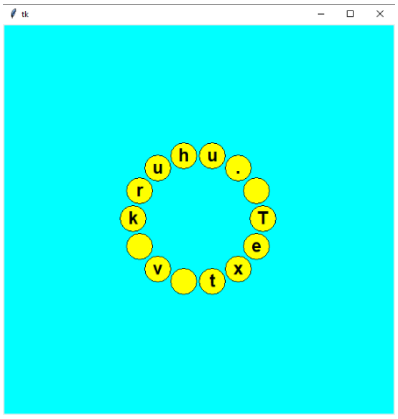
ÚLOHA 12

Vašou úlohou bolo nakresliť tento obrazec [3]. Už teraz, keď vieme pracovať s cyklami, by to malo byť jednoduché. (Zrejme na vykreslenie nebude treba vnorený cyklus.) Skúste tu vhodne použiť aj animáciu (**`canvas after()`**, **`canvas update()`**).



PRÍKLAD 14

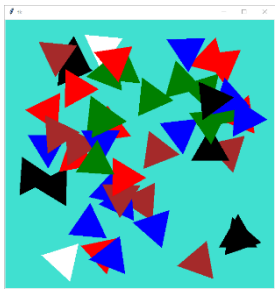
Riešenie programu, ktorý umiestňuje písmenká do kruhu, si ukážeme.




```
1.  import tkinter
2.  import math
3.
4.  slovo = input("Zadaj slovo: ")
5.  dĺžka = 0
6.
7.  for i in slovo: dĺžka = dĺžka + 1
8.
9.  canvas=tkinter.Canvas(width = 600, height = 600, bg = "cyan")
10. canvas.pack()
11.
12. x, y, b = 300, 300, 0
13. k = 2 * math.pi / dĺžka
14.
15. cos, sin = math.cos, math.sin
16.
17. for i in slovo:
18.     canvas.create_oval((300 + cos(b)*100) - 20,  (300 + sin(b)*100) - 20,
19.                        (300 + cos(b)*100) + 20,  (300 + sin(b)*100) + 20,
20.                        fill = "yellow")
21.     canvas.create_text(300 + cos(b)*100, 300 + sin(b)*100,
22.                        text = i, fill = "black", font = "Arial 20 bold")
23.     b = b + k
```

PRÍKLAD 15

Vyskúšajte takéto náhodné rozmiestnenie a náhodnú rotáciu rovnostranných trojuholníkov.



```
1.  import tkinter
2.  import math
3.  import random
4.
5.  canvas = tkinter.Canvas(width = 600, height = 600, bg = "turquoise")
6.  canvas.pack()
7.
8.  cos, sin, rad = math.cos, math.sin, math.radians
9.
10. a = int(input("Zadajte počet trojuholníkov: "))
11. b = int(input("Zadajte veľkosť strany: "))
12.
13. for i in range(a):
14.     x = random.randrange(b+5, 595-b)
15.     y = random.randrange(b+5, 600-b)
16.     u = random.randrange(360)
17.     farba = random.choice(("red", "green", "blue", "black", "white", "brown"))
18.     canvas.create_polygon(x + cos(rad(u+330))*b, y + sin(rad(u+330))*b,
19.                           x + cos(rad(u+90))*b,  y + sin(rad(u+90))*b,
20.                           x + cos(rad(u+210))*b, y + sin(rad(u+210))*b,
21.                           fill = farba)
```

5.5 Výpočty pomocou cyklov

Myslíte si, že oba cykly sa zopakujú rovnaký počet krát, alebo ten prvý skončí skôr? Bez toho, aby sme si to overili v Pythone, porozmýšľajte, aké hodnoty budú postupne nadobúdať **x** a **i**.

```
1.  for i in range(20):
2.      i = i + 5
3.      print("Čo je i?", i)
```

```
1.  x = 0
2.  for i in range(20):
3.      x = x + 5
4.      print("Čo je x?", x)
```

Asi ste zistili, že zatiaľ čo **x** postupuje po 5-násobkoch (**5**, **10**, **15**, **20**, ...), **i** sa začne číslom **5** a končí sa číslom **24**. Z toho vyplýva dôležitý fakt, že **identifikátor v každom novom opakovaní resetuje svoju prípadnú modifikáciu a pokračuje vo svojej postupnosti.** Môžeme ho použiť v cykle hocijako, vedzte však, že v novom cykle bude jeho hodnota hodnotou **ďalšieho člena postupnosti.** Teda, ak sme **x** v prvom cykle pripočítali **5**, v druhom zas **5**, je jeho hodnota **10**. Ak sme však **i** pripočítali v prvom cykle **5**, je jeho hodnota **0 + 5 = 5**, v druhom cykle však pokračuje hodnotou **1**, teda **1 + 5** bude **6**.

Cykly slúžia hlavne na programovanie rôznych sumatívnych a multiplikatívnych vzorcov (takých, ktoré vyžadujú *n* sčítaní alebo *n* násobení a ich hodnotu by bolo zdĺhavé počítať ručne).

PRÍKLAD 16

Zadajte dolný a horný interval a vypočítajte súčet čísel celočíselnej postupnosti.

```
1.  suma = 0
2.  a = int(input("Odkiaľ? "))
3.  b = int(input("Pokiaľ? "))
4.
5.  for i in range(a, b+1):
6.      suma = suma + i
7.
8.  print("Súčet je " + str(suma))
```

PRÍKLAD 17

Napíšte program na výpočet faktoriálu. Program sa bude líšiť od predchádzajúceho len minimálne. V čom?

```
1.  súčin = 1
2.  a = int(input("Zadajte číslo "))
3.
4.  for i in range(1, a+1):
5.      súčin = súčin * i
6.
7.  print("Faktoriál je " + str(súčin))
```

PRÍKLAD 18

Napíšte program, ktorý zistí ciferný súčet zadaného čísla. Všimnite si, že využívame to, že textový reťazec je postupnosť znakov.

```
1.  cif_súčet = 0
2.  a = input("Zadajte číslo ")
3.
4.  for i in a:
5.      cif_súčet = cif_súčet + int(i)
6.
7.  print("Ciferný súčet je " + str(cif_súčet))
```

PRÍKLAD 19

Napíšte program, ktorému zadáte, koľko máte známok, následne známky postupne pozadávate a program vám vypíše ich priemer.

```
1.  n = int(input("Zadajte počet známok "))
2.  súčet = 0
3.  for i in range(1, n+1):
4.      známka = int(input("Zadajte známku č. " + str(i) + ": "))
5.      súčet = súčet + známka
6.  print("Priemer je " + str(súčet / n))
```

PRÍKLAD 20

Keďže sme spomínali, že vieme naprogramovať aj sumatívne a multiplikatívne vzorce, tak takéto vzorce na vyčíslenie Ludolfovho čísla π sú hneď dva – Wallisov a Leibnizov a vyzerajú takto (Wallisov je prvý [5], Leibnizov je druhý [6]):

$$\frac{\pi}{2} = \prod_{n=1}^{\infty} \frac{4n^2}{4n^2-1}, \qquad \frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Ako však toto ako informatici vieme zadať počítaču? Aby to bolo jasnejšie, „prizveme si odborníka“[47] – povedzme, že tým zápisom vôbec nerozumieme (ani ako informatici nemusíme). Odborník (teda matematik) povie: tak Wallisov vzorec sa dá rozpísať takto:

$$\frac{\pi}{2} = \prod_{n=1}^{\infty} \frac{4n^2}{4n^2-1} = \frac{4\cdot\mathbf{1}^2}{4\cdot\mathbf{1}^2-1}\cdot\frac{4\cdot\mathbf{2}^2}{4\cdot\mathbf{2}^2-1}\cdot\frac{4\cdot\mathbf{3}^2}{4\cdot\mathbf{3}^2-1}\cdot\ldots\cdot\frac{4\cdot\mathbf{5000}^2}{4\cdot\mathbf{5000}^2-1}\cdot\ldots$$

a Leibnizov vzorec vlastne znamená toto:

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{(-1)^{\mathbf{0}}}{2\cdot\mathbf{0}+1} + \frac{(-1)^{\mathbf{1}}}{2\cdot\mathbf{1}+1} + \frac{(-1)^{\mathbf{2}}}{2\cdot\mathbf{2}+1} + \cdots + \frac{(-1)^{\mathbf{5000}}}{2\cdot\mathbf{5000}+1} + \cdots$$

a z takto vytvorenej postupnosti nám ako informatikom by malo byť v tejto chvíli už jasné, ako to naprogramujeme, napríklad takto:

```
1.  # Wallisov vzorec
2.  číslo = 4/3
3.  for n in range(2, 1000001):
4.      číslo = číslo * ( (4*(n**2)) / ((4*(n**2)) - 1) )
5.  print(číslo*2)
```

```
1.  # Leibnizov vzorec
2.  číslo = 0
3.  for n in range(1000000):
4.      číslo = číslo + ( ((-1)**n) / ((2*n) + 1) )
5.  print(číslo*4)
```

Ako sme pri tom rozmýšľali? Pri takýchto vzorcoch je potrebné si uvedomiť, ako zadefinovať premennú, ktorá bude akumulátorom (do ktorej sa bude pripočítavať alebo „prinásobovať“) a, samozrejme, pred cyklom nastaviť počiatočnú hodnotu.

Pri Leibnizovom vzorci je to jednoduché – počiatočnú hodnotu nastavíme na nulu, pretože pripočítavame. Potom v cykle s miliónom opakovaní (čo je pre nás dostatočný počet na to, aby sme nasimulovali nekonečno) v uzavretom intervale od 0 do 999 999 jednoducho premennú vždy navýšime o nasledujúci člen postupnosti. Na záver vypíšeme hodnotu štvornásobku, keďže vzorec platí pre $\frac{\pi}{4}$.

[47] *Ak dostane programátor úlohu z inej vednej oblasti, ktorej nerozumie, väčšinou ju dostane od niekoho, kto je v danej oblasti odborník – je preto namieste vždy sa opýtať a poradiť, ako zadanie bolo myslené alebo aké sú jeho špecifiká.*

Pri Wallisovom vzorci tiež v cykle s miliónom opakovaní (aby sme na záver vedeli porovnať, ktorý vzorec je presnejší) už ale nemôžeme počiatočnú hodnotu premennej **číslo** nastaviť na nulu, pretože by sa nám vynuloval každý jeden medzivýpočet. Tu je potrebné ručne vyčísliť prvý člen, teda $\frac{4\cdot 1^2}{4\cdot 1^2-1}=\frac{4}{3}$. Keďže postupnosť sa začala pre $n=1$, čo už ale vyčíslené máme, cyklus sa začne pre $n=2$ a zopakujeme ho ešte 999 999-krát (od hodnoty 2 do 1 000 000). Na záver vypíšeme hodnotu dvojnásobku, keďže vzorec platí pre $\frac{\pi}{2}$.

Po jednom miliónе opakovaní (všimnite si, za aký relatívne krátky čas to počítač dokázal vypočítať a vypísať na shell a koľko by nám také niečo asi trvalo vypočítať na kalkulačke) môžeme porovnať hodnoty a vidíme, že číslo π to vypočítalo v oboch prípadoch na 5 desatinných miest správne. Skúste si vygoogliť jeho presnú hodnotu a zvýšiť počet opakovaní napríklad na dva milióny a uvidíte, ako mu narastá presnosť.

Ďalšie náročnejšie úlohy na cykly a podmienky budeme precvičovať po tom, ako si v ďalšej kapitole povieme niečo o algoritmizácii a zásadách správneho programovania, ktoré v úlohách využijeme.

5.6 Semigrafika pomocou cyklov

Už sme spomenuli aj operácie s textovými reťazcami – sčítanie a násobenie. Pripomeňme si ich:

+	zreťazenie (spojenie dvoch reťazcov)	`'a' + 'b'` má hodnotu `'ab'`
*****	viacnásobné zreťazenie reťazca	`3 * 'x'` má hodnotu `'xxx'`

Poďme to, že už poznáme cykly a to, že Python umožňuje viacnásobné reťazenie reťazca, využiť v semigrafike[48] a vykresliť nasledujúce texty na shell.

PRÍKLAD 21

Nechajte na shell vypisovať takýto číselný trojuholník:

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
```

Riešenie: asi riadky budú znamenať cykly a čísla v riadkoch vnorené cykly. Zamyslite sa nad tým, ako sme použili premenné na základe toho, čo sme si už povedali:

```
1.  a = int(input("Počet riadkov: "))
2.  for i in range(a):
3.      for m in range(i + 1):
4.          print(m, end = " ")
5.      print()
```

ÚLOHA 13

Takýto trojuholník skúste sami. Poradíme vám akurát, že tam bude vhodné zadefinovať si premennú, s ktorou budete potrebovať pracovať, ešte pred cyklom.

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28
```

PRÍKLAD 22

```
11                *                *
101              ***              ***
1001            *****            *****
10001          *******          *******
100001        *********        *********
1000001      ***********      ***********
10000001    *************    *************
100000001  ***************  ***************
```

[48] *Semigrafika je vytváranie obrazcov iba pomocou znakov bez nevyhnutnosti implementovať grafické rozhranie.*

Riešenie:

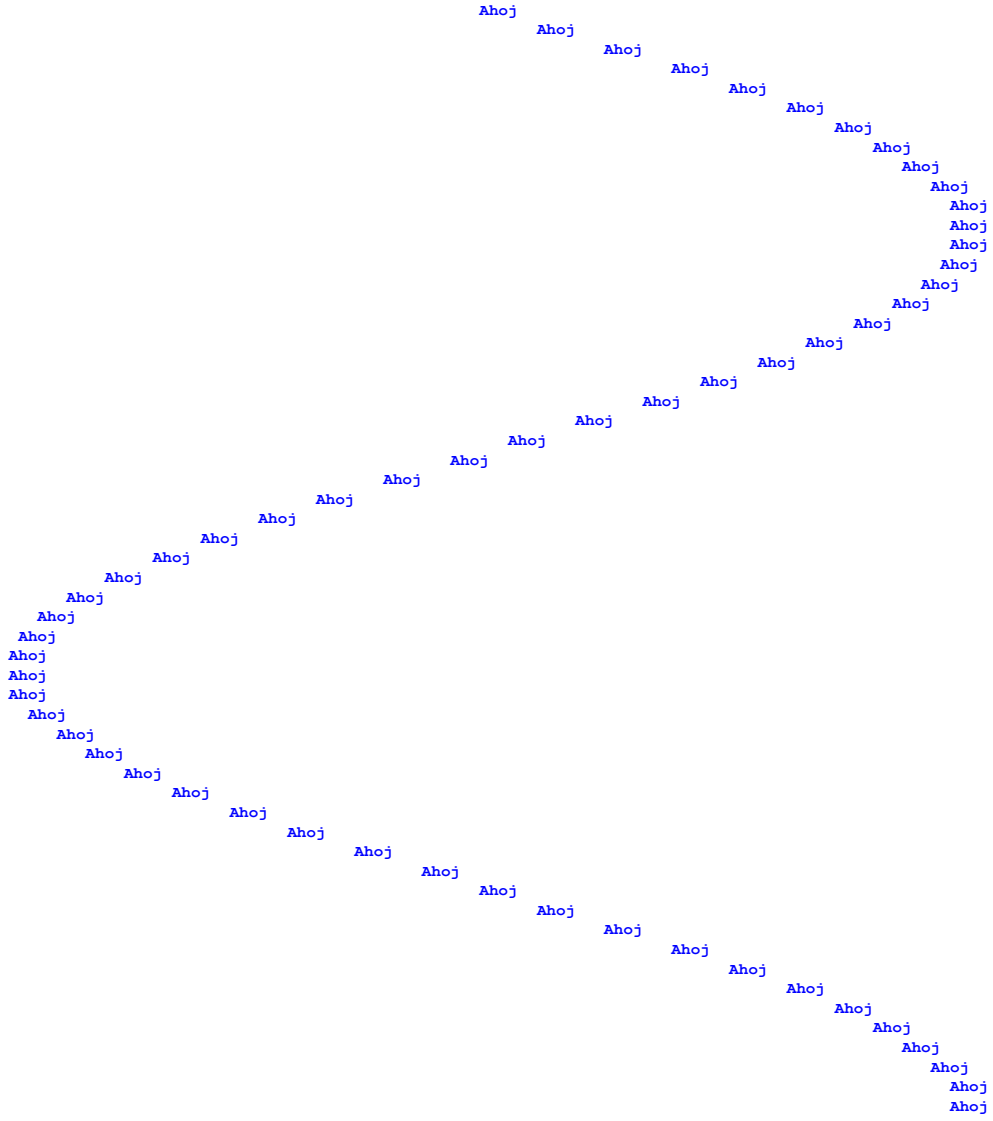
```
1.  a = int(input("Počet riadkov: "))
2.  for i in range(a):
3.      print(1, end = "")
4.      for m in range(i):
5.          print(0, end = "")
6.      print(1, end = "")
7.      print()
```

```
1.  a = int(input("Počet riadkov: "))
2.
3.  for i in range(a):
4.      print((a - i) * " ", end = "")
5.      print((2*i + 1) * "*", end = "")
6.      print()
```

```
1.  a = int(input("Počet riadkov: "))
2.
3.  for i in range(1, a+1):
4.      print(i * "*", end = "")
5.      print(2*(a-i) * " ", end = "")
6.      print(i * "*", end = "")
7.      print()
```

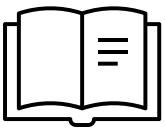
PRÍKLAD 23

Pomocou viacnásobného reťazenia reťazca nechajte semigraficky vypísať sínusoidu na shell.



Riešenie:

```
1.  import math
2.  x = 0
3.  a = input("Zadaj slovo: ")
4.  b = int(input("Zadaj dĺžku sínusoidy (napr. 500): "))
5.  c = int(input("Zadaj šírku sínusoidy (napr. 10): "))
6.
7.  for i in range(b):
8.      u = int(math.sin(math.radians(x)) * 50) + 50
9.      print(u * " ", a)
10.     x = x + c
```



ZHRNUTIE

✓ Cyklus s vopred známym počtom opakovaní je programová konštrukcia, ktorá opakuje zadaný počet krát nejaké príkazy.

✓ Riadok konštrukcie cyklu obsahuje rezervované slovo **for**, meno počítadla cyklu, rezervované slovo **in**, postupnosť, ktorej dĺžka vyjadruje počet opakovaní a dvojbodku.

- ✓ Cyklu patria len príkazy odsadené na rovnakú úroveň od riadka konštrukcie.
- ✓ Počítadlo cyklu nadobúda v každom jednom opakovaní vždy hodnotu nasledujúceho člena postupnosti – každého iterovateľného údajového typu.
- ✓ Na generovanie celočíselných postupností sa najčastejšie v cykle využíva funkcia **range()**.
- ✓ Funkcii **range()** vieme dať jeden až tri parametre a podľa toho vieme nastaviť spodnú, hornú hranicu intervalu a krokovanie.
- ✓ Iterovateľný údajový typ je taký, po ktorého zložkách možno prechádzať cyklom.
- ✓ Do tela cyklu sa dá vložiť ďalší cyklus a vytvoriť tak vnorený cyklus.
- ✓ Identifikátor v každom novom opakovaní resetuje svoju prípadnú modifikáciu a pokračuje vo svojej postupnosti – jeho hodnota bude hodnotou ďalšieho člena postupnosti.

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem cyklus s vopred známym počtom opakovaní.
2. Vymenujte časti, z ktorých sa skladá konštrukcia tohto cyklu.
3. Čo je to iterovateľný údajový typ?
4. Vymenujte všetky iterovateľné údajové typy, s ktorými ste sa už stretli.
5. Akú funkciu používame na generovanie postupnosti?
6. Aké parametre dáte tejto funkcii, ak chcete klesajúcu postupnosť párnych čísel od 50 do 10 vrátane?
7. Ako zapíšeme opakovanie v rámci opakovania?
8. Určte bez počítača, koľko opakovaní sa spolu vykoná, ak hlavný cyklus má postupnosť **range(4, 50, 6)**, vnorený cyklus má postupnosť **"ProGra_m0vAniE"** a jemu vnorený cyklus má postupnosť **"range(16)"**.
9. Podrobne opíšte, ako sa správa identifikátor v tele cyklu.

6 Podmienka

Ciele

Cieľom tejto kapitoly je, aby sa žiak oboznámil so spôsobmi vetvenia v programovaní, osvojil si správne formulovanie logických a relačných výrazov, správne zostavenie cyklov, aby vedel usúdiť, kedy je vhodné použiť úplné, neúplné, viacnásobné a zložené vetvenie a tiež cyklus s počtom opakovaní ohraničeným podmienkou.

Pri programovaní často riešime situácie, keď sa program má na základe podmienky rozhodnúť medzi viacerými možnosťami.

6.1 Jednoduché vetvenie*

Už vieme vykresľovať ľubovoľný počet útvarov na náhodné pozície. Vykreslime si náhodne 500 červených krúžkov na plátno. Čo však spravíme, ak chceme, aby krúžky, ktoré sa vykreslia naľavo, boli inej farby ako tie, ktoré sa vykreslia napravo?

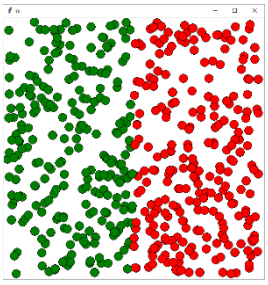
Na to slúži ďalšia programová konštrukcia, ktorá sa volá **podmienka** (resp. vetvenie) a vyzerá takto:

```
1.  if logický_výraz:
2.      príkaz_1
3.      príkaz_2
4.  else:
5.      príkaz_3
6.      príkaz_4
```

Ako sa môžeme dovtípiť: **ak**[49] platí nejaký logický výraz, program vykoná príkaz 1 a príkaz 2, ak neplatí, vykoná príkazy 3 a 4. Tak, ako v cykloch, aj tu je potrebné príkazy **odsadiť na rovnakú úroveň.** Poďme si to vyskúšať. Takejto konštrukcii hovoríme **jednoduché vetvenie** alebo **podmienený príkaz úplný.**

Príklad 24

Vykreslime si náhodne 500 červených krúžkov na plátno. Krúžky, ktoré sa vykreslia naľavo, budú zelené a tie, ktoré sa vykreslia napravo, budú červené.



```
1.  import tkinter
2.  import random
3.
4.  canvas = tkinter.Canvas(width = 600, height = 600, bg = "white")
5.  canvas.pack()
6.
7.  r = 10 # polomer
8.
9.  for i in range(500):
10.     x = random.randrange(10, 591)
11.     y = random.randrange(10, 591)
12.
13.     if x < 300:
14.         farba = "green"
15.     else:
16.         farba = "red"
17.
18.     canvas.create_oval(x - r, y - r, x + r, y + r, fill = farba)
```

[49] *Dovolíme si upozorniť na časté nedbanlivé zamieňanie si spojok **ak** a **keď** (angl. **if** a **when**). **Ak** vyjadruje podmienku – čo sa udeje za okolnosti, ktorá ale **nemusí nastať. Keď** vyjadruje podmienku – čo sa udeje za okolnosti, o ktorej **s určitosťou vieme, že nastane.** Napríklad: **ak** dostanem jednotku, pôjdem von (neviem, či dostanem jednotku); **keď** vyrastiem, budem lekárom (každé dieťa raz vyrastie). **V logike – vo výrokoch sa teda používa výhradne spojka „ak“.***

6.2 Viacnásobné a zložené vetvenie*

Povedzme však, že chceme, aby krúžky, ktoré sa vykreslia v dolnej polovici plátna, boli modré. Existuje aj tzv. viacnásobné vetvenie a zapisuje sa takto:

```
1.  if logický_výraz:
2.      príkaz_1
3.      príkaz_2
4.  elif iný_logický_výraz:
5.      príkaz_3
6.      príkaz_4
7.  else:
8.      príkaz_5
9.      príkaz_6
```

Znamená to, že sa vykonáva vždy tá skupinka príkazov (takzvaná **vetva**), ktorej logický výraz je pravdivý. Ak nie je pravdivý žiadny logický výraz, vykoná sa vetva `else`.

Pozor! **Vo viacnásobnom vetvení sa vykoná vždy len jedna vetva – prvá, ktorej logický výraz je pravdivý.** Aj keby výraz ktorejkoľvek ďalšej vetvy bol pravdivý, už sa nevykoná. Na to slúži neúplné vetvenie (ekvivalentným termínom **podmienený príkaz neúplný**), ktoré si ukážeme o chvíľu.

Mohli by sme program zapísať napríklad takto?

```
1.  if y > 300:
2.      farba = "blue"
3.  elif x < 300:
4.      farba = "green"
5.  else:
6.      farba = "red"
```

Očividne to funguje (program nevyhlási chybu a vykreslí presne to, čo sme chceli), pretože ako sme spomínali, viacnásobné vetvenie vykoná vždy len jednu z vetiev (prvú s pravdivým logickým výrazom) bez ohľadu na to, či tie nasledujúce majú tiež pravdivý logický výraz. **Napriek tomu sa to takto zapisovať nesmie. Logické výrazy vo viacnásobných vetvách je treba totiž konštruovať tak, aby výrazy jednotlivých vetiev boli navzájom disjunktné**[50] – konkrétne naša podmienka hovorí:

- ak je y väčšie ako 300, nastav modrú farbu,
- inak: ak je x menšie ako 300, nastav zelenú
- a inak nastav červenú.

Keby bol cyklus komplexnejší, programátora to zmätie a navyše sa môže poľahky stať, že sa počas vykonávania programu vyskytne nepredvídateľná udalosť a program nebude fungovať tak, ako sme si predstavovali. Čo sa má v tomto prípade vykresliť? **Máme krúžok v ľavom dolnom rohu. Má byť modrý alebo zelený? Veď predsa jeho súradnice spĺňajú obe podmienky** – vodorovná menšia ako 300 a zvislá väčšia ako 300. Keby ste sa ako programátor pozreli na takúto vetvenú podmienku, ktorá má 10 vetiev, asi by ste na to prišli ťažko.

Áno, viacnásobné vetvenie sa tu použiť dá, avšak tie podmienky treba formulovať inak. Na to potrebujeme vedieť, čo sú logické výrazy a ako fungujú.

PRÍKLAD 25

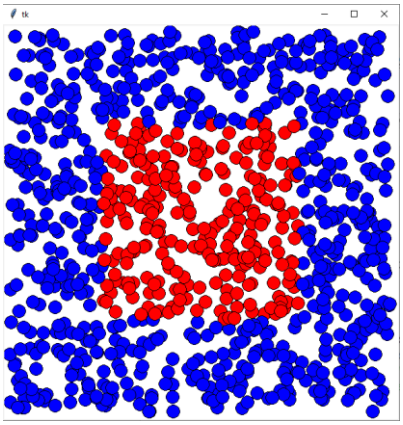
Vedeli by ste formulovať podmienku inak – správne? Bez toho, aby ste využili poznatky z nasledujúcej kapitoly? (Rada: skúste použiť tzv. **zložené vetvenie** (tiež nazývaný aj **podmienený príkaz vnorený** – zapisuje sa podobne ako vnorený cyklus.) Takýto zápis je už správny a navyše prehľadnejší – veď posúďte sami:

```
1.  if y < 300:
2.      if x < 300:
3.          farba = "blue"
4.      else:
5.          farba = "red"
6.  else:
7.      farba = "blue"
```

[50] *Teda, aby ich prienik bola prázdna množina.*

PRÍKLAD 26

Skôr, než si niečo povieme o logických výrazoch, skúste sami takto „správne“[51] definovať podmienku pre tento obrazec:



Veríme, že ste prišli na niečo takéto „rozvetvené“:

```
1.  if x > 150:
2.      if y < 450:
3.          if y > 150:
4.              if x < 450:
5.                  farba = "red"
6.              else:
7.                  farba = "blue"
8.          else:
9.              farba = "blue"
10.     else:
11.         farba = "blue"
12. else:
13.     farba = "blue"
```

6.3 Logické výrazy*

Výraz je zápis, ktorý sa skladá z operátorov a operandov a jeho výsledkom je vždy nejaká hodnota.

Operátory sú znamienka[52] a operandy sú hodnoty (čísla, textové reťazce, … a/alebo premenné, ktoré ich zastupujú). Napríklad, vo výraze **`a + b * 5`**:

- operátory sú **`+`**, **`*`**
- operandy sú **`a`**, **`b`**, **`5`**

Operátorov poznáme niekoľko druhov a podľa nich delíme výrazy:

- **aritmetické** (ich hodnota je číslo), ak nie je uprednostnený zátvorkami a operátory sú si rovné, počíta sa vždy **zľava doprava**
  - môže obsahovať:
    - **premennú**
    - **okrúhlu zátvorku**
    - **funkciu** (**`math.sin()`**, **`math.cos()`**, …)
    - **aritmetický operátor** (**`+`**, **`-`**, **`*`**, **`/`**, **`//`**, **`%`**, **`**`**)
  - prednosti operátorov:
    1. **umocňovanie** (najväčšia prednosť)
    2. **multiplikatívne – násobenie, delenie** (majú prednosť pred aditívnymi, ale sú si rovné)
    3. **aditívne** (**`+`**, **`-`**)

- **relačné** – obsahuje **relačné operátory**, ktoré sa v Pythone zapisujú takto:

`<`	menšie než
`>`	väčšie než
`==`	rovné
`>=`	väčšie alebo rovné
`<=`	menšie alebo rovné
`!=`	nerovné

  - POZOR! **`=` je priraďovací príkaz a `==` je relačný operátor, ktorý zisťuje rovnosť**
  - môže obsahovať aj operátory aritmetického výrazu
  - operátory relačného výrazu majú **vždy prednosť** pred operátormi aritmetického výrazu
  - napr. **`(a+b) * math.sin(x) > b + c`**
  - výsledkom je pravdivostná hodnota – **`True`** alebo **`False`**

[51] *„Správne“ na základe toho, čo sme si doteraz spomenuli; ten „naozaj správny“ spôsob si ukážeme o chvíľu.*
[52] *Nie vždy. Operátory sú aj logické spojky, čo nie sú znamienka, no na úvod a na jasnú predstavivosť je to vhodné pomenovanie.*

- **logické** – obsahuje všetko, čo relačný výraz + **logické operátory:**
    1. `not` (negácia) – má prednosť pred `and` aj pred `or`
    2. `and` (konjunkcia – a zároveň) – má prednosť pred `or`
    3. `or` (disjunkcia – alebo)
  - výsledkom je pravdivostná hodnota – `True` alebo `False`
  - pozor na prednosť – napríklad:

A	B	not A	not A and B	not(A and B)
False	False	True	False	True
False	True	True	True	True
True	False	False	False	True
True	True	False	False	False

Ak si nie ste istí prioritou operácií, **odporúčame používať okrúhle zátvorky vždy** – nič tým nepokazíte, práve naopak – sprehľadníte tým výrazy nielen pre seba, ale aj pre ďalších programátorov, ktorí budú kód po vás čítať.

Podľa toho, koľko operandov potrebuje operátor, aby vykonal operáciu,[53] delíme operátory na:

- **nulárne** – konštanty a premenné
- **unárne** – operátor `not` – do operácie vstupuje jeden operand, ktorému operátor `not` zneguje hodnotu, ďalej sú to zátvorky a znamienko pred hodnotou
- **binárne** – všetky známe (**+**, **-**, *****, **/**, **<**, **>** atď.) – definujú vzťahy **medzi dvoma operandmi** a vyjadrujú ich výslednú hodnotu
- **ternárne** – s nimi sa teraz nestretneme, používajú sa v mnohých programovacích jazykoch (aj v Pythone) a slúžia na jednoduchšie zápisy istých príkazov, **pozor však,** napríklad operácia **`a * b * c` nie je ternárna,** je to binárna operácia medzi **`(a * b) * c`** alebo **`a * (b * c)`**

Asi ste si všimli, že výsledkom relačných a logických výrazov je pravda/nepravda, resp. `True` a `False`. Je to pre nás nový údajový typ **`bool`** (z angl. Boolean), ktorý nadobúda **len tieto dve pravdivostné hodnoty.** Pripomeňme, že sme sa už bližšie zoznámili s údajovými typmi **`int`**, **`float`** a **`str`**. Neskôr si predstavíme ešte ďalšie.

PRÍKLAD 27

Napadne vám už teraz, ako sa dá konštrukcia z predchádzajúceho príkladu zjednodušiť?

Riešenie:

```
1.  if x > 150 and y < 450 and y > 150 and x < 450:
2.      farba = "red"
3.  else:
4.      farba = "blue"
```

Ešte lepší je tento zápis:

```
5.  if 150 < x < 450 and 150 < y < 450:
6.      farba = "red"
7.  else:
8.      farba = "blue"
```

Už by sme teraz mohli vedieť opraviť podmienky vo viacnásobnom vetvení z predchádzajúceho príkladu:

```
1.  if y > 300:
2.      farba = "blue"
3.  elif x < 300
4.      farba = "green"
5.  else:
6.      farba = "red"
```

Keďže už poznáme logické spojky a vieme konštruovať logické a relačné výrazy a vieme, že vo viacnásobnom vetvení by mali byť podmienky zapísané disjunktne, vieme toto vetvenie opraviť takto:

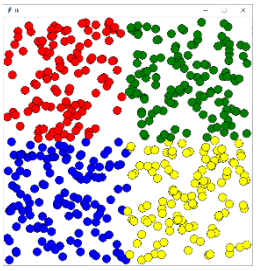
[53] *Takémuto deleniu sa hovorí aj **arita** operátorov.*

```
if y > 300:
    farba = "blue"
elif x < 300 and not(y > 300):
    farba = "green"
else:
    farba = "red"
```

Napriek tomu, že teraz je to už programátorsky správne, odporúčame v tomto prípade z dôvodu väčšej prehľadnosti použiť vnorenú (zloženú) podmienku. Viacnásobné vetvenie tu veľmi nemá opodstatnenie, budeme ho používať pri iných typoch úloh.

```
if y < 300:
    if x < 300:
        farba = "green"
    else:
        farba = "red"
else:
    farba = "blue"
```

PRÍKLAD 28



Nasledujúci príklad skúste najskôr sami. My ukážeme jeho riešenie pomocou zloženého vetvenia aj pomocou viacnásobného vetvenia.

Pomocou zloženého vetvenia:

```
if y < 300:
    if x < 300:
        farba = "red"
    else:
        farba = "green"
else:
    if x < 300:
        farba = "blue"
    else:
        farba = "yellow"
```

Pomocou viacnásobného vetvenia:

```
    if y < 300 and x < 300:
        farba = "red"
    elif y < 300 and not(x < 300):
        farba = "green"
    elif not(y < 300) and x < 300:
        farba = "blue"
    else:
        farba = "yellow"
```

Teraz vidíme, že určite vyzerá zložené vetvenie prehľadnejšie ako viacnásobné vetvenie, ktoré by z hľadiska správnosti malo vyzerať tak, ako ho máme skonštruované my.[54]

Kde je teda ideálne použiť viacnásobné vetvenie?

PRÍKLAD 29

Napíšte program, ktorý nechá vygenerovať n hodov kockou. Zistí, koľkokrát padla jednotka, dvojka, ... šestka a tieto počty vypíše na shell.

Tu je vhodné použiť viacnásobné vetvenie práve preto, lebo logické výrazy v jednotlivých vetvách (teda podmienky) sú **disjunktné** (ak padne jednotka, už nemôže padnúť nič iné, naproti tomu, ak sa krúžok nachádza naľavo, ešte neznamená, že sa súčasne nemôže nachádzať aj dole). **Pamätajte, že ak podmienky disjunktné nie sú, je vhodné použiť zložené vetvenie.**

```
import random

s = t = u = v = w = x = 0
počet = int(input("Počet hodov: "))
```

[54] *Namiesto* `not` *by sa dalo použiť znamienko* `<=` *alebo* `>=`*, chceli sme tu len poukázať na to, že je potrebné „ošetriť" – zahrnúť celý interval a tento zápis je všeobecný.*

```
5.
6.  for i in range(počet):
7.      a = random.randrange(1, 7)
8.
9.      if   a == 6: s = s + 1
10.     elif a == 5: t = t + 1
11.     elif a == 4: u = u + 1
12.     elif a == 3: v = v + 1
13.     elif a == 2: w = w + 1
14.     else       : x = x + 1
15.
16. print("Počet šestiek:"   , s)
17. print("Počet pätiek:"    , t)
18. print("Počet štvoriek:"  , u)
19. print("Počet trojok:"    , v)
20. print("Počet dvojok:"    , w)
21. print("Počet jednotiek:" , x)
```

Asi ste si už všimli, že používame dva zápisy na výpis hodnôt na shell:

- `print("Výpis hodnoty:", premenná)`
- `print("Výpis hodnoty: " + str(premenná))`

Ide o rovnocenné zápisy, pričom v prvom prípade sa informačný výpis (to, čo je v úvodzovkách) od premennej oddeľuje čiarkou, vtedy sa premenná vypíše v takom type, v akom je (ak je to integer, vypíše sa ako integer, ak je to string, vypíše sa ako string), pričom výpis je od premennej automaticky oddelený medzerou. V druhom prípade ide o už známe spájanie textového reťazca. Z toho vyplýva, že ak chceme mať za výpisom medzeru, musí byť aj v úvodzovkách a premenná, ktorú vypisujeme (ak nie je typu string), musí byť pretypovaná na string. Neskôr sa naučíme ešte tretí zápis (tzv. **f-string**), ktorého existenciu za istých okolností určite oceníte.

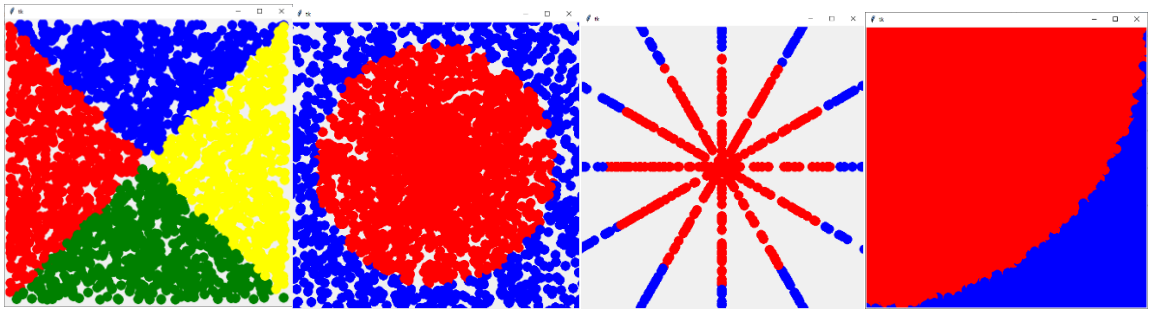
Na zmenu (navyšovanie, znižovanie) hodnôt sa nielen v Pythone používajú skrátené zápisy. Ich prehľad ponúka nasledujúca tabuľka:[55]

tento príkaz	*vykoná to isté ako tento príkaz*
`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`
`meno_premennej **= hodnota`	`meno_premennej = meno_premennej ** hodnota`

Tiež ste si asi všimli, že ak máme podmienku (alebo aj cyklus), v ktorej je vnorený **iba jeden príkaz, nemusíme ho odsadzovať a písať do nového riadka, ale do toho istého riadka za dvojbodku.**

Úloha 14

Zostavte podmienku kreslenia týchto obrazcov:



[55] *Na začiatku to z didaktických dôvodov neodporúčame používať práve preto, aby si žiaci uvedomili to, že priraďovací príkaz nie je komutatívny a pred priradením sa najskôr musí vyčísliť hodnota výrazu na pravej strane.*

6.4 Neúplné vetvenie

Neúplné vetvenie je vetvenie `if` bez vetvy `else` – teda, že **blok príkazov sa vykoná len vtedy, ak je platný logický výraz, inak sa nevykoná nič.**

Tak prečo namiesto tohto:

```
1.  if x > 150:
2.      if y < 450:
3.          if y > 150:
4.              if x < 450: farba = "red"
5.              else: farba = "blue"
6.          else: farba = "blue"
7.      else: farba = "blue"
8.  else: farba = "blue"
```

nenapísať len toto?

```
1.  farba = "blue"
2.
3.  if x > 150:
4.      if y < 450:
5.          if y > 150:
6.              if x < 450: farba = "red"
```

Niekedy je vhodné optimalizovať program tým, že vetvu `else` nepoužijeme, ak to nie je nevyhnutné.

Úloha 15

Skúste v danom programe (ten, ktorý vykresľuje červený štvorec a modré okolie z kruhov) vymazať všetky vetvy `else` bez poslednej a bez toho, aby ste pred podmienku napísali **`farba = "blue"`** asi takto:

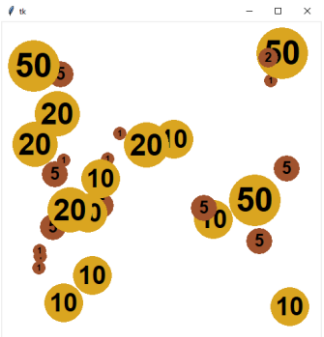
```
1.  if x > 150:
2.      if y < 450:
3.          if y > 150:
4.              if x < 450: farba = "red"
5.  else:
6.      farba = "blue"
```

Znamená to, že vnorené vetvy (podmienky) sú neúplné a hlavná vetva je úplná. Pozorujte zafarbenia krúžkov a pokúste sa zdôvodniť, prečo sa deje to, čo sa deje a prečo sa nedeje to isté, čo tu:

```
1.  if x > 150 and y < 450 and y > 150 and x < 450:
2.      farba = "red"
3.  else:
4.      farba = "blue"
```

Úloha 16

Už ste mali úlohu na náhodné generovanie mincí. Vylepšite program tak, aby 1-, 2- a 5-centové mince mali medenú farbu, 10-, 20-, 50-centové zlatú farbu, aby mince s menšou hodnotou boli menšie a s väčšou hodnotou boli väčšie. Pokúste sa „pohrať“ aj s 1- a 2-eurovou dvojfarebnou mincou (strieborným kruhom v strede alebo na okraji).



6.5 Cyklus s počtom opakovaní ohraničeným podmienkou

Vráťme sa k cyklom. Doteraz sme pracovali len s cyklami, ktorým sme vopred určili, koľkokrát sa majú opakovať. Veľakrát však nastanú situácie, kedy **vopred nevieme učiť, kedy sa má cyklus ukončiť a chceme, aby sa niečo opakovalo dovtedy, kým sa nesplní nejaká podmienka.** Na to slúži cyklus `while` a zapisuje sa veľmi jednoducho:

```
1.  while logický výraz:
2.      príkaz_1
3.      príkaz_2
4.      …
```

Cyklus sa ukončí vtedy, keď podmienka nadobudne nepravdivú hodnotu. Cyklus, ktorého logický výraz je nepravdivý už na začiatku, sa nevykoná ani raz. Všimnime si, že takýto cyklus neobsahuje počítadlo.

PRÍKLAD 30

Už sme počítali priemer známok, kde sme na začiatku museli vopred programu „povedať", koľko známok budeme zadávať. Poďme program prepísať tak, aby sa zadávanie známok skončilo a priemer sa vypočítal, **akonáhle zadáme hodnotu 0.**

```
1.  súčet = 0
2.  i = 0
3.  známka = ""
4.
5.  while známka != 0:
6.      i += 1
7.      známka = int(input("Zadajte známku č. " + str(i) + ": "))
8.      súčet += známka
9.      print("Vykonávam sa", známka)
10.
11. print("Priemer je " + str(súčet / (i - 1)))
```

Poďme si to rozobrať. Asi je vám jasné, že keď máme v cykle príkazy **`i += 1`** a **`súčet += známka`**, že musíme tieto premenné vopred vytvoriť a priradiť im hodnotu **`0`**. Zaujímavejšie je to s tou premennou **`známka`**. Tú musíme tiež vytvoriť vopred, pretože v logickom výraze sa na ňu odvolávame – keby vytvorená nebola, program by vypísal chybu, že takú premennú nepozná. Preto ju musíme vopred vytvoriť, akú hodnotu jej však máme priradiť? Zamyslime sa.

V tomto prípade **je to úplne jedno,** hodnota môže byť akákoľvek (desatinné číslo, reťazec, pole…) okrem nuly. Prečo akákoľvek? Pretože v cykle jej vždy nastavujeme novú hodnotu pomocou funkcie **`input()`** a až potom s ňou pracujeme (pripočítavame ju k výslednému súčtu známok). Okrem nuly preto, lebo keby **`známka == 0`**, cyklus by sa **nevykonal ani raz,** lebo jeho logický výraz by bol nepravdivý, čo by neumožňovalo zadávanie známok a rovno by program skončil s výpisom priemeru **`-0.0`**.[56]

V takýchto prípadoch (ak sú premenné súčasťou logického alebo relačného výrazu cyklu **`while`**) odporúčame premenným jednoducho priradiť hodnotu prázdneho reťazca (**`premenná = ""`**).

A prečo sme zadali podmienku, dokedy známka nebude nula? Je to najjednoduchšie možné riešenie ako zastaviť cyklus – keď totižto zadáme nulu, zmení sa hodnota premennej známka na nulu a tá sa pripočíta do súčtu známok (teda sa nepripočíta nič) a až potom sa bude chcieť vykonať ďalší cyklus, lenže, keďže hodnota **`známka == 0`**, výrok je nepravdivý, teda cyklus sa skončí. Súčet sa teda pripočítaním nuly neskreslí, na čo však musíme pamätať a pri výpočte priemeru musíme pri delení počtom známok jeden cyklus (ten s nulou) odpočítať.

Cyklus je „zraniteľný" – vieme do inputu pridávať „kadejaké somariny" a zistíme, že ak zadáme nejaký textový reťazec, cyklus spadne – shell vypíše chybu. S tým sa však vieme „pohrať" – vyskladaním podmienok, tak, aby neplatné vstupy ignoroval a zarátaval do priemeru len skutočné známky od **`1`** po **`5`**.

PRÍKLAD 31

Pomocou cyklu **`while`** naprogramujte mincovku. Je to program, ktorý pomôže účtovníčke napočítať, koľko a akých bankoviek a mincí má použiť na vyplatenie miezd zamestnancov. Program má fungovať takto: po spustení si bude pýtať jednotlivé mzdy zamestnancov až kým nezadáme nulu. Následne vypíše, koľko ktorých bankoviek a mincí si musí pripraviť účtovníčka, aby vedela zamestnancov presne vyplatiť.[57]

---

[56] **`0 / -1`** *Python registruje ako* **`-0.0`**, *aj keď niečo ako „mínus nula" nepoznáme.*

[57] *Musíme si uvedomiť, že nesmieme mzdy jednotlivých zamestnancov sčítať dokopy a následne „triediť" na jednotlivé počty bankoviek a mincí, musíme to robiť hneď – predstavme si, že máme dvoch zamestnancov so mzdami 1002 a 1003 eur. Nemôžeme si pre nich nachystať 500-eurovky a jednu päťeurovku, lebo päťeurovku na dve a tri eurá nerozdelíme.*

Ako budeme postupovať? Budeme potrebovať počítadlá jednotlivých nominálnych hodnôt bankoviek a mincí, tiež definíciu premennej, ktorá bude uchovávať hodnoty vstupu – hodnoty mzdy.

V cykle **while** sa bude diať toto: na začiatku každého cyklu sa vždy spýtame na mzdu zamestnanca, ktorú následne hneď rozložíme na počet jednotlivých bankoviek a mincí, a to využitím známych funkcií celočíselného delenia a zvyšku po delení – operátory **//** a **%**.

```
1.  päťstovky = dvestovky = stovky = päťdesiatky = dvadsiatky = 0
2.  desiatky = päťky = dvojky = jednotky = 0
3.
4.  z = ""
5.
6.  while z != 0:
7.      z = int(input("Zadaj mzdu: "))
8.      päťstovky   += z // 500
9.      dvestovky   += z%500 // 200
10.     stovky      += z%500%200 // 100
11.     päťdesiatky += z%500%200%100 // 50
12.     dvadsiatky  += z%500%200%100%50 // 20
13.     desiatky    += z%500%200%100%50%20 // 10
14.     päťky       += z%500%200%100%50%20%10 // 5
15.     dvojky      += z%500%200%100%50%20%10%5 // 2
16.     jednotky    += z%500%200%100%50%20%10%5%2 // 1
17.
18. print("500-ky:", päťstovky)
19. print("200-ky:", dvestovky)
20. print("100-ky:", stovky)
21. print("50-ky:",  päťdesiatky)
22. print("20-ky:",  dvadsiatky)
23. print("10-ky:",  desiatky)
24. print("5-ky:",   päťky)
25. print("2-ky:",   dvojky)
26. print("1-ky:",   jednotky)
```

Teda počet päťstoviek je číslo **z** delené **500** so zanedbaním zvyšku, pričom tento zvyšok je následne podelený bankovkou s nižšou nominálnou hodnotou a tak ďalej... až po jednotku. Akonáhle zadáme nulu, takýto proces síce prebehne aj s ňou, no nič sa tým nezmení – nula delené hocičo je vždy nula, lebo je to neutrálny člen. Preto ju je vhodné používať ako hodnotu ukončenia cyklu.

S cyklami a podmienkami sa odteraz budeme stretávať prakticky v každom programe a ich úlohou bude postupne vás naučiť algoritmicky myslieť a tým tvoriť zložitejšie a komplexnejšie programy, čo je veľmi dôležité. Zostavením vhodných podmienok sa pokúste vyriešiť nasledujúce úlohy. **Je veľmi dôležité venovať nasledujúcim dvom úlohám pozornosť a pokúsiť sa najskôr na všetko samostatne prísť.**[58] Podrobný opis kódu nebudeme robiť, v tomto štádiu by ste už mali byť schopní kódu porozumieť samostatne.

PRÍKLAD 32

Zadajte tri ľubovoľné čísla. Vypíšte ich na obrazovku vzostupne.

Riešenie: ako prvé, samozrejme, musíme zabezpečiť načítanie hodnôt.

```
1.  a = float(input("číslo 1: "))
2.  b = float(input("číslo 2: "))
3.  c = float(input("číslo 3: "))
4.  print()
5.  print("Poradie je:")
6.
```

Následne ponúkame tri možné riešenia:[59]

```
7.  # 1. spôsob
8.  if a < b:
9.      if c < a: print(c, a, b)
10.     else:
11.         if c > b: print(a, b, c)
```

[58] *S cieľom rozvoja algoritmického myslenia odporúčame nepoužívať vstavané funkcie jazyka Python na triedenie, vkladanie, vymazávanie prvkov a odporúčame zakázať ich používať aj na hodinách programovania a aj na maturite.*
[59] *Odporúčame zaoberať sa všetkými, a tiež sa zaoberať tými, na ktoré prišli žiaci sami.*

```
12.         else: print(a, c, b)
13. else:
14.     if c < b: print(c, b, a)
15.     else:
16.         if c > a: print(b, a, c)
17.         else: print(b, c, a)
```

```
7.  # 2. spôsob
8.  if a > b: a, b = b, a
9.  if c < a:
10.     print(c, a, b)
11. else:
12.     if c > b:
13.         print(a, b, c)
14.     else: print(a, c, b)
```

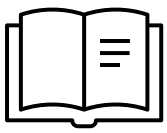
```
7.  # 3. spôsob
8.  if a < b:
9.      if c < a: print(c, a, b)
10.     else:
11.         if c > b: print(a, b, c)
12.         else: print(a, c, b)
13. else: a, b = b, a
```

PRÍKLAD 33

Zadajte *n*. Pozadávajte *n* čísel. Nech program nájde a vypíše najväčšie a najmenšie z nich.

```
1.  n = int(input("počet čísel: "))
2.  MIN = int(input("číslo: "))
3.  MAX = MIN
4.
5.  for i in range(2, n+1):
6.      a = int(input("číslo: "))
7.      if a < MIN:
8.          MIN = a
9.      if a > MAX:
10.         MAX = a
11. print()
12. print("Najmenšie číslo je: ", MIN)
13. print("Najväčšie číslo je: ", MAX)
```

ZHRNUTIE



✓ Vetvenie (alebo podmienka) sa používa, ak máme na základe podmienky rozhodnúť, ktorý príkaz alebo skupina príkazov sa má vykonať.

✓ V Pythone sa rozlišuje jednoduché vetvenie, viacnásobné vetvenie, zložené vetvenie a neúplné vetvenie.

- ✓ Jednoduché vetvenie rozhoduje medzi dvomi stavmi a skladá sa z dohovoreného slova **`if`**, logického výrazu a dvojbodky, za ktorou nasleduje odsadený blok príkazov. Druhá vetva nasleduje za dohovoreným slovom **`else`** a dvojbodkou. Ak je logický výraz pravdivý, vykoná sa prvá vetva, ak je nepravdivý, vykoná sa druhá vetva.
- ✓ Viacnásobné vetvenie okrem vetvy **`if`** a **`else`** obsahuje ľubovoľný počet vetiev **`elif`** tiež s logickým výrazom. Vykoná sa vždy len jedna vetva – prvá pravdivá, bez ohľadu na to, či sú pravdivé aj ostatné. Preto je nevyhnutné výrazy vo viacnásobnom vetvení konštruovať tak, aby boli navzájom disjunktné.
- ✓ Zložené vetvenie je vetvenie vo vetvení.
- ✓ Neúplné vetvenie je vetvenie bez vetvy **`else`**. Ak logický výraz nie je pravdivý, nevykoná sa nič.
- ✓ Výraz je zápis, ktorý sa skladá z operátorov a operandov a jeho výsledkom je vždy nejaká hodnota.
- ✓ Poznáme aritmetické, relačné a logické výrazy. Výsledkom relačných a logických výrazov je pravdivostná hodnota **`True`** alebo **`False`**.
- ✓ Aritmetické operátory sú **+**, **-**, *****, **/**, **//**, **%**, ******, relačné sú **<**, **>**, **==**, **>=**, **<=**, **!=** a logické sú **`not`**, **`and`**, **`or`**, ktoré sú zároveň aj dohovorenými slovami.
- ✓ Najväčšiu prednosť má umocňovanie, potom násobenie s delením a potom sčítanie s odčítaním.

- ✓ Všetky relačné operátory majú prednosť pred aritmetickými a tiež logické výrazy majú prednosť pred relačnými. Negácia má prednosť pred konjunkciou a konjunkcia má prednosť pred disjunkciou.
- ✓ Oboznámili sme sa s novým údajovým typom – bool, ktorý nadobúda hodnoty `True` alebo `False` a sú zároveň dohovorenými slovami.
- ✓ Podľa toho, koľko operandov potrebuje operátor, aby vykonal operáciu, delíme operátory na nulárne, unárne, binárne a ternárne.
- ✓ Okrem cyklu s vopred známym počtom opakovaní poznáme aj cyklus s počtom opakovaní ohraničeným podmienkou – cyklus sa ukončí vtedy, keď podmienka nadobudne nepravdivú hodnotu. Jeho konštrukcia sa začína dohovoreným slovom `while`, nasleduje logický príkaz, dvojbodka a odsadené príkazy. Takýto cyklus neobsahuje počítadlo.

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem vetvenie.
2. Vymenujte druhy vetvení a charakterizujte ich – akú majú syntax a kedy sa používajú.
3. Čo je to výraz?
4. Aké typy výrazov poznáte a čo obsahujú?
5. Ktoré výrazy nadobúdajú pravdivostné hodnoty?
6. Vysvetlite, čo znamená, že množiny sú navzájom disjunktné.
7. Aký nový údajový typ sme si spomenuli?
8. Vymenujte všetky operátory, ktoré poznáte a zoraďte ich podľa ich priority.
9. Vymenujte aspoň po tri príklady výrazov, kde sa používa unárny a binárny operátor.
10. Charakterizujte cyklus s počtom opakovaní ohraničeným podmienkou.
11. Vymenujte všetky dohovorené slová, s ktorými ste sa už stretli.
12. Nájdite čo najviac rozdielov medzi znakmi **=** a **==**.
13. Naprogramujte nekonečný cyklus.
14. Posúďte pravdivosť tvrdenia. Ak je chybné, opravte ho: Cyklus s počtom opakovaní ohraničeným podmienkou sa ukončí vtedy, keď podmienka nadobudne pravdivú hodnotu.

7 Znakový reťazec

Ciele

Cieľom tejto kapitoly bude pripomenúť si všetky znalosti o znakovom reťazci. Žiak si osvojí systém indexovania v reťazci, pomocou ktorého bude schopný zistiť (dekódovať) požadované údaje z reťazca (napríklad na základe rodného čísla zistiť dátum narodenia a pohlavie), bude vedieť použiť únikové sekvencie, formátovať reťazec, nahradiť požadované znaky alebo skupinu znakov iným znakom alebo skupinou znakov, porozumieť systému porovnávania znakových reťazcov či vedieť zmeniť malé písmo na veľké písmo a opačne.

Už sme pracovali s textovým (znakovým) reťazcom v predchádzajúcich kapitolách. Pripomeňme si, čo už vieme:

- reťazec je postupnosť znakov uzavretá v apostrofoch `''` alebo úvodzovkách `""`
- vieme priradiť reťazec do premennej
- zreťaziť (zlepiť) dva reťazce
- násobiť (zlepiť viac kópií) reťazca
- načítať pomocou vstupu (pomocou `input()`) a vypisovať (pomocou `print()`)
- vyrobiť z čísla reťazec (`str()`), z reťazca číslo (`int()`, `float()`)
- rozobrať reťazec v cykle `for`

Postupne prejdeme tieto možnosti práce s reťazcami a doplníme ich o niektoré novinky. Keďže znakový reťazec je postupnosť znakov uzavretá v apostrofoch `''` alebo úvodzovkách `""`, platí:

- môže obsahovať ľubovoľné znaky (okrem znaku apostrof `'` v apostrofovom `''` reťazci a znaku úvodzovka `"` v úvodzovkovom `""` reťazci)
- musí sa zmestiť do jedného riadka (nesmie prechádzať do druhého riadka)
- môže obsahovať špeciálne znaky, takzvané **únikové sekvencie** (zapisujú sa dvomi znakmi, ale pritom v reťazci reprezentujú vždy len jeden), vždy sa začínajú znakom `'\'` (opačná lomka):
  - `\n` – nový riadok
  - `\t` – tabulátor
  - `\'` – apostrof
  - `\"` – úvodzovka
  - `\\` – opačná lomka

Napríklad reťazec `"Programujeme\nv\nPythone."` sa na shell vypíše takto:

```
Programujeme
v
Pythone.
```

Spomeňme tiež dôležitú funkciu, ktorú budeme používať – `len()`. Táto funkcia zisťuje dĺžku reťazca. Napríklad hodnota `len("Programujeme v Pythone.")` bude `23`.

7.1 Indexovanie reťazca

V nasledujúcich úlohách budeme pracovať s operáciou indexovania. Svojím spôsobom sme indexovanie spomínali už dvakrát[60] (vtedy to bol rozsah) – funkcie `randrange()` a `range()`. Podobným spôsobom funguje aj pristupovanie k jednotlivým znakom reťazca, ibaže parametre sa nepíšu do okrúhlych `()`, ale do hranatých zátvoriek `[]`. Nasleduje prehľad príkladov. Nech `reťazec = "Programujeme v Pythone."` Potom:

znak	P	r	o	g	r	a	m	u	j	e	m	e		v		P	y	t	h	o	n	e	.
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
index	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

[60] *A budeme ho spomínať ešte raz – v kapitole venujúcej sa poliam.*

Potom napríklad:

takýto zápis:	*má takúto hodnotu:*
`reťazec.find("n")`	`20`
`len(reťazec)`	`23`
`reťazec[10]`	`m`
`reťazec[-3]`	`n`
`reťazec[:10]`	`Programuje`
`reťazec[0:len(reťazec)]`	`Programujeme v Pythone.`
`reťazec[0:len(reťazec):1]`	`Programujeme v Pythone.`
`reťazec[3:10]`	`gramuje`
`reťazec[10:3:-1]`	`mejumar`
`reťazec[10::-1]`	`mejumargorP`
`reťazec[3:len(reťazec):2]`	`gaueevPtoe`
`reťazec[::-1]`	`.enohtyP v emejumargorP`
`reťazec[:12] + reťazec[13] + reťazec[15:]`	`ProgramujemevPythone.`
`reťazec[:15] + "jazyku " + reťazec[15:len(reťazec)-2] + reťazec[-1]`	`Programujeme v jazyku Python.`

Teda **`reťazec[odkiaľ : pokiaľ : krok]`**, pričom, ako vidíme, ľubovoľný parameter môžeme vynechať, pozor však na dvojbodky, to však považujeme na základe tabuľky za intuitívne pochopiteľné.

Rovnako si uvedomme, že **`odkiaľ`** je opäť uzavretá spodná hranica a **`pokiaľ`** je otvorená horná hranica tak, ako to bolo pri **`randrange()`** a **`range()`**.

Pozor, reťazec je v pamäti nemenný, teda nemôžeme vykonať napríklad:
`reťazec[10] = "t"`, aby sme dostali `"Programujete v Pythone."`

Ak chceme vykonať zmenu, musíme ho **celý v pamäti prepísať novým reťazcom** – napríklad takto:[61]
`reťazec = reťazec[:10] + "t" + reťazec[11:]`

Nasleduje pár jednoduchších zadaní. K zložitejším cvičeniam orientovaným na prácu s textovým reťazcom sa dostaneme v ďalších kapitolách, ktorých pochopenie uľahčí pokročilejšiu prácu s reťazcami.

PRÍKLAD 34

Palindrómy sú slová alebo čísla, ktoré sa odpredu aj odzadu čítajú rovnako, napríklad 1324231 alebo „jeleňovi pivo nelej“. Napíšte najjednoduchší možný program, ktorý zistí, či je zadaný textový reťazec palindróm. (Na zjednodušenie – technicky „jeleňovi pivo nelej“ nie je palindróm, pretože nesedí diakritika a medzery, zatiaľ to však nemusíte riešiť. Pre program bude teda palindróm `"jelenovipivonelej"`.)

```
1.  a = input("zadaj slovo: ")
2.  if a[::-1] == a:
3.      print("Je to palindróm.")
```

PRÍKLAD 35

Vytvorte program, ktorý vypíše každé slovo zadaného textového reťazca do samostatného riadka.

```
1.  a = input("Zadaj slovo: ")
2.  b = ""
3.
4.  for i in a:
5.      if i == " ":
6.          i = "\n"
7.      b += i
8.  print(b)
```

[61] *Sú na to aj funkcie (`replace()`, `insert()`, `delete()` atď.), tie však z didaktického hľadiska odporúčame zakázať žiakom používať (a to aj na maturite). My si (keď sa to v ďalšej kapitole naučíme) na to vytvoríme svoje vlastné funkcie (podprogramy).*

Úloha 17

Vylepšite program na zisťovanie palindrómu tak, aby aj reťazec `"kobyla ma maly bok"` bralo ako palindróm (teda nezávisle od medzier v reťazci).

Príklad 36

Poďme si prácu s indexovaním textového reťazca precvičiť na vytvorení programu, ktorý bude zisťovať deliteľnosť čísel.

Deliteľnosť čísla n číslom m by sa síce dala jednoducho zistiť napríklad takto: `n % m == 0`. Ak by toto bol pravdivý výrok (zvyšok po delení čísla n číslom m by bol nula), bolo by deliteľné. My sa však pokúsime „nasimulovať" aritmetické zisťovanie (ciferné súčty, posledné dvojčíslia a pod.). Kritériá deliteľnosti a stručný princíp nižšie uvedeného kódu je [7]:

- číslo je deliteľné dvomi, ak posledná cifra čísla je deliteľná dvomi,
- číslo je deliteľné tromi, ak je jeho ciferný súčet deliteľný tromi,
- číslo je deliteľné štyrmi, ak je jeho posledné dvojčíslie deliteľné štyrmi,
- číslo je deliteľné piatimi, ak je jeho posledná cifra 0 alebo 5,
- číslo je deliteľné šiestimi, ak je deliteľné súčasne dvomi aj tromi,
- číslo je deliteľné deviatimi, ak je jeho ciferný súčet deliteľný deviatimi,
- číslo je deliteľné desiatimi, ak je jeho posledná cifra 0,
- číslo je deliteľné dvanástimi, ak je deliteľné súčasne tromi aj štyrmi.

Dôležité je uvedomiť, že **najskôr potrebujeme pracovať s číslom ako s textovým reťazcom** – teda v tomto prípade s akousi **postupnosťou cifier.** Až keď „získame" cifry, ktoré potrebujeme, potom ich pretypujeme na celé číslo.

Deliteľnosti čísel na základe poslednej cifry (2, 5, 10) alebo posledných niekoľkých cifier (4) vieme spraviť zistením posledného znaku/znakov (cifry/cifier). Nech `a` je „číslo" typu textového reťazca. Potom je číslo deliteľné dvomi, ak `a[-1] % 2 == 0`. Deliteľné štyrmi je, ak `a[n-2:] % 4 == 0`, deliteľné piatimi, ak `a[-1] == 0 or a[-1] == 5`.

Deliteľnosti čísel na základe ciferných súčtov (3, 9) už vieme robiť – **rozoberaním postupnosti** (zadaného „textu" – čísla) **v cykle**: `for i in a: del3 += int(i)` a následne zisťovaním pravdivostnej hodnoty `int(del3 % 3) == 0`

Deliteľnosti na základe iných deliteľností (6, 12 – číslo je deliteľné 6, ak je 2 aj 3 súčasne) vieme robiť rôznymi premennými. Napríklad pomocná premenná `d = 0`. Ak je číslo deliteľné dvomi, v danej vetve bude príkaz `d += 1`, ak je deliteľné tromi, tiež `d += 1` a následne bude deliteľné 6, ak `d == 2`.

Bližšie si rozoberieme a nasimulujeme deliteľnosť 11 a 17. Napíšme program tak, aby na shell vypisoval postup zisťovania.

Číslo je deliteľné 11, ak je **rozdiel súčtu číslic na párnom a nepárnom mieste deliteľný 11.** Môžeme to riešiť napríklad takto: v cykle budeme postupne striedavo ukladať (pripočítavať) párne a nepárne cifry do dvoch premenných. Nakoniec odčítame a zistíme: `(nepárny - párny) % 11 == 0`. Ak to platí, číslo je deliteľné 11.

Kritériá deliteľnosti 17 si vyskúšame dve.

Prvé hovorí, že číslo je deliteľné 17, ak výsledok nasledujúceho postupu je deliteľný sedemnástimi. Majme číslo (napr. 1471605), ktorému chceme zistiť deliteľnosť.

1. Číslu zoberme cifru z miesta jednotiek (teda 5, vznikne 147160).
2. Túto cifru vynásobíme 5 ($5 \cdot 5 = 25$).

3. Následne od novovzniknutého čísla (147160) odčítame týchto 25. Vznikne 147135.
4. Postup opakujeme: Číslu 147135 zoberieme poslednú cifru (5), ostane 14713 atď.
   Postup opakujeme dovtedy, **kým je rozdiel čísel nezáporný. Číslo je deliteľné 17, ak posledný takýto rozdiel je buď nula alebo číslo deliteľné 17.**

Vyskúšajte si aj ďalšie – ekvivalentné kritérium. Číslo je deliteľné sedemnástimi, ak je výsledok nasledujúceho postupu deliteľný sedemnástimi: striedavo sa odčítajú a pripočítajú dvojice cifier vynásobené dvomi a medzivýsledky sa vždy delia dvomi. Konečný výsledok sa potom vynásobí násobkom desať tak, aby vyšlo celé číslo. Teda pre číslo 1471605 to znamená:

1. na začiatku je nula a posledné dvojčíslie je `"05"`, teda $(0 + 2 \cdot 5) \div 2 = 5$.
2. Ďalšie dvojčíslie je `"16"`, predchádzajúci výsledok bol 5. Teda $(5 - 2 \cdot 16) \div 2 = -13{,}5$.
3. Ďalšie dvojčíslie je `"47"`, teda $(-13.5 + 2 \cdot 47) \div 2 = 40{,}25$.
4. Ďalšie „dvojčíslie“ bude `"01"`, teda $40.25 - 2 \cdot 1 = 38{,}25$.

Keďže posledný výpočet nie je medzivýsledkom, dvomi ho už nedelíme. Jeho hodnota je 38.25, mantisa[62] je 3825. **3825 je deliteľné 17, z čoho vyplýva, že aj 1471605 je deliteľné 17.**

(Ak by vám pomohlo využitie predčasného ukončenia cyklu s pevným počtom opakovaní alebo potrebujete pretypovať „nie úplne čisté“ textové reťazce na čísla, pozrite si o tom viac za riešením tohto príkladu.)

Uvádzame, ako sme úlohy na deliteľnosť riešili my. Nájdete efektívnejší spôsob zápisu algoritmu na deliteľnosť 17 (bez použitia príkazu `break`)?

```
1.  print("Program zistí deliteľnosť prirodzeného čísla číslami "
2.        "2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 a 17")
3.
4.  a = input("Zadaj číslo: ")
5.  n = len(a)
6.  s = d = e = 0
7.
8.  if int(a[-1]) % 2 == 0:
9.      print("deliteľné dvomi")
10.     d += 1
11. else: print("nedeliteľné dvomi")
12.
13. del3 = 0
14. for i in a:
15.     del3 += int(i)
16. if int(del3 % 3) == 0:
17.     print("deliteľné tromi")
18.     d += 1
19.     e += 1
20. else: print("nedeliteľné tromi")
21.
22. if int(a[n-2:]) % 4 == 0:
23.     print("deliteľné štyrmi")
24.     e += 1
25. else: print("nedeliteľné štyrmi")
26.
27. if int(a[-1]) == 0 or int(a[-1]) == 5: print("deliteľné piatimi")
28. else: print("nedeliteľné piatimi")
29.
30. if d == 2: print("deliteľné šiestimi")
31. else: print("nedeliteľné šiestimi")
32.
33. #delenie siedmimi a ôsmimi na zjednodušenie:
34. if int(a)%7 == 0: print("deliteľné siedmimi")
35. else: print("nedeliteľné siedmimi")
36.
37. if int(a)%8 == 0: print("deliteľné ôsmimi")
38. else: print("nedeliteľné ôsmimi")
39.
40. del9 = 0
41. for i in a:
42.     del9 = del9 + int(i)
43. if int(del9 % 9) == 0:
44.     print("deliteľné deviatimi")
45.     d += 1
46. else: print("nedeliteľné deviatimi")
```

[62] *Platné cifry čísla bez desatinnej čiarky.*

```
47.
48. if int(a[-1]) == 0: print("deliteľné desiatimi")
49. else: print("nedeliteľné desiatimi")
50.
51. # ***** deliteľnosť jedenástimi: *****
52.
53. nepárny = 0
54. párny = 0
55.
56. for i in range(n):
57.     if i%2 == 0:
58.         u = a[i]
59.         nepárny += int(a[i])
60.     else:
61.         v = a[i]
62.         párny += int(a[i])
63.
64. print("\n  Rozdiel súčtu cifier na párnych a nepárnych miestach:\n  (", end = "")
65.
66. for i in range(n):
67.     if i%2 == 0:
68.         u = a[i]
69.         print(u, end = "")
70.     elif i < (n-1):
71.         print(" + ", end = "")
72.
73. print(") - (", end = "")
74.
75. for i in range(n):
76.     if i%2 != 0:
77.         v = a[i]
78.         print(v, end = "")
79.     elif i < (n-1):
80.         print(" + ", end = "")
81.
82. print(") = " + str(nepárny - párny))
83.
84. print("  (" + str(nepárny)+ ") - (" + str(párny)+ ") = " + str(nepárny - párny))
85.
86. if (nepárny - párny) % 11 == 0:
87.     print("deliteľné jedenástimi\n")
88. else:
89.     print("nedeliteľné jedenástimi\n")
90.
91. #************
92.
93. if e == 2: print("deliteľné dvanástimi\n")
94. else: print("nedeliteľné dvanástimi\n")
95.
96. # ***** deliteľnosť sedemnástimi 1. spôsob: *****
97. print("deliteľnosť 17-timi - 1. spôsob:")
98.
99. dĺžka = n
100.pomocný = a
101.
102.if int(a) < 100:
103.    print("Číslo je príliš malé")
104.
105.else:
106.    for i in range(dĺžka):
107.            reťazec = pomocný[:(dĺžka-i)] # reťazec skracujem vždy o jedno
108.            if dĺžka-i-1 <= 1: # lebo potom nesedí dĺžka reťazca menšenca [0:0]
109.                break
110.            menšiteľ = int(reťazec[(dĺžka-i-1)]) * 5 # cifru na mieste jednotiek * 5
111.            menšenec = int(reťazec[0:(dĺžka-i-1)]) # skrátený reťazec ešte o 1
112.            rozdiel = str(menšenec - menšiteľ) # na konci z neho usudzujem deliteľnosť
113.            pomocný = rozdiel # to bude "nový" reťazec
114.            if int(rozdiel) < 0: # lebo počítať rozdiel až do mínusu nemá zmysel
115.                break
116.            print("Reťazec:", reťazec)
117.            print("Menšenec:", menšenec)
118.            print("Menšiteľ:", menšiteľ)
119.            print("Rozdiel:", rozdiel)
120.            print()
121.
122.    if int(rozdiel)%17 == 0:
123.        print("Číslo je deliteľné 17 (lebo posledný rozdiel je del. 17)\n")
124.    else:
125.        print("Číslo nie je deliteľné 17\n")
```

```
126.
127. # ***** deliteľnosť sedemnástimi 2. spôsob: *****
128. print("deliteľnosť 17-timi - 2. spôsob:")
129. dĺžka = n + 2
130. začiatok = 0
131. a = a + "00"
132.
133. if dĺžka%2 != 0:
134.     a = "0" + a
135.     dĺžka = dĺžka + 1
136.
137. for i in range (0, dĺžka, 2):
138.     u = a[(dĺžka-2-i):(dĺžka-i)]
139.     if i == dĺžka - 2:
140.         if i % 4 == 0:
141.             koniec = (začiatok - 2*int(u))
142.             print("(" + str(začiatok) + " - 2×" + str(u) + ") = " + str(koniec))
143.         else:
144.             koniec=(začiatok+2*int(u))
145.             print("(" + str(začiatok) + " + 2×" + str(u) + ") = " + str(koniec))
146.         začiatok = koniec
147.     else:
148.         if i % 4 == 0:
149.             koniec = (začiatok - 2*int(u)) / 2
150.             print("(" + str(začiatok) + " - 2×" + str(u) + ") ÷ 2 = " + str(koniec))
151.         else:
152.             koniec = (začiatok + 2*int(u)) / 2
153.             print("(" + str(začiatok) + " + 2×" + str(u) + ") ÷ 2 = " + str(koniec))
154.         začiatok = koniec
155.
156. koniec = str(koniec)
157. krok = 0
158. for i in koniec:
159.     krok += 1
160.     if i == ".":
161.         mantisa = int(koniec[:krok-1]+koniec[krok:])
162.         print("mantisa: " + str(mantisa) + "\n")
163.
164. if mantisa%17 == 0:
165.     print("Číslo je deliteľné 17.")
166. else:
167.     print("Číslo nie je deliteľné 17.")
```

Ak by ste to potrebovali využiť, vedzte, že pretypovacia funkcia **`int()`** dokáže zobrať aj textový reťazec s medzerami a dokáže odstrániť všetky neplatné cifry (nuly pred číslom), napríklad:

```
int("15")
15
int("    15")
15
int("000150")
150
int("15   ")
15
```

Niekedy je v cykle **`for`** výhodné použiť príkaz **`break`**. Príkaz umožňuje „vyskočiť“ z cyklu – predčasne ukončuje vykonávanie cyklu a zaručuje, že za ním sa vykonajú až príkazy, ktoré nasledujú za telom cyklu. Vyskúšajme si, čo program vypíše na shell v nasledujúcom jednoduchom príklade:

```
1.  text = "abcde0fgh"
2.
3.  for i in text:
4.      if i == "0":
5.          break
6.      else:
7.          print(i)
8.  print("Toto sa vykonáva za breakom")
```

Vidíme, že úlohou cyklu je prechádzať znakmi v textovom reťazci a vypisovať ich na shell. Ak však „natrafí“ na znak **`"0"`**, cyklus sa ukončí a program pokračuje ďalšími príkazmi. Takýmto spôsobom sa dajú „prepísať“ do cyklu **`for`** niektoré algoritmy používajúce cyklus **`while`**. Príkaz **`break`**, respektíve algoritmus postavený na cykle **`for`** s potrebou použitia príkazu **`break`** však používajte len v nevyhnutných prípadoch. Ak je to možné, použite vždy cyklus **`while`** – na to bol určený a je to aj programátorsky správnejšie.

PRÍKLAD 37

Ďalšia taká komplexnejšia úloha s dlhším kódom bude program na spracovávanie („kódovanie/dekódovanie“) rodného čísla. Pohrajte sa s podmienkami ošetrenia vstupov (ak používateľ zadá nesprávne hodnoty), precvičíte si tým doteraz spomenuté konštrukcie (cykly, podmienky, údajové typy a prácu s textovým reťazcom).

Napíšte program, ktorý bude pracovať takto:

1. Zadaj rodné číslo.
2. Over platnosť rodného čísla (do 31. 12. 1953 9-miestne, do 1. 1. 1954 10-miestne a deliteľné číslom 11).
3. Na základe rodného čísla vypíš pohlavie a dátum narodenia (prvé dvojčíslie je rok, druhé mesiac, u žien zvýšený o 50, tretie je deň).

Tiež opačná úloha:

1. Zadaj dátum narodenia a pohlavie.
2. Vygeneruj 10 náhodných platných rodných čísel (posledné štvorčíslie tak, aby RČ bolo deliteľné číslom 11). Ak je RČ staršie (trojčíslie na konci), nemusí byť deliteľné č. 11.
3. Vygeneruj a vypíš všetky rodné čísla, ktoré môžu dostať novorodenci v tento deň.

Riešenie:

```
1.   import random
2.
3.   # 1. časť úlohy:
4.
5.   print("RODNÉ ČÍSLA:")
6.   print()
7.
8.   a = input("Zadaj číslo pred lomkou: ")
9.   b = input("Zadaj číslo za lomkou: ")
10.  print("rodné číslo je: " + str(a) + "/" + str(b))
11.  dĺžka = len(a + b)
12.
13.  if a[2:4] == "01" or a[2:4] == "51":    mesiac = "januára"
14.  elif a[2:4] == "02" or a[2:4] == "52":  mesiac = "februára"
15.  elif a[2:4] == "03" or a[2:4] == "53":  mesiac = "marca"
16.  elif a[2:4] == "04" or a[2:4] == "54":  mesiac = "apríla"
17.  elif a[2:4] == "05" or a[2:4] == "55":  mesiac = "mája"
18.  elif a[2:4] == "06" or a[2:4] == "56":  mesiac = "júna"
19.  elif a[2:4] == "07" or a[2:4] == "57":  mesiac = "júla"
20.  elif a[2:4] == "08" or a[2:4] == "58":  mesiac = "augusta"
21.  elif a[2:4] == "09" or a[2:4] == "59":  mesiac = "septembra"
22.  elif a[2:4] == "10" or a[2:4] == "60":  mesiac = "októbra"
23.  elif a[2:4] == "11" or a[2:4] == "61":  mesiac = "novembra"
24.  elif a[2:4] == "12" or a[2:4] == "62":  mesiac = "decembra"
25.
26.  deliteľnosť = int((a + b)) % 11
27.
28.  if (((mesiac == "januára" or mesiac == "marca" or mesiac == "mája" or mesiac == "júla"
29.        or mesiac == "augusta" or mesiac == "októbra" or mesiac == "decembra")
30.       and (int(a[4:6]) <= 31))
31.
32.      or (mesiac == "apríla" or mesiac == "júna" or mesiac == "septembra" or mesiac == "novembra"
33.          and (int(a[4:6]) <= 30))
34.
35.      or (mesiac == "februára"
36.          and (int(a[4:6]) <= 28))):
37.
38.      if dĺžka == 9 and int(a[0:2]) <= 53:
39.          print("je to starší typ RČ")
40.
41.          if a[2] == "5" or a[2] == "6":
42.              print("pohlavie: žena")
43.              print("Dátum narodenia je: " + str(a[4:6]) + ". " + mesiac + " 19" + str(a[0:2]))
44.          else:
45.              print("pohlavie: muž")
46.              print("Dátum narodenia je: " + str(a[4:6]) + ". " + mesiac + " 19" + str(a[0:2]))
47.
48.      else:
49.          if dĺžka == 10:
50.              if deliteľnosť == 0:
51.                  print("je to novší typ RČ")
```

```
52.
53.                 if a[2] == "5" or a[2] == "6":
54.                     print("pohlavie: žena")
55.
56.                     if int(a[0:2])>=54:
57.                         print("Dátum narodenia je: " + str(a[4:6]) + ". " + mesiac + " 19" +
58.                               str(a[0:2]))
59.                     else:
60.                         print("Dátum narodenia je: " + str(a[4:6]) + ". " + mesiac + " 20" +
61.                               str(a[0:2]))
62.
63.                 else:
64.                     print("pohlavie: muž")
65.
66.                     if a[0] == "5" or a[0] == "6" or a[0] == "7" or a[0] == "8" or a[0] == "9":
67.                         print("Dátum narodenia je: " + str(a[4:6]) + ". " + mesiac + " 19" +
68.                               str(a[0:2]))
69.                     else:
70.                         print("Dátum narodenia je: " + str(a[4:6]) + ". " + mesiac + " 20" +
71.                               str(a[0:2]))
72.             else:
73.                 print("RČ je neplatné.")
74.         else:
75.             print("RČ je neplatné.")
76. else:
77.     print("RČ je neplatné.")
78.
79. print()
80.
81. # 2. časť úlohy:
82.
83. x = input("Zadaj dátum narodenia (DD.MM.RRRR): ")
84. y = input("Zadaj pohlavie (muž/žena): ")
85.
86. if ((int(x[0:2]) <= 31 and
87.       (x[3:5] == "01" or x[3:5] == "03" or x[3:5] == "05" or x[3:5] == "07"
88.        or x[3:5] == "08" or x[3:5] == "10" or x[3:5] == "12"))
89.
90.     or (int(x[0:2]) <= 30 and
91.         (x[3:5] == "04" or x[3:5] == "06" or x[3:5] == "09" or x[3:5] == "11"))
92.
93.     or (int(x[0:2]) <= 28 and
94.         x[3:5] == "02")):
95.
96.     if int(x[6:10]) < 1954:
97.
98.         if y == "muž":
99.             print("10 ľubovoľných RČ môže byť napríklad:")
100.            for i in range(10):
101.                za_lomkou = random.randrange(100,1000)
102.                print(str(x[8:10]) + str(x[3:5]) + str(x[0:2]) + "/" + str(za_lomkou))
103.        else:
104.            print("10 ľubovoľných RČ môže byť napríklad:")
105.            for i in range (10):
106.                za_lomkou = random.randrange(100,1000)
107.                print(str(x[8:10]) + str((int(x[3:5]) + 50)) + str(x[0:2]) + "/" +
108.                      str(za_lomkou))
109.
110.    else:
111.        if y == "muž":
112.            print("10 ľubovoľných RČ môže byť napríklad:")
113.            u = 0
114.            while u < 10:
115.                za_lomkou = random.randrange(1000,10000)
116.                if (int(str(x[8:10]) + str(x[3:5]) + str(x[0:2]) + str(za_lomkou)))%11 == 0:
117.                    print(str(x[8:10]) + str(x[3:5]) + str(x[0:2]) + "/" + str(za_lomkou))
118.                    u = u + 1
119.
120.        else:
121.            print("10 ľubovoľných RČ môže byť napríklad:")
122.            u = 0
123.            while u < 10:
124.                za_lomkou = random.randrange(1000,10000)
125.                if (int(str(x[8:10]) + str((int(x[3:5]) + 50)) + str(x[0:2]) +
126.                    str(za_lomkou)))%11 == 0:
127.                    print(str(x[8:10]) + str((int(x[3:5]) + 50)) + str(x[0:2]) + "/" +
128.                          str(za_lomkou))
129.                    u = u + 1
130.
```

```
131. else:
132.     print("Zadali ste neplatný dátum.")
133.
134.
135. print()
136.
137. x = input("Zadaj dátum narodenia (DD.MM.RRRR): ")
138. y = input("Zadaj pohlavie (muž/žena): ")
139.
140. if ((int(x[0:2]) <= 31 and
141.      (x[3:5] == "01" or x[3:5] == "03" or x[3:5] == "05" or x[3:5] == "07"
142.       or x[3:5] == "08" or x[3:5] == "10" or x[3:5] == "12"))
143.
144.     or (int(x[0:2]) <= 30 and
145.         (x[3:5] == "04" or x[3:5] == "06" or x[3:5] == "09" or x[3:5] == "11"))
146.
147.     or (int(x[0:2]) <= 28
148.         and x[3:5] == "02")):
149.
150.     if y == "muž":
151.         print("Všetky možné čísla chlapcov:")
152.         for i in range(1000,10000):
153.             if (int(str(x[8:10]) + str(x[3:5]) + str(x[0:2]) + str(i))) % 11 == 0:
154.                 print(str(x[8:10]) + str(x[3:5]) + str(x[0:2]) + "/" + str(i))
155.
156.     else:
157.         print("Všetky možné čísla dievčat:")
158.         for i in range(1000,10000):
159.             if (int(str(x[8:10]) + str((int(x[3:5]) + 50)) + str(x[0:2]) + str(i))) % 11 == 0:
160.                 print(str(x[8:10]) + str((int(x[3:5]) + 50)) + str(x[0:2]) + "/" + str(i))
161.
162. else:
163.     print("Zadali ste neplatný dátum.")
```

7.2 Formátovanie znakového reťazca

Keď sme potrebovali vytvoriť znakový reťazec, ktorý mal obsahovať veľa premenných vložených medzi text, museli sme každú časť textu dať do úvodzoviek, všetky potrebné hodnoty pretypovať a nakoniec všetko pospájať znamienkom plus. Bolo to asi aj pre vás trochu otravné. Našťastie má na toto Python definovanú zjednodušenú syntax, takzvaný **formátovací reťazec** (neformálne sa to nazýva **f-string**). Uvedieme hneď príklad:

Chceli by sme napríklad vypísať: **"Ahoj, ja sa volám Anička, mám 12 rokov a chodím do 6. ročníka."** a mali by sme premenné: **meno**, **vek**, **ročník**, museli by sme zrejme napísať:

```
print("Ahoj, ja sa volám " + meno + ", mám " + str(vek) + " rokov
a chodím do " + str(ročník) + ". ročníka.")
```

Použitím **f-stringu** stačí zapísať:

```
print(f"Ahoj, ja sa volám {meno}, mám {vek} rokov a chodím do {ročník}.
ročníka.")
```

Pred reťazec musíme napísať písmeno **f** a potom do zložených zátvoriek **{}**[63] zapisujeme premenné, ktoré už ale netreba pretypovať.

7.3 Generovanie farieb

Poďme to nejako zaujímavo využiť. Vráťme sa na malú chvíľu ku kresleniu a spomeňme si, akým spôsobom sme zadávali farby. Boli „ľudsky“ pomenované – **red, green, turquoise, brown, darkgrey**... To sú však len názvy vybraných farieb, ktoré dostali aj takéto mená. Čo však, keď chceme generovať úplné náhodné farby celého farebného priestoru modelu RGB? V Pythone sa štandardne používa **24-bitová farebná hĺbka, to znamená, že každá jedna farba je kódovaná 24 bitmi – jednotkami a nulami.** Znamená to, že každá zložka (červená, zelená aj modrá) je kódovaná $24 \div 3 = 8$ bitmi. V desiatkovej sústave teda vieme povedať, že každá jedna zložka farby môže uchovať $2^8 = 256$ stavov (odtieňov). Keďže sú zložky tri, farieb môže byť 256^3. Preto necháme generovať náhodné hodnoty od 0 po 256^3 (čo je 16 777 216 farieb).

[63] *Zložené zátvorky napíšeme na slovenskej klávesnici stlačením Alt Gr + B a Alt Gr + N.*

Python na zápis farby však nepoužíva hodnoty RGB v desiatkovej, ale šestnástkovej sústave. Pre nás červená farba by bola (255, 0, 0), čierna (0, 0, 0), biela (255, 255, 255), hnedá (140, 70, 20) atď.

V šestnástkovej sústave sa zápis kódu farby **začína mriežkou a šesťmiestnym hexadecimálnym kódom,** ktorý sa na toto dokonale hodí. Pre bielu `"#ffffff"` (lebo 255_{10} = FF_{16}), pre červenú `"#ff0000"` (G a B zložky sú nulové), pre čiernu `"#000000"` a napríklad pre hnedú `"#8c4614"` (pretože 140_{10} = $8C_{16}$, 70_{10} = 46_{16}, 20_{10} = 14_{16}). Vyskúšať si to môžete napríklad tu: https://www.w3schools.com/colors/colors_rgb.asp, [7].

A keďže chceme náhodne vybratú farbu, jednoducho **číslo od 0 do 16 777 216 necháme prekonvertovať do šestnástkovej sústavy a v prípade malého čísla mu doplníme nuly zľava.** Použijeme na to zápis `:06x`. Tomuto a podobným zápisom hovoríme **špecifikácia formátu.** Nebudeme sa nimi zaoberať, teraz však použijeme jeden, ktorý sa používa na celé číslo: má mať šírku 6, zľava doplnenú nulami a číslo sa konvertuje do šestnástkovej sústavy. Nasledujúci príkaz **uloží do premennej `farba` náhodnú farbu:**

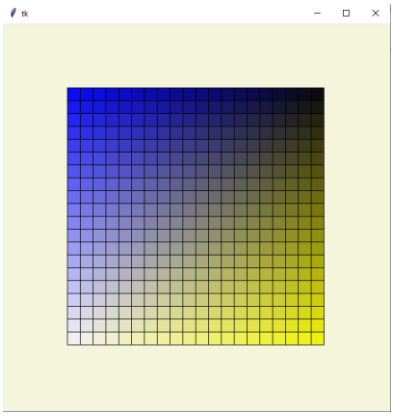
```
farba = f'#{random.randrange(256**3):06x}'
```

Vyskúšajte si to na nasledujúcom programe:

```
1.  import tkinter
2.  import random
3.
4.  canvas = tkinter.Canvas(width = 600, height = 600, bg = "beige")
5.  canvas.pack()
6.
7.  def kresli():
8.      r = random.randrange(256*256*256)
9.      farba = f"#{r:06x}"
10.     canvas.create_rectangle(100, 100, 500, 500, fill = farba)
11.
12. tlačidlo = tkinter.Button(text = "Kresli", command = kresli)
13. tlačidlo.pack(side = tkinter.BOTTOM)
```

PRÍKLAD 38

Už by vám na základe uvedeného mohlo napadnúť, ako naprogramovať takýto farebný gradient:[64]

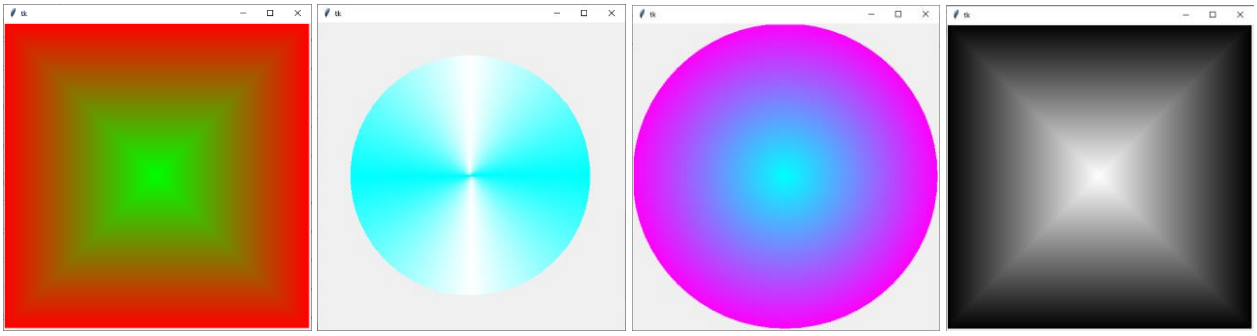


```
1.  import tkinter
2.  import random
3.  canvas = tkinter.Canvas(width = 600, height = 600, bg = "beige")
4.  canvas.pack()
5.
6.  r, g, b = 0, 0, 255
7.  x = y = 100
8.
9.  for i in range(20):
10.     r += (255 // 20)
11.     g += (255 // 20)
12.     for j in range(20):
13.         b -= (255 // 20)
14.         farba = f"#{r:02x}{g:02x}{b:02x}"
15.         canvas.create_rectangle(x, y, x + 20, y + 20, fill = farba, outline = "")
16.         x += 20
17.     b = 255
18.     x = 100
19.     y += 20
```

[64] *Gradient je farebná výplň tvorená postupným prechodom z jednej farby do druhej.*

Príklad 39

Ďalšie príklady gradientov. Riešenie prvých dvoch nájdete nižšie.



```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 510, height = 510)
3.  canvas.pack()
4.
5.  r, g, b = 255, 0, 0
6.  for i in range(255):
7.      r -= 1
8.      g += 1
9.      f = f"#{r:02x}{g:02x}{b:02x}"
10.     canvas.create_rectangle(i, i, 510-i, 510-i, fill = f, outline = "")
11.     canvas.update()
12.     canvas.after(5)
```

```
1.  import tkinter
2.  import math
3.  canvas = tkinter.Canvas(width = 510, height = 510)
4.  canvas.pack()
5.
6.  r, g, b = 0, 255, 255
7.  k = 0
8.  for i in range(255):
9.      r += 1
10.     f = f"#{r:02x}{g:02x}{b:02x}"
11.     k += math.pi / 510
12.     canvas.create_line(255 + math.cos(k)*200, 255 + math.sin(k)*200,
13.                        255 - math.cos(k)*200, 255 - math.sin(k)*200, width = 3, fill = f)
14.     canvas.update()
15.     canvas.after(5)
16.
17. r, g, b = 255, 255, 255
18. for i in range(255):
19.     r -= 1
20.     f = f"#{r:02x}{g:02x}{b:02x}"
21.     k += math.pi / 510
22.     canvas.create_line(255 + math.cos(k)*200, 255 + math.sin(k)*200,
23.                        255 - math.cos(k)*200, 255 - math.sin(k)*200, width = 3, fill = f)
24.     canvas.update()
25.     canvas.after(5)
```

Úloha 18

Naprogramujte animáciu „donekonečna“ stmavujúcich a zosvetľujúcich sa kruhov v odtieňoch sivej. (Pomôcka: použite cyklus **while** a uvedomte si, aké sú zložky RGB odtieňov sivej.)

7.4 Porovnávanie znakových reťazcov

Spomeňme ešte skutočnosť, ktorú možno niekedy využijeme, a to, že **znakové reťazce sa dajú aj porovnávať** (relačnými operátormi **<**, **>**, **<=**, **>=**, **==**, **!=**). Na to Python používa tabuľku UNICODE (UTF-8). Znak, ktorý je v tabuľke „skôr“, je menší ako znak, ktorý je v tabuľke „neskôr“. Z toho však vyplýva, že napríklad slová s diakritikou sa takto porovnávať nedajú, pretože v tabuľke sú na prvých miestach písmená základnej abecedy a až za nimi nasledujú písmená s diakritikou. Preto napríklad platí:

```
"cesnak" < "mrkva"
True
"Mrkva" > "mrkva"
False
```

ale neplatí:

```
"čerešne" < "maliny"
False
```

Ak sú prvé znaky rovnaké, Python ich preskakuje až dokedy nenájde odlišný. Preto platí aj:

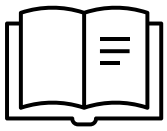
```
"Petronela" > "Petra"
True
```

7.5 Zmena veľkosti

Ďalšie užitočné funkcie sú **`lower()`** a **`upper()`**, ktoré spravia z veľkých písmen malé alebo z malých písmen veľké. Ostatné znaky nemenia. Napríklad:

```
veta = "Dnes Bude pekné POČASIE a teplota 25 °C."
veta.upper()
'DNES BUDE PEKNÉ POČASIE A TEPLOTA 25 °C.'
veta.lower()
'dnes bude pekné počasie a teplota 25 °c.'
```

S textovými reťazcami ešte budeme pracovať v ďalších kapitolách.



ZHRNUTIE

- ✓ Reťazec je postupnosť znakov uzavretá v apostrofoch alebo úvodzovkách.
- ✓ Vieme priradiť reťazec do premennej, zreťaziť dva reťazce, násobiť reťazec, načítať pomocou vstupu a vypísať pomocou výstupu.
- ✓ Vieme tiež vyrobiť z čísla reťazec, z reťazca číslo a rozobrať reťazec v cykle **`for`**.
- ✓ Reťazec môže obsahovať ľubovoľné znaky okrem apostrofu v úvodzovkovom reťazci a úvodzovky v apostrofovom reťazci.
- ✓ Reťazec môže obsahovať špeciálne znaky, tzv. únikové sekvencie nového riadka, tabulátora, apostrofu, úvodzovky a opačnej lomky.
- ✓ Dĺžku reťazca zisťujeme pomocou funkcie **`len()`**.
- ✓ Reťazec indexujeme takto: **`reťazec[odkiaľ : pokiaľ : krok]`**, pričom **`odkiaľ`** je opäť uzavretá spodná hranica a **`pokiaľ`** je otvorená horná hranica – všetko funguje tak, ako to bolo pri **`randrange()`** a **`range()`**.
- ✓ Reťazec je v pamäti nemenný, ak v ňom chceme vykonať zmenu, musíme ho v pamäti celý prepísať novým reťazcom.
- ✓ Pretypovacia funkcia **`int()`** dokáže zobrať aj textový reťazec s medzerami a dokáže odstrániť všetky neplatné cifry (nuly pred číslom).
- ✓ Príkaz **`break`** umožňuje „vyskočiť“ z cyklu – predčasne ukončuje vykonávanie cyklu a zaručuje, že za ním sa vykonajú až príkazy, ktoré nasledujú za telom cyklu.
- ✓ Keď chceme vytvoriť textový reťazec, ktorý má obsahovať veľa premenných vložených medzi text, nemusíme každú časť textu dať do úvodzoviek, všetky potrebné hodnoty pretypovať a nakoniec všetko pospájať znamienkom plus vďaka tzv. f-stringu alebo formátovaciemu reťazcu. Pred reťazec musíme

napísať písmeno **f** a potom do zložených zátvoriek **{}** zapisujeme premenné, ktoré už ale netreba pretypovať.

- ✓ Zápis kódu farby (nielen v Pythone) sa skladá z mriežky a šesťmiestneho hexadecimálneho kódu.
- ✓ Znakové reťazce sa dajú aj porovnávať podľa poradia znakov v tabuľke UNICODE.
- ✓ Veľké písmená na malé a naopak sa dajú meniť funkciami **lower()** a **upper()**.

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem znakový reťazec.
2. Čo sú to únikové sekvencie a čo umožňujú? Vymenujte všetky, ktoré poznáte.
3. Ako zisťujeme dĺžku reťazca a akú dĺžku má každá úniková sekvencia?
4. Opíšte spôsob indexovania reťazca a porovnajte ho s indexovaniami, s ktorými ste sa už stretli (funkcie na generovanie postupnosti a náhodných čísel).
5. Ako musíme postupovať, keď chceme upraviť reťazec, ktorý je uložený v pamäti?
6. Charakterizujte, čo znamená, že 24-bitová farebná hĺbka RGB je uložená v hexadecimálnom kóde.
7. Čo to je f-string? Na čo slúži?
8. Charakterizujte príkaz **break**.
9. Akou funkciou vieme prepísať **"takýto text"** na **"TAKÝTO TEXT"**?
10. Prečo je logický výraz **"človek" < "príroda"** nepravdivý?

8 Algoritmizácia

Ciele

Táto kapitola obsahuje všetku potrebnú teóriu z programovania, ktorú by mal žiak maturitného ročníka ovládať (vrátane teórie, ktorá je spomenutá v iných kapitolách). Žiak bude vedieť vymenovať a vysvetliť vlastnosti algoritmu, bude vedieť prepísať akýkoľvek algoritmus z vývojového diagramu do programovacieho jazyka a naopak, osvojí si základné typografické zásady (úpravu kódu) a zásady postupu pri programovaní. Bude vedieť charakterizovať chyby v programe a klasifikovať programovacie jazyky podľa základných kritérií.

8.1 Charakteristika algoritmu

Keďže sa dostávame postupne k tvorbe čoraz komplexnejších a zložitejších programov a najmä vo zvyšnej časti učebnice to budú výpočtové programy, je namieste túto kapitolu venovať tomu, ako správne vytvoriť postup (algoritmus), ako ho graficky znázorniť a na čo si dať pozor a čo dodržiavať pri programovaní.

Algoritmus je súbor príkazov, ktorý má tieto vlastnosti:

- **konečnosť** – každý algoritmus musí mať jasne definovaný začiatok a podmienky, za ktorých sa končí. Nekonečnosť v informatike nemá zmysel, je to abstraktný pojem, ktorý používajú len matematici.
- **elementárnosť** (jednoduchosť) – pre vykonávateľa algoritmu (procesor, ktorý riadi celý počítač) sú kroky elementárne (pripočítaj, zopakuj, zisti pravdivosť, porovnaj, vráť sa na, preskoč na, vypíš, …), t. j. vie, ako ich má vykonávať
- **determinovanosť** – vždy v každom kroku vieme, čo nasleduje (po porovnaní zopakuj, po pripočítaní vypíš, po vypísaní skoč na, po zopakovaní ukonči, …)
- **hromadnosť (univerzálnosť)** – snažíme sa algoritmus vytvoriť tak, aby sa dal použiť na viacero vstupov
- **rezultatívnosť** – má výsledok na konci a pri rovnakom vstupe vždy rovnaký výstup
- **efektívnosť** algoritmu – meria sa dvojako:
  - **časová** – meria sa strojným časom – čas, ktorý potrebuje algoritmus, aby sa dostal zo začiatku na koniec (ak sa to dá, znížime počet príkazov)
  - **pamäťová** – koľko operačnej pamäte je potrebné rezervovať na správne vykonanie algoritmu (snažíme sa použiť menej premenných)

8.2 Vývojové diagramy

Na jednoznačné a prehľadné vyjadrenie algoritmov sa používa tzv. **vývojový diagram.**

- Je „esperanto“[65] programovacích jazykov.
- Jazyk vývojových diagramov má byť **jednoznačný** – dokáže nakresliť (zaznamenať) postup – algoritmus, ktorý **je univerzálny** a dá sa prepísať do všetkých programovacích jazykov.
- Na označenie začiatku činnosti sa používa **Z** (resp. B – begin), koniec je **K** (resp. E – end). 
- Prvky:
  - **vstup, výstup –** 
  - **vykonávací blok –** 
  - **rozhodovací blok –** **–**  **+** (podľa podmienky – splnená (**+**) alebo nesplnená (**–**))
  - **spojka –**  (používa sa, ak sa postup nezmestí na potrebné miesto, označujú sa číslami vpísanými dovnútra, alebo sa používa, keď potrebujeme spojiť viacero tokov (postupov) do jedného)
  - **postup –** čiary → alebo ← (označuje sa šípkami), mali by viesť iba vodorovne a zvislo a mali by sa podľa možností nekrižovať

[65] *Esperanto je umelo vytvorený ľahko naučiteľný jazyk, ktorý mal pôvodne slúžiť na medzinárodnú komunikáciu – ako však vieme, neujal sa a úlohu medzinárodného jazyka prevzala angličtina.*

- používať by sa mala symbolika ako v matematike – nerovnosť ≠, konjunkcia ∧, disjunkcia ∨ atď.
- ako priraďovací znak používame šípku doľava (←)
- na vstup používame šípku doľava (←), na výstup šípku doprava (→)

8.3 Etapy práce programátora

1. **Zadanie úlohy** – programátor dostane od zákazníka požiadavku vytvoriť program. Usúdi, či dokáže úlohu splniť. Ak nie, jeho „práca“ sa tu končí.
2. **Analýza úlohy** – programátor si môže prizvať odborníka z oblasti, ktorej nerozumie, tento odborník nemusí mať skúsenosti s programovaním.
3. **Algoritmus** – programátor si na základe poznatkov od odborníka zostaví postup, kde je potrebné zvážiť a ošetriť všetky možnosti, aby bol program **idiotensicher,**[66] najlepšie pomocou **vývojového diagramu.**
4. **Realizácia algoritmu** na počítači – pomocou nejakého programovacieho jazyka, testovanie programu.
5. **Odovzdanie programu zákazníkovi.**
6. **Servis –** kontakt, aktualizácie, poradenstvo, poskytovanie licencie pre ďalšie počítače atď.

8.4 Príklady algoritmizácie

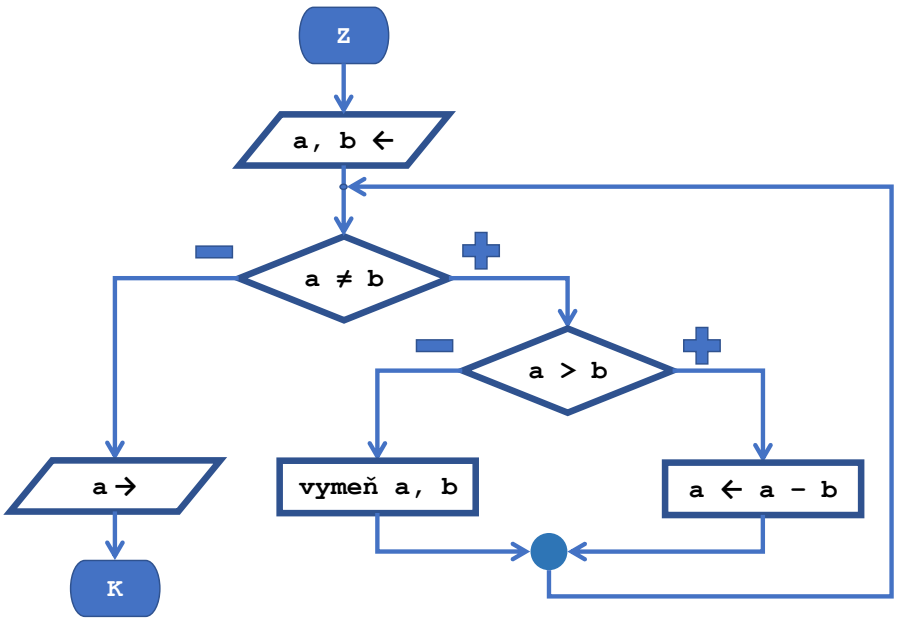
Poďme si ukázať vytvorenie dvoch algoritmov, na základe ktorých by ste mali byť samostatne schopní prepísať ich do akéhokoľvek programovacieho jazyka, ktorý ovládate (pre nás to je Python).

Postup „odčítacej“ verzie Euklidovho algoritmu znie takto: Máme dve čísla. Od väčšieho čísla odčítavajme menšie až dokedy sa čísla nerovnajú. Pre človeka je to zrozumiteľná inštrukcia. Počítaču to však takto „povedať“ nemôžeme, pretože je to síce algoritmus, ale **nie je elementárny a ani determinovaný.** (Ako má počítač vedieť, ktoré od ktorého čísla treba v danom momente odčítať? Čo presne má urobiť po tom, ako čísla odčíta?)

Preto sa nad tým treba zamyslieť takto:

1. máme dve prirodzené čísla a a b
2. opakuj, dokedy sa a nerovná b:
   A. ak je a väčšie ako b, od a odčítaj b
   B. inak vymeň hodnoty a a b a vráť sa na krok 2A
3. vypíš (trebárs) a (a je teraz hodnota najväčšieho spoločného deliteľa)

Teraz, keď sme si algoritmus rozpísali až na elementárne kroky, vieme ľahko urobiť vývojový diagram:



[66] *Program, ktorý je „blbuvzdorný“ (po česky), a teda zabezpečený voči padnutiu pri neúmyselnom aj úmyselnom zadaní neplatného vstupu alebo vykonaní neplatnej operácie, sa volá* ***idiotensicher*** *(je to azda jediný nemecký pojem zaužívaný v programovaní).*

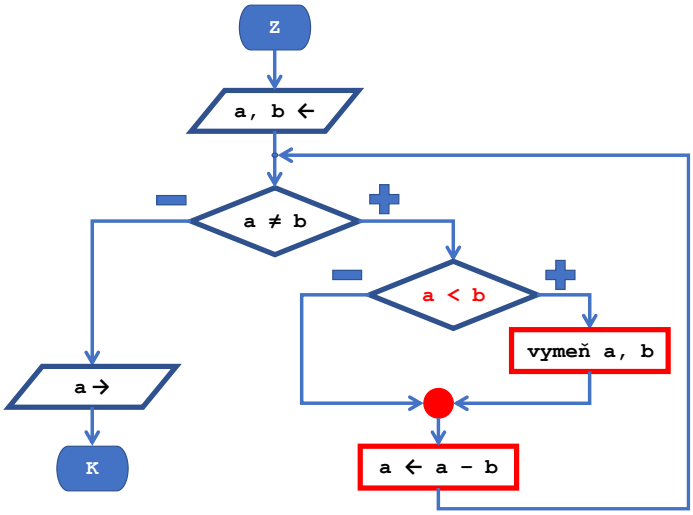
Ako máme takýto diagram čítať?

1. Ako prvé, máme dve prirodzené čísla *a*, *b*, ktoré majú byť požadované na vstupe.
2. Následne zisťujeme, či sa rovnajú.
    a. Ak áno, na výstup (konzolu, v Pythone sa volá shell) vypíšeme *a* a program sa tým končí.
    b. Ak nie, zisťujeme ďalej, či je *a* väčšie ako *b*.
        i. Ak áno, od *a* odčítame *b* a opäť zisťujeme, či sa rovnajú.
        ii. Ak nie, vymeníme vzájomne *a* a *b* a opäť zisťujeme, či sa rovnajú.

A podľa toho vieme napísať program:

```
1.  a = int(input("Zadaj a: "))
2.  b = int(input("Zadaj b: "))
3.
4.  while a != b:
5.      if a > b:
6.          a -= b
7.      else:
8.          a, b = b, a
9.
10. print(a)
```

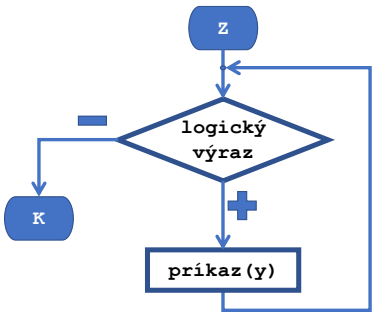
Asi hlavne vďaka vývojovému diagramu vidíme, že nemá zmysel opäť zisťovať, či sa premenné rovnajú, ak sa iba vymieňajú ich hodnoty. Keď sa však pozrieme na samotný kód, zrejme to také neefektívne a „do očí bijúce“ nie je, však? A presne o to ide, na to má vývojový diagram slúžiť. Prepíšte diagram tak, aby sa rovnosť premenných opäť zbytočne nezisťovala len pri ich výmene. Skúste bez toho, aby ste niekde pridávali nový príkaz. Napríklad takto:



Optimalizovaný program:

```
1.  a = int(input("Zadaj a: "))
2.  b = int(input("Zadaj b: "))
3.
4.  while a != b:
5.      if a < b:
6.          a, b = b, a
7.      a -= b
8.
9.  print(a)
```

Na základe predchádzajúceho diagramu si všimnime a uvedomme, ako šikovne sa zapisuje cyklus **while**:



Úloha 19

Na základe predchádzajúceho príkladu rozpíšte pomocou vývojového diagramu a následne naprogramujte násobiacu verziu Euklidovho algoritmu:

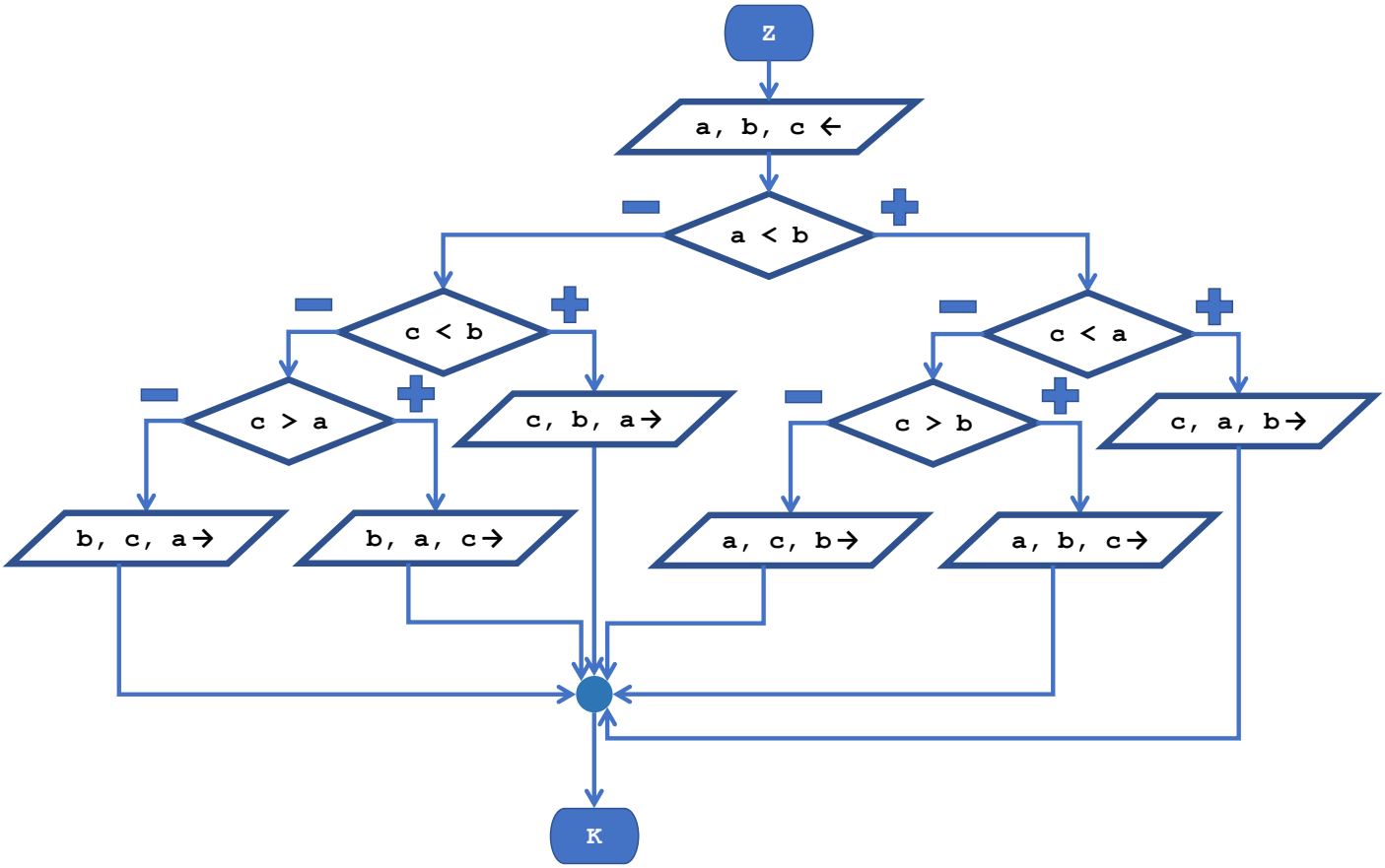
Je to postup, ktorého znenie je takéto: Najväčší spoločný deliteľ prirodzených čísel a a b, $a > b$ určíme takto [9]:

1. Väčšie z čísel a a b nahradíme zvyškom po delení väčšieho čísla menším.
2. Ak menšie z čísel je nula, najväčším spoločným deliteľom je väčšie z čísel. Inak pokračujeme bodom 1.

Príklad 40

Spomeňte si na príklad, v ktorom sme usporadúvali trojicu čísel od najmenšieho po najväčšie. Skúste teraz opačne – na základe programu spraviť vývojový diagram (nejde teraz o to, že sme vybrali práve ten najneefektívnejší, ale o to precvičiť si kreslenie diagramov):

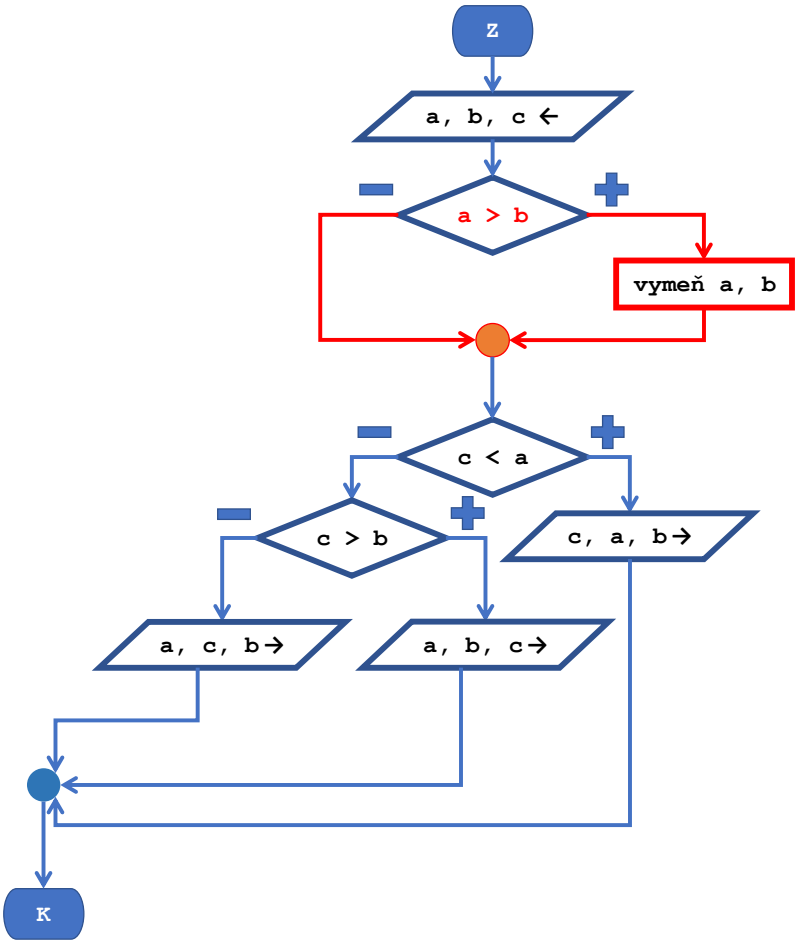
```
1.  a = float(input("číslo 1: "))
2.  b = float(input("číslo 2: "))
3.  c = float(input("číslo 3: "))
4.
5.  if a < b:
6.      if c < a: print(c, a, b)
7.      else:
8.          if c > b: print(a, b, c)
9.          else: print(a, c, b)
10. else:
11.     if c < b: print(c, b, a)
12.     else:
13.         if c > a: print(b, a, c)
14.         else: print(b, c, a)
```



Tu pekne vidíme, že po prvom vetvení sa obidve vetvy vlastne opakujú s tým, že majú iba vzájomne vymenené premenné a a b.

Optimalizujme vývojový diagram podľa tohto algoritmu:

```
1.  # 2. spôsob
2.  if a > b: a, b = b, a
3.  if c < a:
4.      print(c, a, b)
5.  else:
6.      if c > b:
7.          print(a, b, c)
8.      else: print(a, c, b)
```



PRÍKLAD 41

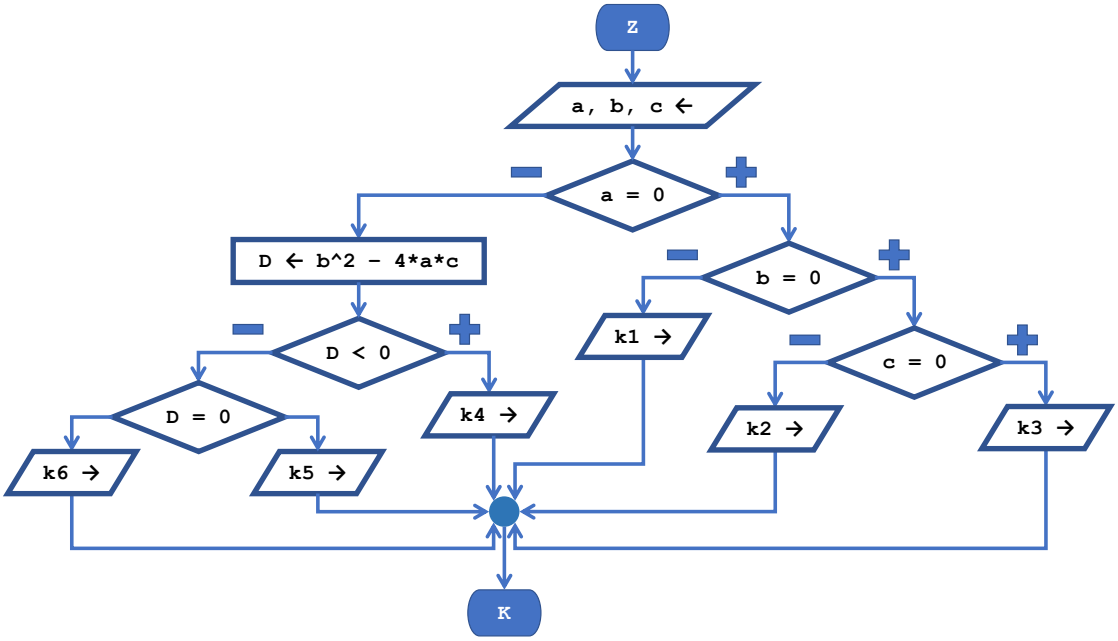
Na základe pravidiel riešenia kvadratických rovníc zostavte program, ktorý po zadaní koeficientov vypíše korene kvadratickej rovnice.

Na základe etáp práce programátora predpokladajme, že vôbec nerozumieme matematike a nevieme, ako sa počíta kvadratická rovnica. To nám neprekáža, prizveme si odborníka, ktorý povie:

- Kvadratická rovnica je rovnica typu $ax^2 + bx + c = 0$, kde a je kvadratický, b je lineárny a c je absolútny člen.
- ak a je nulové:
  - ide o lineárnu rovnicu
  - ak je b nulové:
    - ak je c nulové, rovnica má nekonečne veľa riešení
    - ak je c nenulové, rovnica nemá riešenie
  - ak je b nenulové, rovnica má jeden koreň: $x = -\frac{c}{b}$
- ak a je nenulové:
  - ide o kvadratickú rovnicu
  - počítame diskriminant takto: $D = b^2 - 4ac$

- ak je diskriminant záporný, rovnica nemá riešenie v množine reálnych čísel (žiaci, ktorí majú v maturitnom ročníku povinne voliteľný predmet venujúci sa algebre, vedia, že vtedy má rovnica 2 komplexne združené korene: $x_{1,2} = \frac{-b \pm i\sqrt{|D|}}{2a}$)
- ak je diskriminant rovný nule, rovnica má jeden dvojnásobný reálny koreň: $x = -\frac{b}{2a}$
- ak je diskriminant kladný, rovnica má dva reálne korene: $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$

Na základe tejto informácie od odborníka si už vieme zostaviť vývojový diagram:



k1: Rovnica má riešenie: x = -c/b
k2: Rovnica nemá riešenie.
k3: Rovnica má nekonečne veľa riešení.
k4: Rovnica má 2 komplexne združené korene:
xi1 = -b/2a + i * √(|D|)/2a,
xi2 = -b/2a - i * √(|D|)/2a.
k5: Rovnica má jeden dvojnásobný reálny koreň: x = -b/(2a)
k6: Rovnica má 2 reálne korene:
x1 = (-b + √(|D|))/2a,
x2 = (-b - √(|D|))/2a.

Program by potom mohol vyzerať napríklad takto:

```
1.  import math
2.  a = int(input("Zadaj koeficient a: "))
3.  b = int(input("Zadaj koeficient b: "))
4.  c = int(input("Zadaj koeficient c: "))
5.
6.  if a == 0:
7.      if b == 0:
8.          if c == 0:
9.              print("Rovnica má nekonečne veľa riešení.")
10.         else:
11.             print("Rovnica nemá riešenie.")
12.     else:
13.         print(f"Rovnica má riešenie x = {-c/b}")
14. else:
15.     D = (b*b) - 4*a*c
16.     if D < 0:
17.         print(f"Rovnica má 2 komplexne združené korene:\n"
18.               f" xi1 = {-b/(2*a)} + i{math.sqrt(abs(D))/(2*a)} \n"
19.               f" xi2 = {-b/(2*a)} - i{math.sqrt(abs(D))/(2*a)}.")
20.     else:
21.         if D == 0:
22.             print(f"Rovnica má jeden dvojnásobný reálny koreň x = {-b/(2*a)}.")
23.         else:
24.             print(f"Rovnica má 2 reálne korene:\n"
25.                   f" x1 = {(-b+math.sqrt(D))/2*a} \n x2 = {(-b-math.sqrt(D))/2*a}.")
```

ÚLOHA 20

Vytvorte vývojový diagram a naprogramujte hru „hádaj číslo“, ktorá:

- vygeneruje náhodné číslo
- hráč háda toto číslo, kým ho neuhádne
- ak zadá číslo, ktoré je väčšie ako vygenerované, program vypíše „menej“
- ak zadá číslo, ktoré je menšie ako vygenerované, program vypíše „viac“
- ak hráč uhádne, program vypíše: „uhádol si na x. pokus“

8.5 Typografické zásady

Napísaný kód (naprogramovaný algoritmus – program) by mal byť nielen funkčný a efektívny, ale aj prehľadne zapísaný. Snažte sa dodržiavať hlavné typografické konvencie (ktoré ste si možno všimli z ukážkových kódov tejto učebnice a asi ich už aj podvedome praktizujete):

- mená premenných obsahujú len malé písmená
- znak **=** (priraďovací príkaz) je oddelený medzerou pred aj za
- operácie v aritmetických, relačných a logických výrazoch sú väčšinou tiež oddelené od operandov medzerami (podľa zváženia prehľadnosti ich nemusíme písať, ak používate operátory `*`, `/`, `//`, `%`, ak by sa tým kód zbytočne roztiahol. A navyše, napríklad `a + b*c - e + d//f` dáva jasne graficky najavo, že prioritu má `b*c` a `d//f`)
- riadky programu by nemali byť dlhšie ako 79 znakov
- za čiarky, napr. ktoré oddeľujú parametre vo funkciách, dávame vždy medzeru
- aj pred ľavú aj za pravú zátvorku je vhodné dať medzeru, medzery pred zátvorku však nedávame, ak používame unárny operátor `not()`, alebo keď voláme funkcie – `range()`, `input()`, `print()`, alebo keď indexujeme – `slovo[od:do]`

8.6 Programátorské desatoro

Ponúkame vám súhrn zásad programátora. Veríme, že mnohé ste si už osvojili.

1. **Pýtajte sa, zisťujte** – ten, kto sa nepýta, sa nič nedozvie. K dispozícii je vám učiteľ, spolužiak, obrovská komunita ochotných programátorov na diskusných fórach alebo dokumentácia k programovacím jazykom, nie je hanba googliť, práve naopak – je to jeden z najsilnejších nástrojov každého programátora. Z didaktických dôvodov (pri učení sa začiatkov programovania) však odporúčame nepoužívať „vygoogglené“ konštrukcie, ktoré zjednodušujú to, na čo máte v úlohách prísť sami zo všetkého, s čím ste sa dovtedy v učebnici stretli – úlohy tým stratia význam. Žiadna úloha v tejto učebnici nie je stavaná tak, aby sa nedala vyriešiť na základe dovtedy získaných vedomostí z učebnice. Osobitne však odporúčame zakázať použitie konštrukcie `try:`... `except:`... (konštrukcia na zachytávanie výnimiek), tým by väčšina úloh, stavaných na uvedomenie si a naformulovanie podmienených príkazov, stratila význam.

2. **Píšte do kódu komentáre** – teraz síce viete, čo, prečo a ako ste naprogramovali, verte však, že už o týždeň, nieto o mesiac či keď sa k programu vrátite o pár rokov, nemusíte vedieť, na čo ktorý príkaz slúži. Alebo ak program bude po vás čítať niekto iný, alebo opačne – ak budete program po niekom čítať vy – určite oceníte, keď v ňom budú výstižné komentáre.

3. **Dodržiavajte typografické zásady** – vkladajte medzery medzi operátory, udržiavajte prehľad v kóde (vkladaním prázdnych riadkov vizuálne oddeľujte časti, ktoré spolu najviac súvisia), zrozumiteľne pomenovávajte premenné, nepoužívajte také „jednopísmenové“ premenné, ktoré sú ľahko zameniteľné za iné znaky (malé L (`l`) sa podobá na jednotku (`1`), veľké O (`O`) zas na nulu (`0`) atď.). Hovorí sa totiž, že napísať kód, ktorému rozumie počítač, dokáže každý blázon. Dobrý programátor však napíše kód, ktorému rozumejú predovšetkým ľudia.

4. **Udržiavajte si v programoch poriadok, vytvárajte si priečinky** – výstižne a stručne pomenovávajte svoje projekty, keď budete vytvárať program, ktorý pracuje so súbormi, musia byť systematicky potriedené v priečinkoch. Python to síce nerobí, ale mnohé programovacie prostredia pri ukladaní a prekladaní

programu do spustiteľnej verzie vytvoria množstvo „pomocných“ súborov. Keď takto vytvoríte viac programov, nikto sa v tom neporiadku v jednom priečinku potom nevyzná.

5. **Zálohujte si programy** – hlavne predtým ako sa chystáte v programe urobiť väčšiu zmenu – náhodou sa stane, že to bude „zmena k horšiemu“ a po uložení a preložení programu už nebude cesty späť.

6. **Používajte len to, čomu rozumiete** – aby ste niečo nepoužili na niečo iné ako je určené a aby sa vám to neskôr nevypomstilo v podobe neočakávaného správania programu. Aj na maturite odporúčame – ak si žiak niečo „nové“ vygoogli, musí fungovanie toho vedieť obhájiť a vysvetliť – znova zdôrazňujeme – nepoužívať `try:...` `except:...`!

7. **Neskúšajte všetko spôsobom pokus-omyl –** nepristupujte k oprave nefunkčného programu spôsobom „čo keby som skúsil toto vymazať, tamto presunúť...“ – ak niečo nefunguje, trasujte premenné, to znamená, tam, kde si myslíte, že je chyba, vložte príkaz na informačný výpis hodnôt premennej a sledujte jej zmeny. Možno napríklad zistíte, že niektoré príkazy sa ani nevykonajú (nevypíšu), lebo ste definovali nesplniteľný logický výraz alebo premenná nemení hodnotu tak, ako by ste chceli. Odhalíte tým drvivú väčšinu logických chýb.

8. **Kreslite si vývojové diagramy** – ak vám to pomôže, hlavne zo začiatku, ak sa už dostanete k riešeniu komplexných programov, veľa vecí budete vedieť aj bez diagramu.

9. **Snažte sa rozložiť si danú úlohu na čiastkové úlohy** – to pochopíte po tom, ako sa naučíme vytvárať a používať podprogramy.

10. **Sledujte cudzie kódy** – aj tak sa naučíte programátorsky myslieť a prípadne inšpirovať inými „myšlienkovými pochodmi“ iného programátora. To sa vám podarí bez veľkej námahy len vtedy, keď program bude „uprataný“ – okomentovaný a typograficky upravený.

Veľmi odporúčame na testovanie programov tento vizualizér [10]:

https://pythontutor.com/visualize.html#mode=edit

8.7 Programovacie jazyky

Programovací jazyk je nástroj na opis algoritmu. Podľa úrovne[67] rozdeľujeme:

1. **Strojový kód**
   - je to základný programovací jazyk prispôsobený počítaču (vývojovo najstarší)
   - obsahuje **len jednotky a nuly,** je **strojovo závislý** (inštrukcie pre jeden procesor sú iné ako pre iný procesor)
   - používa **strojové inštrukcie,** ktorým rozumie konkrétny procesor, napr. pripočítanie, presunutie informácie do pamäťového miesta atď.
2. **Nižšie jazyky**
   - sú stále **strojovo závislé**
   - už nepoužívajú len jednotky a nuly, ale inštrukcie sú **v abecedno-číselných kódoch,** napr. ak chceme spočítať: `ADD R1, R2` (znamená to „spočítaj čísla z pamäťových miest `R1` a `R2`“)
   - potrebuje tzv. **asembler,** prekladač nižšieho jazyka **do strojového kódu**
3. **Vyššie jazyky**
   - vznikali až po tom, ako počítače začali mať vyšší výkon
   - je **strojovo nezávislý,** lebo programovací jazyk – teda už **zdrojový kód** – zabezpečí to, aby sa pri neskoršom prepise do **strojového kódu** zabezpečila kompatibilita so zariadením[68]
   - funguje už na základe **príkazov** – viacero inštrukcií sa spája dokopy, takže je kód kratší
   - slová, ktoré sa používajú, sú človeku blízke (napr. z angličtiny), a teda **sa ľahšie človeku učí**
   - jedným z prvých takýchto jazykov bol **PL/I** (Programming language I, z roku 1964)

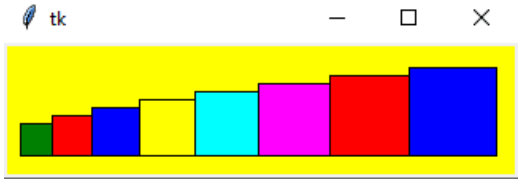
---

[67] Úrovne z hľadiska toho, ako „ľudsky“ jazyk pôsobí – od jazykov, ktoré sú prispôsobené výhradne strojom až po jazyky, ktoré obsahujú pre človeka ľahko zapamätateľné príkazy v angličtine, príp. slovenčine.

[68] Čo vystihuje napríklad slogan programovacieho prostredia Lazarus – „write once, compile anywhere“ – napíš raz a skompiluj (nechaj preložiť zdrojový kód do strojového kódu) kdekoľvek.

Podľa filozofie programovania – súboru princípov, na ktorých je jazyk založený, tzv. **paradigmy,** jazyky rozdeľujeme[69] na dva základné typy:

- **imperatívne** –počítaču zadávame pomocou príkazov **súbor usporiadaných krokov,** ktoré je treba vykonať, aby sa niečo udialo. Chceme napríklad nakresliť takýto obrázok:



  Počítaču zadáme pokyny typu: „nastav prvému štvorcu farbu výplne, nakresli ho na zvolené súradnice, potom si posuň súradnice ďalej, zväčši si rozmery štvorca, zas zmeň farbu výplne, nakresli ďalší štvorec…“ Medzi imperatívne paradigmy patrí:

  - **Štruktúrované programovanie** je programovacia paradigma, kedy počítač dostáva pokyny, ktoré vykonáva v takom poradí, v akom sú naprogramované. Postupnosť príkazov sa vykonáva jeden za druhým (tzv. sekvenčne), nepovoľuje preskakovanie medzi príkazmi a používa iba riadiace štruktúry **výberu a opakovania** (teda cykly a podmienky).[70]

  - **Procedurálne programovanie** – problém riešime tak, že **program rozdelíme na menšie časti,** ktoré spolu súvisia (do podprogramov) a problém vyriešime ich pospájaním.[71] Použitím podprogramov sa program stáva prehľadnejší a zamedzuje sa opakovaniu častí programu, pretože raz definovaný podprogram môžeme zavolať („použiť“) viackrát.

  - **Objektovo orientované programovanie** – prostredie modelujeme tak, že **vytvárame objekty,** ktoré majú svoje vlastnosti, ktoré im naprogramujeme, naprogramujeme vzťahy s inými objektmi, udalosti, ktoré medzi nimi môžu nastať, dedičnosť (ak nejaký objekt je potomkom iného objektu – má rovnaké vlastnosti) atď. Vlastnosti a príkazy objektu sa oddeľujú bodkami (napr. `canvas.pack()`).[72] Na rozdiel od podprogramov[73] majú objekty aj svoju vnútornú pamäť a vlastnosti definované vo svojich podprogramoch, ktoré sa už ale v objektoch nazývajú **metódy.**

- **deklaratívne** – programátor hovorí počítaču, ČO má spraviť a nie AKO to má spraviť. Najnázornejší príklad je jazyk pracujúci s databázami (SQL). Ak napríklad chceme vymazať tabuľku, v ktorej sú rodné čísla, jednoducho napíšeme `DROP TABLE Rodné číslo;`. Nestaráme sa, akým spôsobom má byť tabuľka vymazaná, len sme počítaču „povedali“, aby ju vymazal.

Existuje nespočetné množstvo programovacích jazykov – niektoré podporujú len jednu paradigmu (napr. BASIC podporuje len štruktúrované programovanie, Java je čisto objektovo orientovaný jazyk), v iných jazykoch sa dá programovať viacerými paradigmami – hovoríme, že sú tzv. **multiparadigmatické** (napr. Python).

Existuje ešte jedno zásadné kritérium delenia programovacích jazykov, a to delenie podľa typu prekladu zdrojového kódu do strojového kódu:

- **kompilátor**
  - **prekladá kód** z dohovorených slov do jednotiek a núl (napr. Pascal: `program.pas` sa preloží do `program.exe`[74] = spustiteľná verzia programu) väčšinou pre konkrétny procesor alebo skupinu procesorov
  - preloží program len vtedy, ak je všetko dobre zapísané, **ak nájde chybu, nevykoná nič**

---

[69] Rozdelení jazykov alebo programovacích štýlov je niekoľko a sú podľa rôznych kritérií, dokonca delenia doteraz stále nie sú presne vymedzené.
[70] Takto sme programovali my doteraz.
[71] K procedurálnemu programovaniu a charakteristikám podprogramov sa dostaneme v kapitole venovanej podprogramom.
[72] Aj keď sme „nevedomky“ použili pár metód (vlastne všetko, čo sme oddeľovali tzv. bodkovou notáciou v tvare `niečo.metóda()`), a teda sme „zasiahli“ aj do sféry objektovo orientovaného programovania, nebudeme sa mu v tejto učebnici venovať.
[73] Ktorých lokálne premenné vzniknú pri jeho zavolaní a zaniknú po jeho vykonaní. Aj k tomu sa neskôr dostaneme.
*[74] *.exe prípona je len v OS Windows, v iných systémoch je to trochu inak.*

- používateľ na jeho spustenie **nepotrebuje mať nainštalovaný programovací nástroj**

- **interpreter** (prekladač)
  - nevytvára **`*.exe`** verziu programu – napr. **`program.py`**
  - prekladá a následne vykonáva všetko postupne, **zastaví sa, až keď narazí na chybu**
  - lepší je na vyučovanie, lebo vidíme, kde máme chybu
  - keďže program sa potrebuje najskôr preložiť a až potom vykonať, takto vytvorený program nespustíme na počítači, v ktorom nie je nainštalovaný príslušný programovací nástroj
- **hybridný jazyk** (napr. Java)
  - problém v kompilovaných jazykoch môže byť ich **nekompatibilita**
  - pri programovaní v hybridných jazykoch sa urobí iba tzv. **bajtkód** (čiastočne skompilovaný kód – „polotovar"), skompilujú sa len časti programu, pri ktorých je zaručené, že nebudú nekompatibilné
  - takýto čiastočne skompilovaný kód je schopný vykonávať priamo procesor, ktorého architektúra je na to určená

8.8 Chyby v programe

Určite ste sa už stretli s tým, že ste napísali niečo zle a program sa nespustil. Alebo, že program vykonal čosi iné, než to, čo ste chceli. Stručne si klasifikujeme, aké druhy chýb existujú v programoch:

1. **Syntaktické chyby** – ak napíšeme niečo zle, program sa nedokáže preložiť do binárneho kódu, a teda sa ani nespustí. Sú to „najvďačnejšie" chyby, lebo ich vieme hneď odhaliť a opraviť.

2. **Sémantické chyby** (alebo logické chyby) – očakávame istý výsledok, ale nedeje sa to, čo chceme – buď sme zle pracovali s premennými alebo nebola vhodne zvolená podmienka vo vetvení – ak ju chceme odhaliť, musíme si nechať vypisovať hodnoty premenných, tzv. trasovať.

3. **Chyby počas vykonávania programu** (tzv. run-time errors) – takéto chyby sú „najzákernejšie" a odhaliť sa dajú často až po mnohonásobnom testovaní programu – program sa dokáže správne spustiť, nie sú v ňom syntaktické chyby, ale spadne na chybe, ak **počas jeho vykonávania**:
   a. **zadáme zlý vstup** – údajový typ (program musí byť idiotensicher)
   b. nastane **delenie nulou**
   c. nastane **preplnenie** (v premennej je veľmi veľká alebo veľmi malá hodnota, ktorú nedokáže spracovať alebo máme málo operačnej pamäte, alebo miesta na disku) – prejaví sa to najčastejšie tak, že program a/alebo celý počítač začne „mrznúť"

8.9 Porovnanie syntaxe jazyka Python s jazykmi Pascal a Java

Pozrime sa zo zaujímavosti na porovnanie syntaxí implementácie[75] euklidovho algoritmu [11]:

Python:

```
def euklides(u, v):
    while v != 0:
        u, v = v, u % v
    return abs(u)
```

Pascal:

```
function euklides(u, v: longint): longint;
  var
    t: longint;
  begin
    while v <> 0 do
    begin
      t := u;
```

[75] *Z dôvodu zjednotenia označenia dohovorených slov sú všetky zafarbené namodro, aj keď každý jazyk, dokonca každé programovacie prostredie používa odlišné zvýrazňovanie.*

```
      u := v;
      v := t mod v;
    end;
    euklides := abs(u);
  end;
```

Java:

```
public static int euklides(int u, int v)
{
        while(v > 0)
        {
                int t = t % v;
                u = v;
                v = t;
        }
        return u;
}
```

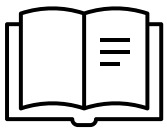
Ako môžeme vidieť, najjednoduchšiu syntax má práve Python. Poďme si opísať základné rozdiely – niektoré sme už spomínali:

- **Python nevyžaduje deklarácie premenných**
  - **Pascal:** `euklides(u, v: longint): longint;` – `u`, `v` aj celá funkcia má byť typu `longint` (dlhé celé číslo), tiež pomocná premenná `t: longint;` má byť dlhé celé číslo – je to iba ich deklarácia („vyhlásenie“, že existujú, akého majú byť typu a rezervovanie miesta pre ich hodnoty v operačnej pamäti)
  - **Java:** `int u, int v, int t = t % v;` v prvých dvoch prípadoch ide o deklaráciu, v treťom aj o definíciu (deklarujeme, že premenná `t` má byť celé číslo a hneď jej aj priradíme hodnotu)

- **Python nepoužíva tzv. oddeľovače príkazov, za koniec príkazu sa považuje koniec riadka**
  - **Pascal** používa bodkočiarky ako tzv. **oddeľovač príkazov** (nesmú byť pred vetvou **`else`** a nemali by byť ani za posledným príkazom v bloku)
  - **Java** používa bodkočiarky ako tzv. **ukončovač príkazov** (musia byť za každým príkazom)

- **Python nepoužíva zátvorky, namiesto toho striktne dodržiava odsadenie príkazov na úrovne**
  - **Pascal** nevyžaduje nevyhnutne odsadenie príkazov na rovnakú úroveň, ale „slovné zátvorky“ **`begin`** a **`end;`**
  - **Java** tiež nevyžaduje nevyhnutne odsadenie príkazov na rovnakú úroveň, používa zložené zátvorky **`{ }`**.

- **Mnohé iné odlišnosti** – napríklad znak priradenia, znamienko nerovnosti, Python podporuje hromadné priradenie, iné jazyky si musia pomôcť ďalšou premennou a pod.



ZHRNUTIE

✓ Algoritmus je súbor príkazov, ktorý je konečný, elementárny, determinovaný, hromadný, rezultatívny a efektívny.

✓ Vývojový diagram sa používa na jednoznačné a prehľadné vyjadrenie algoritmu, ktorý sa podľa neho dá prepísať do všetkých programovacích jazykov.

✓ Vývojový diagram obsahuje označenie začiatku činnosti, označenie vstupu a výstupu, vykonávacie, rozhodovacie bloky, spojky a smerové šípky vyjadrujúce postup.

✓ Etapy práce programátora sú: zadanie úlohy, analýza úlohy, tvorba vývojového diagramu, realizácia algoritmu, odovzdanie programu zákazníkovi a jeho servis.

✓ Konvencie programovania v Pythone sú: písať mená premenných s malými písmenami, dávať medzery pred aj za operátory a priraďovací znak, nepísať riadky dlhšie ako 79 znakov a nedávať medzeru pred zátvorku volanej funkcie.

- ✓ Zásady dobrého programátora sú: pýtať sa, písať do kódu komentáre, dodržiavať typografické zásady, udržiavať si v programoch poriadok, používať len to, čomu rozumie, neskúšať spôsobom pokus-omyl, kresliť si vývojové diagramy, snažiť sa rozložiť si úlohu na čiastkové úlohy a sledovať cudzie kódy.
- ✓ Podľa úrovne rozdeľujeme programovacie jazyky na strojový kód a zdrojový kód (kam patria nižšie a vyššie jazyky).
- ✓ Podľa paradigmy ich rozdeľujeme na imperatívne a deklaratívne, imperatívne ďalej na štruktúrované, procedurálne a objektovo orientované… a iné, delení je viacero.
- ✓ Z hľadiska typu prekladu zdrojového kódu do strojového kódu poznáme kompilátory, interpretery (čiže prekladače) a hybridné jazyky.
- ✓ Chyby v programe poznáme syntaktické, sémantiké a chyby počas vykonávania programu.
- ✓ Python, na rozdiel od mnohých iných jazykov, nevyžaduje deklarácie premenných, nepoužíva oddeľovače/ukončovače príkazov (bodkočiarky) a namiesto zátvoriek striktne dodržiava odsadenie príkazov na úrovne.

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem algoritmus.
2. Aký je rozdiel medzi algoritmom a počítačovým programom? Resp., v akom sú vzťahu?
3. Definujte, čo je vývojový diagram a opíšte jeho hlavné prvky, zmysel a zásady jeho tvorby.
4. Vymenujte a stručne charakterizujte etapy práce programátora.
5. Opíšte hlavné typografické zásady písania kódu.
6. Vymenujte delenie programovacích jazykov podľa úrovne a charakterizujte každú z nich.
7. Aký je rozdiel medzi zdrojovým a strojovým kódom?
8. Čo je to programovacia paradigma?
9. Vymenujte delenie paradigiem a charakterizujte každú z nich.
10. Aký je rozdiel medzi kompiláciou a interpretáciou a čím je špecifický „ten tretí“ typ?
11. Aké chyby v programe poznáte a čím sa vyznačujú – ako dokážete zistiť, keď program nepracuje správne, ktorý typ chyby nastal?
12. Stručne opíšte hlavné rozdiely jazyka Python oproti jazykom Pascal a Java, aj z hľadiska syntaxe aj spôsobu prekladu do strojového kódu.

9 Polia

Ciele

Cieľom tejto kapitoly bude práca s poľami. Žiak si osvojí systém indexovania v poli, pomocou ktorého bude schopný vybrať a vložiť požadované údaje z poľa a do poľa, vytvoriť si analógiu so všetkými ostatnými indexovaniami, ktoré sa doteraz naučil a na záver tejto pomerne krátkej kapitoly naprogramovať jednoduchý triediaci algoritmus.

9.1 Zoznamy, polia, n-tice

Už sme robili program, ktorého úlohou bolo vygenerovať niekoľko hodov kockou a spočítať, koľkokrát padlo ktoré číslo. Každú premennú sme museli pripočítavať a vypisovať samostatne.

```
1.  import random
2.
3.  s = t = u = v = w = x = 0
4.
5.  počet = int(input("Počet hodov: "))
6.
7.  for i in range(počet):
8.      a = random.randrange(1, 7)
9.
10.     if   a == 6: s = s + 1
11.     elif a == 5: t = t + 1
12.     elif a == 4: u = u + 1
13.     elif a == 3: v = v + 1
14.     elif a == 2: w = w + 1
15.     else       : x = x + 1
16.
17. print("Počet šestiek:"   , s)
18. print("Počet pätiek:"    , t)
19. print("Počet štvoriek:"  , u)
20. print("Počet trojok:"    , v)
21. print("Počet dvojok:"    , w)
22. print("Počet jednotiek:" , x)
```

Dá sa to však aj oveľa jednoduchšie. Predstavme si taký údajový typ, ktorý by uchovával **niekoľko hodnôt naraz,** teda dvojicu, trojicu, štvoricu… *n*-ticu. Už sme niečo také používali, keď sme zafarbovali útvary náhodne vybratou farbou, ako sme ich do funkcie **`random.choice()`** zapisovali. Bolo to asi takto: **`farba = random.choice(("red", "green", "blue"))`**.

To, čo sme zadali, **bola usporiadaná trojica farieb, bola to skrátka postupnosť a postupnosť je usporiadaný sled nejakých hodnôt** (v matematike čísel; postupnosť 1 2 9 16 25 je iná ako 1 9 16 2 25). Údajové typy, ktoré takúto postupnosť uchovávajú, poznáme[76] v Pythone tri. Sú to **zoznam (list), pole (array) a n-tica (tuple).** Tabuľka uvádza stručný rozdiel medzi nimi [12]:

`list` ***(zoznam)***	**`array`** ***(pole)***	**`tuple`** ***(n-tica)***
je to **usporiadaná n-tica položiek**		
zoznam je meniteľný	pole je meniteľné	n-tica je nemenná (tak, ako reťazec)
zoznam môže uchovávať viac ako jeden typ údajov	pole môže uchovávať iba rovnaké typy údajov	n-tica môže uchovávať viac ako jeden typ údajov

Ako vidíme, *n*-tica aj pole majú určité obmedzenia aj napriek tomu, že v mnohých prípadoch je ich použitie optimálnejšie hlavne vzhľadom na pamäťovú nenáročnosť.[77] Najuniverzálnejší typ je však zoznam (**`list`**), s ktorým budeme pracovať my a práca s ním je najjednoduchšia, odporúčame ho preto z didaktických dôvodov.

Aj keď budeme pracovať vlastne so zoznamom, **budeme mu hovoriť pole.**

[76] *Samozrejme, okrem znakového reťazca, to je tiež predsa postupnosť – postupnosť znakov.*
[77] *„Okolo tejto problematiky" je toho viac (napr. rozdiel medzi zásobníkom a radom). My sa tým však zaoberať nebudeme.*

9.2 Rezy polí

Keďže je to postupnosť, na prístup k položkám **používame rovnaké indexovanie ako v prípade textového reťazca.**

Vyberanie, spájanie alebo **modifikovanie** poľa pomocou indexovania nazývame **rez poľa** (angl. slice). Poďme si to vyskúšať. Majme takéto pole:

```
pole = ["Peter", 14.7, "Milan", "december", 245, [3, 6, 7], abs(-45)]
```

Indexovanie je nasledujúce:

znak	`"Peter"`	`14.7`	`"Milan"`	`"december"`	`245`	`[3, 6, 7]`	`abs(-45)`
index	0	1	2	3	4	5	6
index	-7	-6	-5	-4	-3	-2	-1

Potom napríklad:

takýto zápis:	*má takúto hodnotu:*
`pole.index("Milan")`	`2`
`len(pole)`	`7`
`pole[5]`	`[3, 6, 7]`
`pole[-4]`	`'december'`
`pole[-1]`	`45`
`pole[2:4]`	`['Milan', 'december']`
`pole[::-1]`	`[45, [3, 6, 7], 245, 'december', 'Milan', 14.7, 'Peter']`
`pole[3] = "január"`	`['Peter', 14.7, 'Milan', 'január', 245, [3, 6, 7], 45]`
`pole[5][2]`	`7`
`pole[5][1] = "nič"`	`['Peter', 14.7, 'Milan', 'január', 245, [3, 'nič', 7], 45]`
`pole = pole + ["nový"]`	`['Peter', 14.7, 'Milan', 'január', 245, [3, 'nič', 7], 45, 'nový']`
`pole = pole[:5] + [85] + pole[5:]`[78]	`['Peter', 14.7, 'Milan', 'január', 245, 85, [3, 'nič', 7], 45, 'nový']`

Ako vidíme, zoznam môže obsahovať naozaj ľubovoľné údajové typy, dokonca aj aritmetické, relačné a logické výrazy, ktoré sa do poľa ale ukladajú vyčíslené (ako sme mohli vidieť vyššie – `abs(-45)` sa uložilo ako `45`, `15 + 5` by sa uložilo ako `20` atď.). Dokonca, zoznam môže obsahovať ďalší zoznam (aj toto budete možno neskôr využívať), ku ktorému sa potom pristupuje tzv. **dvojitým indexovaním** (`pole[index1][index2]`).

Tiež, ako vidíme, sa dá veľmi jednoducho meniť hodnota ktorejkoľvek položky, **čo sa na rozdiel od textového reťazca nedalo,** preto sme hovorili iba o indexovaní, **modifikácia poľa indexovaním** sa nazýva **rez.**

PRÍKLAD 42

Vráťme sa k programu, ktorý vypisoval hody kockou. Vedeli by ste, na základe toho, čo ste sa dozvedeli o poliach, zefektívniť tento program? Skúste porozmýšľať sami a potom sa pozrite na možné riešenie.

```
1.  import random
2.
3.  a = [0, 0, 0, 0, 0, 0]
4.
5.  for i in range (3000):
6.      x = random.randrange(1, 7)
7.      a[x-1] = a[x-1] + 1
```

[78] *Funkcie ako* `index()`, `max()`, `min()`, `sort()`, `remove()` *a iné v hlavnom učive nespomínané odporúčame* ***zakázať žiakom používať, a to aj na maturitnej skúške.*** *V počiatočných fázach učenia sa programovať má ísť o rozvoj algoritmického myslenia a nie o to „použiť hotové bez rozmýšľania“. V nasledujúcej kapitole si totiž na toto vytvoríme svoje vlastné funkcie, čo bude ťažiskom úloh v kapitole o podprogramoch. Celkovo odporúčame používať výhradne funkcie, ktoré spomíname v učebnici a ktoré vyslovene odporúčame používať.*

```
8.
9.  for i in range(6):
10.     print(f"Číslo {i+1} padlo {a[i]}-krát.")
11.
12. print("Celé pole:", a)
```

Na začiatku si definujeme 6-prvkové pole šiestich núl. Následne v cykle generujeme náhodné celé číslo v intervale ⟨1; 6⟩. Podľa toho, aké je to číslo, na $x - 1$. pozíciu pripočítame jednotku (ak padla dvojka, prvej pozícii v poli pripočítame jednotku, ak padla jednotka, nultej pozícii v poli pripočítame jednotku atď.). Následne, aby sme nemuseli „manuálne“ vypisovať informáciu o tom, koľkokrát padlo ktoré číslo, dáme výpis do cyklu **for** so 6 opakovaniami a k hodnotám v poli pristupujeme už známym indexovaním.

ÚLOHA 21

Zamyslite sa, aký je rozdiel medzi týmito dvoma príkazmi. Porozmýšľajte a potom si to overte na vhodnom príklade v Pythone:

```
pole = pole + [položka]
pole[index] = pole[index] + položka
```

PRÍKLAD 43

Učiteľ matematiky potrebuje vybrať žiakov z triedy na matematickú súťaž s nadpriemernou známkou. Nasimulujte túto situáciu – vygenerujte n známok. Vypočítajte priemernú známku. Na základe priemernej známky vyberte tie známky žiakov, ktoré sú lepšie ako priemer a nechajte ich vypísať na shell.

```
1.  import random
2.  počet = 50
3.  súčet = 0
4.  známky = []
5.
6.  for i in range(počet):
7.      známky += [0]
8.
9.  for i in range(počet):
10.     známky[i] = random.randrange(1, 6)
11.     súčet += známky[i]
12.
13. priemer = súčet/počet
14. print("Priemer je", priemer)
15. print(známky)
16. print("Matematická súťaž:")
17.
18. for i in range(počet):
19.     if známky[i] < priemer: print(známky[i], end=", ")
20.
21. print()
```

PRÍKLAD 44

Nasimulujte priebeh žrebovania čísel hry Loto 5 z 35 [13]. V osudí je 35 loptičiek s číslami 1 – 35, vyberte 5 z nich bez vracania (vybraté čísla sa nesmú opakovať). Týchto 5 čísel nechajte vypísať na shell. (Asi vás to bude navádzať na použitie funkcie **remove()** alebo **index()** – zdôrazňujeme, nepoužívajte ich! Určite to budete vedieť aj bez nich.)

Jedno z možných riešení:

```
1.  import random
2.  pole = []
3.  výhra = []
4.
5.  for i in range(1, 36):
6.      pole += [i]
7.  print(pole)
8.
9.  for i in range(5):
10.     poradie = -1
11.     index = 0
12.     číslo = random.choice(pole)
```

```
13.     výhra += [číslo]
14.
15.     while poradie == -1:
16.         if pole[index] == číslo:
17.             poradie = index
18.         else:
19.             index += 1
20.
21.     pole = pole[:index] + pole[index+1:]
22. print("výhra", výhra)
23. print("skrátené pole", pole)
```

Druhé možné riešenie:

```
1.  import random
2.  pole = []
3.  výhra = []
4.
5.  for i in range(1, 36):
6.      pole += [i]
7.  print(pole)
8.
9.  for i in range(5):
10.     poradie = -1
11.     index = 0
12.     číslo = random.choice(pole)
13.     výhra += [číslo]
14.
15.     while poradie == -1:
16.         if pole[index] == číslo:
17.             poradie = index
18.         index += 1
19.
20.     pole = pole[:index-1] + pole[index:]
21. print("výhra", výhra)
22. print("skrátené pole", pole)
```

Úloha 22

Riešenia predchádzajúceho príkladu sa odlišujú len minimálne. Nájdite, uvedomte si a zdôvodnite zmeny.

Skúste úlohu riešiť tak, aby pole čísel od 1 do 35 ostalo nezmenené. Napríklad takto:

- vyber náhodné číslo od 1 do 35, pridaj ho do poľa
- pri výbere ďalšieho náhodného čísla prejdi pole náhodných čísel a skontroluj, či sa tam dané číslo už nenachádza, ak áno, vygeneruj nové číslo
- toto opakuj dovtedy, kým nebude v poli výherných čísel 5 rôznych čísel

Alebo tak, že nebudete meniť dĺžku poľa, iba toto číslo z poľa „vytiahnete“

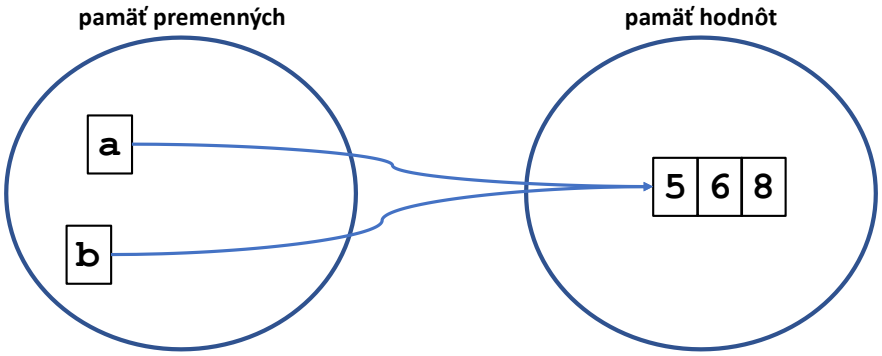
- vyber náhodné číslo od 1 do 35, pridaj ho do poľa
- prepíš vybraté číslo z poľa všetkých čísel inou hodnotou (napríklad reťazcom `"X"`)
- vyberaj ďalšie číslo z poľa (generuj nový index), ak sa na tomto mieste nachádza `"X"`, vygeneruj ďalší náhodný index
- toto opakuj dovtedy, kým nebude v poli výherných čísel 5 rôznych čísel

9.3 Kopírovanie poľa

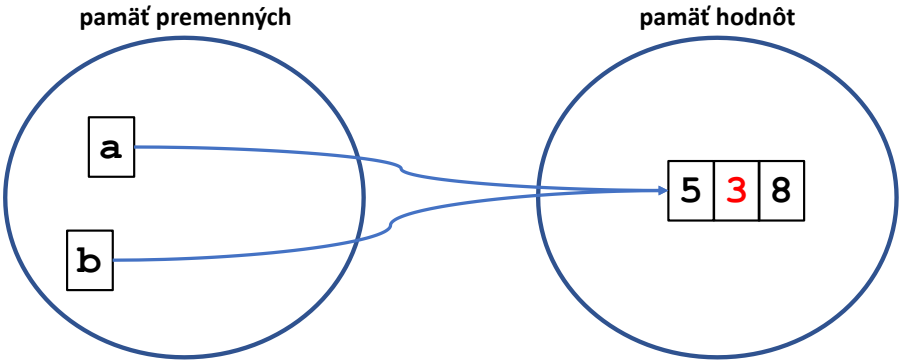
Asi si hovoríte – načo sa tu venujeme kopírovaniu poľa, keď to je také jednoduché – veď si predsa vytvorím pole, priradím ho do premennej a ak chcem kópiu, tak premennú priradím zas ďalšej – novej premennej tak, ako to bolo doteraz a ako sme boli zvyknutí. Naozaj? Vyskúšajme si to v interaktívnom režime – v shelli. Vytvorme si nové pole, uložme ho do premennej. Potom si do novej premennej uložme „starú“ premennú. V novej premennej zmeňme nejaký prvok poľa za iný. Potom si dajme vypísať hodnotu prvej a hodnotu druhej premennej. Asi vás to „nemilo prekvapí“, že sa zmenili „obidve“ polia, keď sme menili iba jedno.

```
>>> a = [5, 6, 8]
>>> b = a
>>> b[1] = 3
>>> b
    [5, 3, 8]
>>> a
    [5, 3, 8]
>>>
```

Na to existuje pomerne jednoduché vysvetlenie a spôsob, ako tomuto zamedziť. Spomeňte si, ako sme sa „hrali“ s pamäťou identifikátorov a pamäťou premenných na úvode učebnice, keď sme sa učili, ako sa priraďuje hodnota do premennej. Pole po svojom vytvorení v operačnej pamäti[79] vyzerá asi takto:



„Nebezpečné“ pri poliach je, že konkrétne toto pole má dve referencie. Po zmene prvku poľa sa stane toto:



Prečo sa však nevytvorilo nové pole ako pri bežných premenných? Je to z dôvodu optimalizácie. Predstavte si, že by sme mali pole obrázkov s dĺžkou 20-tisíc (predstavme si pod tým napríklad fotky stiahnuté z fotoaparátu v priečinku v počítači). Bolo by veľmi neefektívne, keby sa z dôvodu zmeny jednej fotky museli všetky ostatné skopírovať do pamäte druhý raz. (Toto však platí, len ak nemeníme dĺžku poľa a meníme len jeho existujúce hodnoty. Skúste poľu pridať prvok a uvidíte sami, čo sa stane.)

Ak chceme pole skutočne skopírovať, musíme ho „prinútiť“ zduplikovať svoj obsah aj v pamäti hodnôt. Ako? Napríklad pomocou cyklu `if`, alebo tak, že novému poľu dáme „akože“ nový prvok (pričom mu nedáme nič). Ešte lepšie je jednoducho použiť príkaz `b = a[:]`.

```
>>> a = [5, 6, 8]
>>> b = a + []
>>> b[1] = 3
>>> b
[5, 3, 8]
>>> a
[5, 6, 8]
>>>
```

9.4 Bublinkové triedenie

Často budeme potrebovať zoradiť prvky poľa (väčšinou čísel) od najmenšieho po najväčšie.

Majme pole čísel: `čísla = [4, 8, 12, 47, -14, 1, 0, 9, -3]`. Keby sme niekomu povedali, aby ich zotriedil od najmenšieho po najväčší, určite by to bez problémov zvládol – metódou „pozriem-vidím“. Toto však takto počítaču povedať nemôžeme. Ako teda?

[79] *Často sa operačnej pamäti nesprávne hovorí „iba“ RAM. RAM je však akákoľvek pamäť s priamym prístupom (teda aj pevný disk, v súčasnosti skoro všetky pamäťové médiá). Aj keď každá operačná pamäť je typu RAM, nie každá pamäť RAM je operačnou pamäťou.*

Predstavme si situáciu, že sú tieto čísla napísané na kartičkách a tie sú otočené naopak – tak, že nevidíme, aké číslo na kartičkách je. Vašou úlohou by bolo postupne po dvoch susedných kartičkách otáčať a rozhodovať, ktoré z dvoch čísel je väčšie. Zoberieme si prvú a druhú kartičku. Ak by bolo väčšie to druhé, kartičky iba otočíme naspäť a vezmeme si druhú a tretiu. Ak by bolo väčšie to prvé, kartičky medzi sebou prehodíme a pokračujeme ďalej. Takto sa dostaneme až na koniec a o poslednom čísle dokážeme povedať, že je najväčšie.

Takto prechádzame zoznam čísel opäť, tentokrát však o jedenkrát menej, pretože posledné číslo je už tam, kde má byť (najväčšie je na konci). Dovtedy, kým zoznam čísel nevytriedime.

Takýto algoritmus dostal názov **bublinkové triedenie** (angl. bubble sorting) vďaka tomu, že väčšie prvky „vystúpia na povrch" ako bublinky v pohári. Predtým, ako sa pustíme do jeho naprogramovania, odporúčame pustiť si toto video [14]:[80]

https://www.youtube.com/watch?v=yIQuKSwPlro&ab_channel=KCAng

PRÍKLAD 45

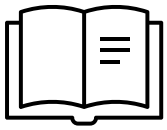
Skúsme to naprogramovať a necháme si aj vypisovať priebeh triedenia na shell:

```
1.  čísla = [4, 8, 12, 47, -14, 1, 0, 9, -3]
2.
3.  print("PÔVODNÉ POLE:", čísla)
4.
5.  for i in range(len(čísla) - 1):
6.      for j in range(len(čísla) - i - 1):
7.          if čísla[j] > čísla[j+1]:
8.              čísla[j], čísla[j+1] = čísla[j+1], čísla[j]
9.
10.         print("  Postup:", čísla)
11.     print("Ďalší cyklus:")
12.
13. print("ZOTRIEDENÉ POLE:", čísla)
```

ÚLOHA 23

Naprogramujte animáciu bublinkového triedenia. Nech počítač náhodne vygeneruje 10 hodnôt. Vykreslite tieto hodnoty na canvas v podobe stĺpčekov s rôznou (príslušnou) veľkosťou (niečo ako stĺpcový graf). Nech sa po každom kliknutí na tlačidlo vykoná jeden (ďalší) krok algoritmu – napr. animácia výmeny stĺpčekov, aktualizácia informácie o porovnávaní atď. Nech sa stĺpčeky, ktoré sú už zotriedené, zafarbia (napr. zelenou) farbou. Nech sa na záver algoritmu na canvas vypíše, že triedenie je ukončené.

Pripomíname, že triediacich algoritmov je hneď niekoľko a bublinkové triedenie je jedným z najhorších (najneefektívnejších), avšak didakticky najnázornejších a najjednoduchších, preto si myslíme, že žiakovi na úrovni maturanta znalosť princípu tohto triedenia bohato stačí, keďže triediace algoritmy sa od maturantov ani nevyžadujú.



ZHRNUTIE

- ✓ Pole, zoznam a n-tica sú údajové typy, ktoré umožňujú uchovávať niekoľko hodnôt naraz vo forme usporiadanej n-tice (členy, kde záleží na poradí), teda vo forme nejakej iterovateľnej postupnosti.
- ✓ Zoznam je meniteľný a môže uchovávať viac ako jeden typ údajov.
- ✓ Pole môže uchovávať iba rovnaké typy údajov a n-tica je nemenná.
- ✓ V prípade prístupu k hodnotám (členom) poľa používame rovnaký spôsob indexovania ako v prípade textového reťazca.

[80] Pozor, niektoré videá na YouTube neznázorňujú úplne správny postup bubble sortingu, považujú totiž niekedy za vytriedené rovno dva/tri prvky naraz, pamätajme však, že za vytriedený prvok sa považuje v každom cykle len jeden ďalší od konca – ako vo videu na odkaze.

- ✓ Vyberanie, spájanie alebo modifikovanie poľa pomocou indexovania nazývame rez poľa.
- ✓ Ak pristupujeme k prvku poľa, ktorým je ďalšie pole, nazývame to dvojité indexovanie a zapisujeme: `pole[index1][index2]`.
- ✓ Na rozdiel od textového reťazca môžeme jednoducho meniť hodnotu ktorejkoľvek položky.
- ✓ Ak chceme pole skopírovať do inej premennej, musíme ho buď „prehnať" cyklom a ukladať po prvkoch alebo novému poľu pridať prázdny prvok (prázdne hranaté zátvorky – `[]`), alebo novej premennej priradiť „staré pole od začiatku do konca" pomocou rezu `[:]`.
- ✓ Bublinkové triedenie funguje tak, že vo vnorenom cykle porovnávame vždy dvojicu susedných hodnôt poľa a ak je prvá hodnota väčšia ako druhá, hodnoty medzi sebou vymeníme. Na konci každého jedného vnoreného cyklu vždy nájdeme ďalšie a ďalšie najväčšie číslo a vnorený cyklus sa potom opakuje vždy o jedenkrát menej.
- ✓ Bublinkové triedenie je z hľadiska efektívnosti najhorší triediaci algoritmus.

OTÁZKY NA ZOPAKOVANIE

1. Porovnajte údajové typy pole, zoznam a n-tica, nájdite rozdiely a spoločné znaky.
2. Čo je to rez poľa?
3. Opíšte spôsob indexovania poľa.
4. V čom je hlavný rozdiel medzi pojmami rez a indexovanie?
5. Aký je rozdiel medzi zmenou položky (hodnoty alebo znaku) v poli a v textovom reťazci? Demonštrujte na konkrétnom príklade.
6. Ako by ste pristúpili k hodnote `18`? `pole[2, 5, [7, 3, 9], [14, 11, 18, 22], 15]`
7. Ako túto hodnotu `18` z poľa vymažete (len použitím rezu, bez vstavaných funkcií)?
8. Vykonajte kópiu poľa na vlastnom príklade. Dokážte, že je to kópia a nie to isté pole s dvoma referenciami.
9. Podrobne opíšte a demonštrujte na príklade bublinkové triedenie (napr. napísaním čísel na kartičky).

10 Podprogramy

Ciele

V tejto kapitole sa žiak oboznámi s funkciou a spôsobom fungovania podprogramov, osvojí si základnú terminológiu podprogramov, bude vedieť opísať stavy premenných (trasovať ich) pred zavolaním, počas volania a po volaní podprogramu, definovať si vlastné podprogramy – na prácu s textovým reťazcom, poľom, na konverziu čísel a podobne a dozvie sa, ako funguje rekurzia. Žiak bude tiež vedieť oceniť zjednodušenie práce použitím podprogramov a vďaka tomu si odteraz vedieť rozložiť komplexnejšiu úlohu na menšie časti.

Asi ste si už uvedomili pri programovaní Lota 5 z 35, že sa vždy vykoná istá skupinka príkazov, ktorá zabezpečí, aby sa číslo neopakovalo, alebo pri poslednom programe ste si uvedomili, že, hoci sme teraz programovali bublinkové triedenie, existuje na to aj funkcia **`sort()`**, ktorú však máme „zakázanú“ používať. Tá však musí tiež nejako fungovať, táto funkcia (a všetky ostatné, samozrejme) musí niečo obsahovať, tiež istý súbor príkazov, ktoré vykonávajú to, čo od tej funkcie očakávame. Áno, je to tak. Už sme používali niekoľko funkcií, napr. **`input()`**, **`print()`**, ktoré v sebe obsahujú množstvo príkazov, ktoré zabezpečujú načítanie a výpis hodnôt.[81] Funkcia **`randrange()`** v sebe obsahuje pomerne rozsiahle množstvo zložitých príkazov, ktoré zaručujú, že ich výsledkom je vždy iné číslo. Aritmetická funkcia **`abs()`** určuje absolútnu hodnotu čísla, **`sin()`** a **`cos()`** zas obsahujú príkazy na výpočet hodnoty sínus a kosínus a podobne. Mnohé z nich sú vstavané, iné bolo treba importovať, pretože boli definované v iných moduloch a pristupovať k nim takzvanou **bodkovou notáciou.**[82] Napríklad:

`import random` umožňoval pracovať s **`random.randrange()`** a **`random.choice()`**
`import math` umožňoval pracovať s **`mat.sin()`**, **`math.cos()`**, **`math.radians()`**
`import tkinter` umožňoval pracovať s **`tkinter.Canvas()`**

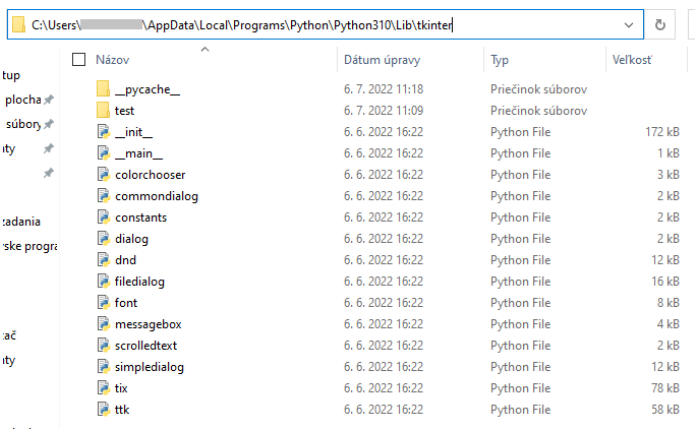
a podobne.

Tiež si spomeňte, že keď sme kreslili na plátno a priraďovali akciu tlačidlu, tak to, čo sme „zabalili“ (teda odsadili od kraja) za konštrukciu **`def akcia_tlačidla()`**, bol tiež podprogram (funkcia). V tejto kapitole sa tieto podprogramy naučíme vytvárať.

Čo je to ten podprogram? **Podprogram je časť kódu „schopná samostatnej existencie“ (t. j. sama o sebe dokáže niečo vykonať), je to blok príkazov, ktorý je nejako pomenovaný.**

V Pythone sa každý podprogram nazýva funkcia,[83] v iných jazykoch sa rozlišujú funkcie a procedúry. Podprogram je definovaný buď priamo v kóde programu (ak je to náš vlastný), alebo je **vstavaný** (tzv. built-in, teda je súčasťou „základného Pythona“ a je zafarbený fialovou farbou), **alebo sa takzvane importuje z knižnice** (ako sme importovali moduly **`math`**, **`random`** a **`tkinter`**).

Knižnicu si predstavme ako **priečinok, v ktorom sú uložené jednotlivé pythonovské kódy** (programy), kde každý jeden takýto súbor (program) je vlastne podprogramom. Keď importujeme nejaký modul (napríklad **`math`**), Pythonu tým vlastne (zjednodušene) povieme, aby otvoril priečinok **`math`** a vedel použiť funkcie, ktoré sa v priečinku nachádzajú (**`sin()`**, **`cos()`**, **`radians()`** atď.). Na obrázku môžeme napríklad vidieť obsah priečinka **`tkinter`**, umiestnený v inštalačnom priečinku prostredia IDLE Python. Čiže **`import`**-om rozhrania **`tkinter`** vlastne otvoríme priečinok a „dovolíme“ Pythonu používať funkcie (podprogramy) v ňom.



[81] *Nielen na konzolu – na shell, k tomu sa dostaneme v ďalšej kapitole.*
[82] *Bodková notácia je charakteristická v objektovo orientovanom programovaní – ním sa však v tejto učebnici zaoberať nebudeme.*
[83] *Aby vás to nepomýlilo s definíciou funkcie v matematike.*

10.1 Názov, definícia, premenné, volanie, návratové hodnoty

Každý podprogram má svoj **názov, definíciu, parametre, premenné a návratové hodnoty** a v Pythone vyzerá takto:

```
def názov_podprogramu(p1, p2, p3):
    príkaz 1
    d = p2 * p3
    príkaz 2
    return d
```

Názov podprogramu sa píše za ďalšie „nové“ dohovorené slovo `def`, za ním nevyhnutne nasleduje zátvorka (aj keď je podprogram bez parametrov) a dvojbodka.

Definícia podprogramu sú všetky príkazy, ktoré nasledujú za konštrukciou `def` a sú odsadené na rovnakú úroveň zľava, je to teda **podprogram samotný** – kód, ktorý sa po využití stáva kúskom, časťou iného programu.

Volanie podprogramu: počas vykonávania hlavného programu sa v mieste, kde je to potrebné, použije funkcia (podprogram), a teda sa tam napíše riadok odkazujúci na jej použitie (**teda sa zavolá**), aby vykonala svoju úlohu. Preto sme skôr upozorňovali na to, aby ste **nepoužívali termín volanie v súvislosti spomínania názvov premenných.** Ak chceme vyjadriť, že funkcia alebo premenná majú svoje pomenovanie, musíme povedať, že sa nazývajú. Ak však chceme povedať, že ideme funkciu niekde použiť, povieme, že funkcia sa volá, teda, že ju zavoláme (v tej-ktorej časti programu).[84] Napríklad:

```
a = input("Zadaj: ")
príkaz 1
príkaz 2
názov_podprogramu(a, a*3, 14) # tu sa funkcia zavolá
príkaz 3
príkaz 4
...
```

Vidíme, že v hlavnom programe sa vykoná **`príkaz 1`**, **`príkaz 2`**, následne **sa zavolá** funkcia, kde sa následne hodnoty **`a`**, **`a*3`** a **`14`** takzvane **privážajú** do parametrov **`p1`**, **`p2`** a **`p3`**, funkcia vykoná, čo sa od nej očakáva a nasleduje **`príkaz 3`** a **`príkaz 4`**.

Typy parametrov:

- **`a`**, **`a*3`** a **`14`** sú tzv. **skutočné parametre,** teda hodnoty, ktoré sa **privážajú** podprogramu pri jeho **volaní**
- **`p1`**, **`p2`** a **`p3`** sú tzv. **formálne parametre** alebo tzv. **lokálne premenné,** s ktorými pracuje iba podprogram, k lokálnym premenným patrí aj **`d`**, ktoré však **nie je parametrom**
- existujú aj funkcie **bez parametrov –** tie sa **musia definovať aj volať s prázdnymi zátvorkami**[85]

Premenné z pohľadu podprogramov:

- **lokálne premenné** – to sú také, ktoré **existujú len v rámci jednej jedinej funkcie** (podprogramu), **vzniknú pri jej zavolaní a zaniknú po jej vykonaní.** Z iných častí programu teda k lokálnym premenným **neexistuje prístup.**
- **regulárne premenné** – sú premenné, ktoré **sú definované v hlavnom programe a**[86] **podprogramu nie sú prístupné** – tie s nimi nevedia pracovať. Keby sme mali napríklad v hlavnom programe definovanú premennú **`d`**, bola by to úplne iná premenná **`d`** ako tá lokálna z podprogramu **`názov_podprogramu`**[87]

---

[84] *Po anglicky: názov funkcie = function name, volanie funkcie = function calling.*

[85] *Výnimkou sú snáď len akcie tlačidla, tam sa funkcia volá bez zátvoriek. Tam je však iný dôvod, ktorý nepovažujeme za potrebné rozoberať.*

[86] *... podprogramu sú (v špeciálnych prípadoch)* ***prístupné iba na čítanie*** *(„read only“) a podprogramy s nimi nepracujú – majme však „za svoje“, aby nás to zbytočne nepoplietlo, že ...*

[87] *Odporúčame zatiaľ „nemýliť“ žiakov tým, že regulárne premenné sú prístupné iba na čítanie, my sa k tomu dostaneme pri tvorbe hry, kde to budeme potrebovať, teraz je cieľom naučiť sa korektne vytvárať podprogramy s privážaním parametrov a uchovávaním návratových hodnôt a nie „spoliehať sa na to“, že si podprogram „kdesi z programu zoberie“ premennú, „ktorá sa mu hodí“.*

- **globálne premenné** – sú premenné, ktoré **majú platnosť celý čas v celom programe, vrátane vybraných podprogramov.** Ak chceme z regulárnej premennej spraviť globálnu, musíme pred názov premennej napísať dohovorené slovo `global` **v každom podprograme, v ktorom chceme, aby bol k nej prístup.**[88] Použitie globálnych premenných sa však neodporúča, **správny postup je definovať premenné s parametrom a privážať hodnoty tam.** Inak ľahko stratíme kontrolu nad tým, akú premennú, kedy a kde ktorý podprogram mení. Preto sa globálnym premenným v tejto učebnici veľmi venovať nebudeme (ale ukážeme si, ako ich použiť, pri vhodnom príklade).[89]

Návratová hodnota:

- **návratová hodnota je výsledok, ktorý funkcia takzvane vráti** (vypočíta a „vypľuje“) **po jej vykonaní a je možné ho uložiť do premennej.** Nám známou funkciou s návratovou hodnotou je funkcia `input()`. Tento príkaz: `a = input("Zadaj: ")` zavolá funkciu `input()`, ktorej návratová hodnota je to, čo prečíta zo shellu, kde sme niečo napísali po stlačení klávesu enter a **uloží túto návratovú hodnotu do premennej** `a`.[90] Pri definícii podprogramu s návratovou hodnotou **použijeme dohovorené slovo** `return`.[91] Je to príkaz, ktorý **vráti danú hodnotu** (`return x` = vráť x) a **následne ukončí podprogram, hoci by v ňom nasledovali ďalšie príkazy, podobne ako cyklus ukončuje** `break`, toto je veľmi dôležité si uvedomiť.

- **funkcia bez návratovej hodnoty** je funkcia, ktorá vykoná požadované príkazy, ale **výsledok „jej práce“ nie je konkrétna hodnota, nedá sa uložiť do premennej**, napríklad `print("Vypíš na shell.")`. Funkcia síce spraví to, čo má – vypíše text na konzolu, avšak je to činnosť, ktorej výsledok nie je hodnota, ale akcia, ktorá nie je „uložiteľná“ do premennej. Takéto podprogramy mnohé jazyky nazývajú **procedúry.**

Poďme si toto všetko konečne ukázať a pomenovať na konkrétnych príkladoch.

PRÍKLAD 46

Začnime posledným. Definujme si funkciu `zotrieď(pole)`, ktorá po zavolaní vráti zotriedené pole.

```
1.  def zotrieď(pole):
2.      dĺžka = len(pole)
3.      for i in range(dĺžka - 1):
4.          for j in range(dĺžka - i - 1):
5.              if pole[j] > pole[j+1]:
6.                  pole[j], pole[j+1] = pole[j+1], pole[j]
7.
8.      return pole
9.
10. čísla = [4, 8, 12, 47, -14, 1, 0, 9, -3]
11.
12. print("PÔVODNÉ POLE:", čísla)
13.
14. čísla = zotrieď(čísla)
15.
16. print("ZOTRIEDENÉ POLE:", čísla)
```

Poďme si to postupne rozobrať. **Funkcie je potrebné definovať vždy na začiatku programu,** pretože program ide postupne – **nemôže odkazovať/odvolať sa na niečo, čo ešte neexistuje** (nedá sa zavolať funkcia, ktorá nie je najskôr definovaná).[92]

Definovali sme funkciu `zotrieď(pole)`. Podprogram **sa nazýva** `zotrieď` a **volá sa** po výpise pôvodného poľa (na riadku 14). Pri volaní sa **do formálneho parametra** (lokálnej premennej) `pole` podprogramu

[88] *Prístup už teraz nie v zmysle iba na čítanie ako to je pri regulárnych premenných, ale aj na modifikáciu premennej.*
[89] *Pri tvorbe hry Logik.*
[90] *Resp. vytvorí sa v pamäti hodnôt a prepojí sa referenciou takisto, ako napríklad* `a = 5`*; ale to nie je teraz podstatné rozoberať.*
[91] *Z angličtiny return = návrat, teda návratová hodnota.*
[92] *Nedá sa len v procedurálnom programovaní – keď sa však dostaneme k udalostiam plátna, „načrieme“ aj do udalosťami riadeného programovania (tzv. event-driven programming), ktoré sa neriadi „hlavným prúdom“ príkazov, ale reaguje na udalosti, odporúčame však toto zatiaľ nespomínať a nemýliť žiakov – stále sme „len“ v procedurálnom programovaní, kde to takto nefunguje.*

priváža hodnota, na ktorú odkazuje premenná `čísla`, ktorá je **regulárnou premenou** a v tomto prípade aj **skutočným parametrom.** Tým sa podprogram **aktivuje, vznikne tam lokálna premenná `pole`**, ktorá je zároveň **formálnym parametrom.** Následne tam **vznikne lokálna premenná `dĺžka`** a podprogram ďalej vykonáva svoju činnosť – triedenie. Nasleduje príkaz **`return pole`**, ktorý takzvane **vráti** zotriedené **`pole`** a **podprogram sa deaktivuje** (ukončí svoju činnosť a **všetky lokálne premenné zaniknú**). **Vrátená hodnota** sa v hlavnom programe **priradí regulárnej premennej `čísla`**, čím (a to vám už musí byť známe) premenná **`čísla`** zruší referenciu na pôvodné pole a teraz referuje na už zotriedené pole. Ako posledné, program toto zotriedené pole vypíše na shell.

Všimnime si, že **podprogram môže volať ďalší podprogram** (v tomto prípade podprogram **`zotrieď(pole)`** zavolal podprogram **`len(pole)`**, ktorý mu vráti dĺžku zadaného reťazca). Dokonca, **podprogram môže volať aj sám seba, tomu sa hovorí rekurzia** alebo **rekurzívny podprogram.** Neskôr si ukážeme na rekurziu len veľmi jednoduchý príklad.[93]

Ešte jedna vec – uvedomme si, že program sa vlastne začína **až riadkom 10,** funkcia je tam až do jej zavolania len „uskladnená" a neaktívna.

PRÍKLAD 47

Poďme skúsiť **funkciu s dvomi parametrami.** Rozšírme existujúcu funkciu o ďalší parameter, ktorý bude prijímať hodnoty **`True`** alebo **`False`**. Ak prijme **`True`**, zoradí hodnoty v poli vzostupne, ak **`False`**, zoradí ich zostupne. Skúste najskôr sami a potom sa pozrite na možné riešenie.

```
1.  def zotrieď(pole, rast):
2.      dĺžka = len(pole)
3.      if rast:
4.          for i in range(dĺžka - 1):
5.              for j in range(dĺžka - i - 1):
6.                  if pole[j] > pole[j+1]:
7.                      pole[j], pole[j+1] = pole[j+1], pole[j]
8.      else:
9.          for i in range(dĺžka - 1):
10.             for j in range(dĺžka - i - 1):
11.                 if pole[j] < pole[j+1]:
12.                     pole[j], pole[j+1] = pole[j+1], pole[j]
13.     return pole
14.
15. čísla = [4, 8, 12, 47, -14, 1, 0, 9, -3]
16.
17. print("PÔVODNÉ POLE:", čísla)
18.
19. čísla = zotrieď(čísla, True)
20. print("ZOTRIEDENÉ POLE vzostupné:", čísla)
21.
22. čísla = zotrieď(čísla, False)
23. print("ZOTRIEDENÉ POLE zostupné:", čísla)
```

ÚLOHA 24

Okomentujte priebeh programu podobne ako v predchádzajúcom príklade. Snažte sa uvedomiť si **typy premenných, kedy sa ktorá ktorej priraďuje, kedy čo vzniká a zaniká, návratovú hodnotu aj spôsob volania podprogramu.**

ÚLOHA 25

Pokúste sa sami zostaviť vývojový diagram tohto programu. Dodávame, že vo vývojovom diagrame sa podprogram označuje obdĺžnikom s dvojitými zvislými čiarami:

[93] *Hlbšie sa tomu venovať nebudeme a nepotrebujú to vedieť ani maturanti.*

PRÍKLAD 48

V programe Loto 5 z 35 sme v cykle zisťovali, na ktorom indexe sa nachádza náhodne vybraté číslo. Definujte na toto zisťovanie podprogram **`kde_to_je(čo, pole)`**, ktorý bude prijímať parametre:

- **`čo`**: čo (akú položku) hľadáme
- **`pole`**: v akom poli

a návratovou hodnotou bude **`index`** (miesto, kde sa daná položka v poli nachádza).

Napríklad takto:

```
1.  import random
2.  pole = []
3.  výhra = []
4.
5.  def kde_to_je(čo, pole):
6.      poradie = -1
7.      index = 0
8.      while poradie == -1:
9.          if pole[index] == čo:
10.             poradie = index
11.         else:
12.             index += 1
13.     return index
14.
15.
16. for i in range(1, 36):
17.     pole += [i]
18. print(pole)
19.
20. for i in range(5):
21.     číslo = random.choice(pole)
22.     výhra += [číslo]
23.
24.     index = kde_to_je(číslo, pole)
25.
26.     pole = pole[:index] + pole[index+1:]
27. print("výhra", výhra)
28. print("skrátené pole", pole)
```

10.2 Vlastné podprogramy

ÚLOHA 26

Už sme spomínali, že funkcie ako **`index()`**, **`max()`**, **`min()`**, **`sort()`**, **`remove()`** nebudeme a nechceme používať preto, aby sme sa naučili algoritmicky myslieť a nie iba používať hotové funkcie.[94] Funkciu **`zotrieď(pole, rast)`** sme si už definovali vyššie. Vašou úlohou bude teraz samostatne definovať funkcie:

- **`kde_to_je(čo, pole)`** – túto funkciu sme tiež definovali, skúste ju však optimalizovať a ošetriť – nech, ak sa hodnota v poli nenájde, funkcia vráti hodnotu **`False`**
- **`maximum(pole)`** – vráti najväčší prvok poľa
- **`minimum(pole)`** – vráti najmenší prvok poľa
- **`vyhoď(index, pole)`** – odstráni položku poľa (pole tým skráti o jeden prvok)
- **`vyhoď_všetky(čo, pole)`** – nájde a vyhodí všetky výskyty z poľa; ak nenájde ani jeden výskyt, vráti pôvodné pole (pomôcka – oplatí sa v tomto prípade prehľadávať pole a vymazávať položky od konca)

PRÍKLAD 49

Definujte funkciu, ktorá zistí, či strany, ktoré prijme do parametrov, môžu byť stranami pravouhlého trojuholníka – ak áno, vráti výslednú hodnotu **`True`**, ak to neplatí, vráti výslednú hodnotu **`False`**. Ak ste zabudli, čo je trojuholníková nerovnosť a Pytagorova veta – vygooglite si to.

[94] *Neskôr – až keď si osvojíte a precvičíte všetko, o čom táto učebnica píše – keď s programovaním pokročíme ďalej, je, samozrejme, zbytočné „znova objavovať koleso" a odmietať funkcie používať – veď predsa na to sú.*

Riešenie:

```
1.  def pravouhlosť(x, y, z):
2.      if x*x + y*y == z*z or y*y + z*z == x*x or x*x + z*z == y*y:
3.          return True
4.      else:
5.          return False
6.
7.  def zostrojiteľnosť(x, y, z):
8.      if x <= 0 or y <= 0 or z <= 0 or x+y <= z or z+x <= y or z+y <= x:
9.          return False
10.     else:
11.         return True
12.
13. a = int(input("Zadaj stranu a: "))
14. b = int(input("Zadaj stranu b: "))
15. c = int(input("Zadaj stranu c: "))
16.
17. if zostrojiteľnosť(a, b, c):
18.     print("Trojuholník je zostrojiteľný.")
19.     if pravouhlosť(a, b, c):
20.         print("Trojuholník je aj pravouhlý.")
21. else:
22.     print("Trojuholník nie je zostrojiteľný.")
```

Mohli by sme tieto funkcie definovať aj jednoduchšie? Porozmýšľajte, ako funguje **`return`** (a aj **`break`**) a skúste z funkcie **vymazať jeden príkaz tak, aby sa ich správanie nezmenilo:**

```
1.  def pravouhlosť(x, y, z):
2.      if x*x + y*y == z*z or y*y + z*z == x*x or x*x + z*z == y*y:
3.          return True
4.      return False
5.
6.  def zostrojiteľnosť(x, y, z):
7.      if x <= 0 or y <= 0 or z <= 0 or x+y <= z or z+x <= y or z+y <= x:
8.          return False
9.      return True
```

Dá sa to predsa len ešte jednoduchšie. Uvedomte si, aký údajový typ nadobúdajú logické výrazy a z funkcií hneď viete odstrániť ďalšie dva riadky:

```
1.  def pravouhlosť(x, y, z):
2.      return x*x + y*y == z*z or y*y + z*z == x*x or x*x + z*z == y*y
3.
4.  def zostrojiteľnosť(x, y, z):
5.      return x <= 0 or y <= 0 or z <= 0 or x+y <= z or z+x <= y or z+y <= x
```

PRÍKLAD 50

Na základe toho, čo by ste už mali vedieť z teórie informatiky o prevode čísel medzi číselnými sústavami, definujte tri funkcie:

- **`z10do(sústava, číslo)`** – funkcia prevedie zadané číslo v desiatkovej sústave do zadanej sústavy
- **`do10z(sústava, číslo)`** – funkcia prevedie zadané číslo zo zadanej sústavy do desiatkovej sústavy
- **`zNdoN(sústava1, sústava2, číslo)`** – funkcia prevedie zadané číslo v jednej (zadanej) sústave do druhej (zadanej) sústavy

Ukážeme si možné riešenie prvých dvoch definícií:

```
1.  def z10do(sústava, číslo):
2.      výsledok = ""
3.      while číslo != 0:
4.          if číslo % sústava < 10:
5.              výsledok += str(číslo % sústava)
6.          číslo = číslo // sústava
7.          if číslo % sústava > 9:
8.              if číslo % sústava == 10: výsledok += "A"
9.              elif číslo % sústava == 11: výsledok += "B"
10.             elif číslo % sústava == 12: výsledok += "C"
11.             elif číslo % sústava == 13: výsledok += "D"
12.             elif číslo % sústava == 14: výsledok += "E"
13.             elif číslo % sústava == 15: výsledok += "F"
14.
15.     výsledok = výsledok[::-1]
16.     return výsledok
```

```
17.
18. def do10z(sústava, číslo):
19.     číslo = str(číslo)
20.     dĺžka = len(číslo)
21.     výsledok = 0
22.     počítadlo = 0
23.     for i in číslo:
24.         počítadlo += 1
25.         if i == "A": i = 10
26.         elif i == "B": i = 11
27.         elif i == "C": i = 12
28.         elif i == "D": i = 13
29.         elif i == "E": i = 14
30.         elif i == "F": i = 15
31.         výsledok += int(i)*(sústava**(dĺžka - počítadlo))
32.     return výsledok
```

Tretí podprogram je teraz možné „poskladať“ z týchto dvoch, asi takto:

```
33.
34. def zNdoN(sústava1, sústava2, číslo):
35.     medzivýpočet = do10z(sústava1, číslo)
36.     výsledok = z10do(sústava2, medzivýpočet)
37.     return výsledok
```

A takto môže vyzerať volanie a kontrola pomocou výpisu:

```
38.
39. a = z10do(2, 16)
40. b = do10z(2, 1011)
41. c = zNdoN(2, 16, 111110001)
42. print(a, b, c)
```

Z dôvodu úplnosti uvedieme príklad na volanie funkcie **bez parametrov a bez návratovej hodnoty:**

```
1.  import tkinter
2.  import random
3.
4.  canvas = tkinter.Canvas(width = 500, height = 500, bg = "yellow")
5.  canvas.pack()
6.
7.  def kresli_obdĺžnik(x, y):
8.      canvas.create_rectangle(x + 50, y + 50, x + 110, y + 110, fill = "red")
9.
10. def kresli_kruh():
11.     x = random.randrange(50, 449)
12.     y = random.randrange(10, 449)
13.     canvas.create_oval(x, y, x + 50, y + 50)
14.
15. kresli_obdĺžnik(10, 50) # volanie funkcie s parametrom a bez návratovej hodnoty
16. kresli_kruh()           # volanie funkcie bez parametra a bez návratovej hodnoty
```

Všimnime si, že ak voláme funkciu s návratovou hodnotou, je potrebné[95] túto návratovú hodnotu **uložiť do premennej** (veď preto sme ich volali, aby dali nejakú požadovanú hodnotu). Ak však voláme funkciu bez návratovej hodnoty (či s parametrami alebo bez nich), **nikam nič neukladáme** – „nemáme čo“ ukladať. Nie je to však „zakázané“, akurát premenná bude potom obsahovať tzv. **prázdny údajový typ** – `None`.[96]

Úloha 27

Opíšte a spomeňte čo najviac rozdielov medzi týmito dvomi volaniami:

- **`x.funkcia()`**
- **`funkcia(x)`**

[95] *Nie však nevyhnutné. Tomu sa však venovať nebudeme.*
[96] *V iných jazykoch sa možno stretnete s označením* **`null`** *alebo* **`nil`**.

10.3 Rekurzia

Už sme zistili, že jeden podprogram môže volať aj druhý podprogram. Podprogram však môže volať aj sám seba. Takýmto spôsobom sa dajú veľmi elegantne a oveľa jednoduchšie zapísať mnohé algoritmy (napríklad Euklidov algoritmus). Pripomeňme multiplikatívnu verziu tohto algoritmu, ktorú ste si mali sami naprogramovať [9]:

Je to postup, ktorého znenie je takéto: Najväčší spoločný deliteľ prirodzených čísel a a b, $a > b$ určíme takto:

1. Väčšie z čísel a a b nahradíme zvyškom po delení väčšieho čísla menším.
2. Ak menšie z čísel je nula, najväčším spoločným deliteľom je väčšie z čísel. Inak pokračujeme bodom 1.

Nechceme hneď prezrádzať riešenie, najskôr opíšeme algoritmus slovne:

```
a = 15
b = 50
ak a < b:
  výmena: a = 50, b = 15

kým sa b nerovná nule:
  c = 50%15, teda c = 5
  výmena: a = 15, b = 50
          b = c, teda b = 5

  c = 15%5, teda c = 0
  výmena: a = 5, b = 15
          b = c, teda b = 0 => KONIEC, chcem hodnotu a, teda a = 5
                                              teda NSD(50, 15) = 5
```

Pomocou rekurzie algoritmus vyzerá takto:

```
1.  def euklides(x, y):
2.      if y == 0:
3.          return x
4.      else:
5.          return euklides(y, x%y)
6.
7.  a = int(input("Zadaj číslo a: "))
8.  b = int(input("zadaj číslo b: "))
9.
10. print(f"Najväčší spoločný deliteľ je číslo {euklides(a, b)}.")
```

Zrejme je z tohto príkladu pomerne jasné, kedy sa program opäť sám zavolá a s akými hodnotami. Opíšeme princíp fungovania aj tu:

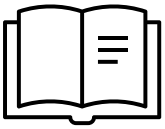
```
zavolá sa euklides(50, 15)
   ak y = 0: vráť x
   inak: vráť euklides(15, 50%15)

       zavolá sa euklides(15, 5)
          ak y = 0: vráť x
          inak: vráť euklides(5, 15%5)

              zavolá sa euklides(5, 0)
                 ak y = 0: vráť x
                 podmienka splnená, podprogram vráti x, x = 5,
                                                 čiže rekurzia sa končí

                                          teda euklides(50, 15) = 5
                                          teda NSD(50, 15) = 5
```

Diskutujte a formulujte, na základe skúseností a vedomostí, ktoré ste v tejto kapitole nadobudli, výhody použitia podprogramov. Tieto výhody uvádza slovenská Wikipédia [15]:

- redukuje sa duplicita kódu v programe
- možnosť opakovania kódu v zložených programoch
- rozloženie komplexných problémov na jednoduchšie časti
- vylepšenie čitateľnosti programu
- skrytie alebo regulovanie časti programu

Práve zručnosť a schopnosť rozloženia komplexného problému na jednoduchšie časti je „alfou a omegou" programátorského myslenia a algoritmizácie.

Ak sa vám zdá, že sme v kapitolách o poliach a podprogramoch veľa cvičení nespravili, je to tak „zámerne" – precvičovať budeme potom komplexnejšie zadania, kde bude „všehochuť" – všetko to, čo sme sa doteraz naučili, ale až potom, ako sa oboznámime **s možnosťou čítania a zápisu údajov do textových súborov,** aby sme nemuseli prácne zadávať veľké množstvo údajov.



ZHRNUTIE

✓ Podprogram je časť kódu „schopná samostatnej existencie", je to blok príkazov, ktorý je nejako pomenovaný.

✓ V Pythone sa každý podprogram nazýva funkcia, v iných jazykoch sa rozlišujú funkcie a procedúry.

- ✓ Podprogram je buď definovaný priamo v kóde programu alebo je vstavaný, alebo sa importuje z knižnice.
- ✓ Knižnica je priečinok, v ktorom sú uložené jednotlivé (pythonovské) programy a každý takýto súbor je vlastne podprogramom. Importovaním knižnice (modulu) sprístupníme programu používanie týchto funkcií.
- ✓ Názov podprogramu sa píše za dohovorené slovo `def`, za ním nevyhnutne nasleduje zátvorka (aj keď je podprogram bez parametrov) a dvojbodka.
- ✓ Definícia podprogramu sú všetky príkazy, ktoré nasledujú za konštrukciou `def` a sú odsadené na rovnakú úroveň zľava, je to teda podprogram samotný – kód, ktorý sa po využití stáva kúskom, časťou iného programu.
- ✓ Použitie podprogramu na mieste hlavného programu, kde je to potrebné, sa nazýva volanie podprogramu.
- ✓ Skutočné parametre sú hodnoty, ktoré sa privážajú podprogramu pri jeho volaní.
- ✓ Formálne parametre sú premenné, ktoré sa nachádzajú v konštrukcii podprogramu v zátvorke za názvom a pracuje s nimi len podprogram. Pri jeho zavolaní vzniknú a po jeho vykonaní zaniknú.
- ✓ Existujú aj funkcie bez parametrov – pri ich volaní nevyžadujú žiadne parametre, definovať aj volať sa však musia s prázdnymi zátvorkami.
- ✓ Lokálne premenné sú premenné, ktoré existujú len v rámci jednej jedinej funkcie, vzniknú pri jej zavolaní a zaniknú po jej vykonaní. Sú nimi aj formálne parametre.
- ✓ Regulárne premenné sú premenné, ktoré sú definované v hlavnom programe a podprogramom nie sú prístupné.
- ✓ Globálne premenné sú premenné, ktoré majú platnosť celý čas v celom programe, vrátane vybraných podprogramov. V každom podprograme, v ktorom chceme, aby bol k premennej prístup, musíme pred ňu napísať dohovorené slovo `global`.
- ✓ Návratová hodnota je výsledok, ktorý funkcia vráti po jej vykonaní a je možné ho uložiť do premennej. Aby sa tak stalo, podprogram musí obsahovať dohovorené slovo `return` a za ním premennú, ktorú treba vrátiť. To, čo nasleduje za príkazom `return`, sa už nevykoná.
- ✓ Ak je funkcia bez návratovej hodnoty, výsledok jej práce nie je hodnota, ktorá by sa dala uložiť do premennej. Takéto podprogramy sa v mnohých jazykoch nazývajú procedúry, v Pythone sa medzi týmito pojmami nerozlišuje a tiež sa nazývajú funkcie.
- ✓ V procedurálnom programovaní (v tom, do ktorého sme sa touto kapitolou dostali my) je funkcie potrebné definovať vždy na začiatku programu.
- ✓ Podprogram môže byť definovaný tak, aby pri jeho volaní volal ďalší podprogram.

- ✓ Ak podprogram volá sám seba, nazýva sa to rekurzia.
- ✓ Výhody použitia podprogramov sú redukcia duplicity kódu v programe, možnosť opakovania kódu, rozloženie komplexných problémov na menšie časti, vylepšenie čitateľnosti programu a regulovanie časti programu.

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem podprogram.
2. Opíšte rozdiel medzi funkciou a procedúrou.
3. Kde všade môžete „nájsť“ podprogram?
4. Čo je to knižnica alebo modul?
5. Čo je to definícia podprogramu?
6. Čo je to volanie podprogramu? Opíšte na vhodnom príklade, ako to celé prebieha.
7. Definujte, v akom sú vzťahu formálne a skutočné parametre, akú majú „životnosť“ a prístupnosť.
8. Porovnajte lokálne, regulárne a globálne premenné.
9. Čo je to návratová hodnota a ktorý príkaz ju zabezpečuje?
10. Volania ktorých podprogramov je vhodné uložiť do premennej a ktorých nie? Prečo?
11. Čo je to rekurzia?
12. Vymenujte čo najviac výhod použitia podprogramov.

11 Textový súbor

CIELE

V tejto kapitole sa žiak oboznámi s čítaním a zápisom v textovom súbore, bude schopný napísať programy pracujúce s obsahom textových súborov. Uvedomí si a bude vedieť oceniť praktickosť používania textových súborov ako trvalých uchovávateľov veľkého množstva údajov. Ďalším cieľom tejto kapitoly bude, aby si žiak precvičil a zautomatizoval všetky doteraz osvojené poznatky na príkladoch a úlohách, ktoré táto kapitola obsahuje.

V tejto (poslednej veľkej a dôležitej) kapitole budeme pracovať s textovými súbormi. Je to veľmi praktické a dôležité – predstavte si, že by sme si napísali dlhý text a vytlačili by sme si ho. Program by sme zatvorili. Spomenuli by sme si však, že v texte chceme niečo opraviť, takže po otvorení programu by sme ho museli celý odznova napísať, vytlačiť a po zatvorení programu by sa text zas stratil z pamäte. Asi by vás to nepotešilo. Textový súbor poznáme veľmi dobre (skôr však ako textový dokument – súbor obsahujúci naformátovaný text vytvorený textovým procesorom – napríklad MS Wordom a uložený na pevnom disku). **Textový súbor** je však (na rozdiel od textového dokumentu) **neformátovaný obyčajný text.**

Podľa filozofie údajových typov môžeme povedať, **že textový súbor je** (veľká/rozsiahla/dlhá) **postupnosť znakov uložených v trvalej pamäti počítača.** Je to taká „postupnosť postupnosti" – **textový súbor** je totižto **postupnosť riadkov a až jednotlivé riadky sú postupnosťou znakov – toto je veľmi dôležité si uvedomiť.**

Riadok je každý reťazec, ktorý je ukončený únikovou sekvenciou `\n`.

11.1 Atribúty textového súboru, otváranie, zatváranie

Ak chceme pracovať so súborom, najskôr ho musíme **otvoriť** a definovať, čo s ním chceme robiť – pomocou príkazu `f = open("názov súboru.txt", "r", encoding = "utf-8")`, ak sa súbor nenachádza v tom istom priečinku ako program, uvedieme aj **cestu k súboru.**

atribúty:

- `"r"` = **read** – otvorí súbor len **na čítanie** (nemusí sa uvádzať)
- `"w"` = **write** – ak existuje súbor, príkaz jeho obsah **vymaže** alebo ak súbor ešte neexistuje, **vytvorí sa nový,** preto si **vždy robíme zálohu** pôvodného súboru, aby sme oň náhodou neprišli
- `"a"` = **append** – zapisuje sa na koniec súboru
- na zápis následne používame funkciu `print(s, file = f)`
- `encoding` – uvádza sa, ak je problém s kódovaním (napr. v diakritike v slovenčine)

Po práci so súborom ho **musíme zatvoriť** (zabránime jeho poškodeniu a uvoľníme miesto v operačnej pamäti) príkazom `f.close()`

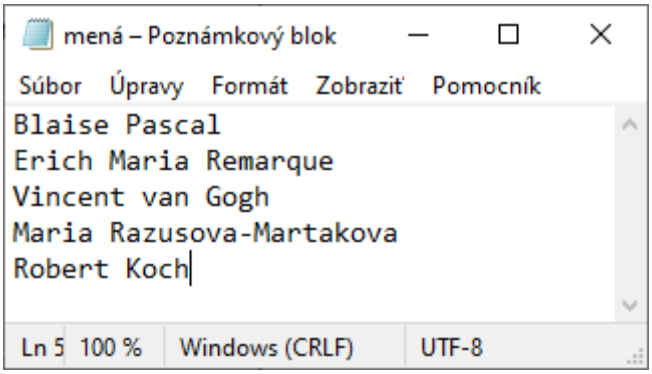
Počas práce (nielen) so súborom využívame príkazy:

- `s = f.read()` – ak je volaná bez parametra, do premennej `s` uloží celý obsah textového súboru, ak ju zavoláme s parametrom, prečíta zadaný počet znakov, **pamätá si, ktorý znak čítala naposledy** a po jej opätovnom zavolaní číta nasledujúci znak alebo skupinu znakov
- `s = f.readline()` – číta **po riadkoch,** tiež si pamätá, kde skončila
- `upper(), lower()` – zmení malé písmená na veľké a naopak
- `a = s.find("")` – vráti **prvý výskyt** hľadanej hodnoty[97]

[97] *Didaktická poznámka – povedali sme síce, že neodporúčame používať túto funkciu a už sme si aj ukazovali programovanie vlastnej, ale už teraz, v zložitejších prípadoch, ju môžete žiakom povoliť používať.*

11.2 Čítanie zo súboru

Poďme si to vyskúšať. K dispozícii máte takýto súbor s menami – **`mená.txt`**.



Skúsme vypísať jednotlivé riadky súboru a oddeľme ich od seba čiarkou a medzerou:

```
1.  f = open("mená.txt", "r")
2.  for x in f:
3.      print(x, end = ", ")
4.  f.close()
```

Na konzolu sa vypíše:

```
Blaise Pascal
, Erich Maria Remarque
, Vincent van Gogh
, Maria Razusova-Martakova
, Robert Koch,
>>>
```

Prečo sa však čiarky vypisujú až na začiatok ďalšieho riadka? **Uvedomme si, že každý riadok je člen postupnosti a v cykle ho reprezentuje `x`.** Riadok je reťazec, ktorý sa končí únikovou sekvenciou **`\n`**, čo znamená nový riadok. Preto koniec riadka je koniec priezviska, nasleduje „enter“ (**`\n`**) a **až potom čiarka.**

Poďme teraz vsunúť čiarku medzi každý jeden znak. Keďže vieme, že **riadky sú postupnosťou znakov, vieme to urobiť veľmi jednoducho – vnoreným cyklom:**

```
1.  f = open("mená.txt", "r")
2.  for x in f:
3.      for y in x:
4.          print(y, end = ", ")
5.  f.close()
```

Na konzolu sa vypíše:

```
B, l, a, i, s, e,  , P, a, s, c, a, l,
, E, r, i, c, h,  , M, a, r, i, a,  , R, e, m, a, r, q, u, e,
, V, i, n, c, e, n, t,  , v, a, n,  , G, o, g, h,
, M, a, r, i, a,  , R, a, z, u, s, o, v, a, -, M, a, r, t, a, k, o, v, a,
, R, o, b, e, r, t,  , K, o, c, h,
>>>
```

Už vieme, ako „dostať“ jednotlivé údaje zo súboru a vedeli by sme s nimi aj ďalej pracovať – ukladať do polí, potom triediť a podobne – to si všetko teraz precvičíme.

11.3 Zápis do súboru

Ako ukladať do súboru? Chceme doň pridať ďalšie meno. Urobíme to takto:

```
1.  f = open("mená.txt", "a")
2.  a = input("Zadaj meno: ")
3.  print(a, file = f)
4.  f.close()
```

1. Otvoríme súbor `mená.txt` na dopisovanie (`"a"` = append, teda pridanie na koniec).
2. Vyžiadame od používateľa meno (textový reťazec).
3. Tento textový reťazec **uložíme funkciou** `print()` (ktorá, ako vieme, slúži na výstup) do textového súboru: `print(a, file = f)`. V premennej `a` sa nachádza **reťazec, ktorý sa zapisuje,** v premennej `f` sa nachádza **názov súboru,** do ktorého sa zapisuje.
4. Textový súbor **nesmieme zabudnúť zatvoriť,** aby sa doň uložili zmeny.

Ak chceme **prepísať súbor novým obsahom,** namiesto atribútu `"a"` píšeme `"w"` (write).

11.4 Zopár úloh využívajúcich prácu s textovým súborom

PRÍKLAD 51

Máme textový súbor `menopriezvisko.txt`, v ktorom sú mená a priezviská ľudí. Napíšte program, ktorý ich prečíta a mená zapíše zvlášť do súboru `meno.txt` a priezviská do `priezvisko.txt`. Z dôvodu zjednodušenia predpokladajme, že každý človek má len jedno meno a jedno priezvisko.

```
1.  def kde_to_je(čo, pole):
2.      poradie = -1
3.      index = 0
4.      while poradie == -1:
5.          if pole[index] == čo:
6.              poradie = index
7.          else:
8.              index += 1
9.      return index
10.
11. f = open("menopriezvisko.txt", "r")
12. mená = []
13. priezviská = []
14.
15. for x in f:
16.     index = kde_to_je(" ", x)
17.
18.     mená += [x[:index] + "\n"]
19.     priezviská += [x[index+1:]]
20.
21. print("Meno:", mená)
22. print("Priezvisko:", priezviská)
23. f.close()
24.
25. g = open("meno.txt", "w")
26. for i in mená:
27.     print(i, file = g, end = "")
28. g.close()
29.
30. h = open("priezvisko.txt", "w")
31. for i in priezviská:
32.     print(i, file = h, end = "")
33. h.close()
```

Ako prvé – použijeme vlastnú funkciu[98] (použite svoju, ktorú ste mali optimalizovať), ktorú sme si už definovali predtým. Všimnite si, že my sme ju vytvorili s cieľom hľadania prvého výskytu v poli, funguje však aj pre reťazec. Prečo? **Je to predsa tiež druh postupnosti – postupnosti znakov, a, ako už vieme, znakový reťazec sa indexuje rovnakým spôsobom ako pole.**

Takže v skratke – otvoríme si súbor, v cykle prechádzame každým riadkom a vždy zisťujeme index výskytu medzery, následne pomocou získaného indexu meno uložíme do poľa s menami a priezvisko do poľa s priezviskami. Zo zaujímavosti si polia vypíšeme na konzolu a, hlavne, nezabudnime **zatvoriť všetky súbory!!!**

[98] *Chceli sme demonštrovať jej reálne využitie, pokojne však, keďže už poznáme jej princíp, použite funkciu* `find()`.

Vypísali sme si ich preto, aby sme si uvedomili, že každá položka poľa (či už meno alebo priezvisko) sa končí únikovou sekvenciou `\n`. Pri ukladaní mien sme ju tam museli pridať manuálne, pri priezviskách ostala – lebo **priezviskom sa končí riadok.**

Teraz si otvorme (vytvorme) nový súbor `meno.txt` na zapisovanie a **v cykle musíme prejsť každú položku, ktorú zapíšeme do súboru.** Keby sme ich v cykle neprešli a zapísali iba „surové" pole, v súbore by bolo aj so zátvorkami, čiarkami, úvodzovkami aj znakom `\n`, čo nechceme. Ale vyskúšajte si to, aby ste sa presvedčili.

Výpis sa predvolene končí enterom, ak nezadáme inak – musíme teda zadať `end = ""`, pretože znak nového riadka (`\n`) tam už je – inak by sme mali každý druhý riadok prázdny – „odenterovaný". To isté urobíme s priezviskami, opäť – oba súbory **je nevyhnutné zatvoriť!**

PRÍKLAD 52

Máme textový súbor `teploty 1.txt`, v ktorom sú namerané teploty, každá hodnota v novom riadku. Napíšte program, ktorý tieto hodnoty prečíta, vypočíta ich priemer a vypíše ho na shell.

```
1.  f = open("teploty.txt", "r")
2.  súčet = počet = 0
3.
4.  for x in f:
5.      x = float(x)
6.      súčet += x
7.      počet += 1
8.  print("Priemer teplôt", súčet/počet)
9.
10. f.close()
```

Je to úplne jednoduché, však? Ale asi si poviete – ako je možné, že sme zobrali zakaždým celý riadok (teda aj s `\n`) a pri pretypovávaní to nevyhodilo chybu? Už sme spomínali, že pretypovacia funkcia `int()` dokáže z reťazca odstrániť medzery aj neplatné cifry. **Dokáže však odstrániť aj únikové sekvencie,** pozor však, „obyčajné" textové znaky odstrániť nevie a spadne na chybe.

PRÍKLAD 53

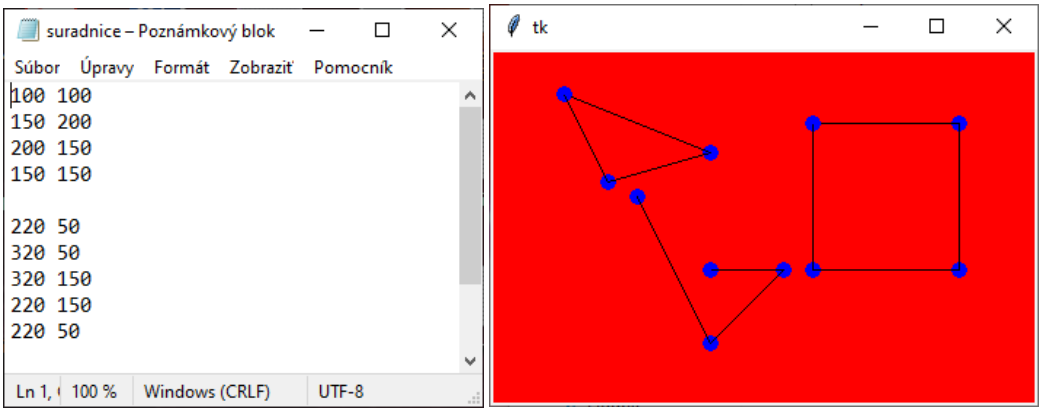
Spravme to isté, akurát teraz budeme mať hodnoty zapísané v jednom riadku – v súbore `teploty 2.txt`. Skúste najskôr sami ako viete, my si ukážeme použitie jednej praktickej funkcie, ktorú môžete používať (už sme ju spomínali). Predpokladajme s cieľom zjednodušenia, že v textovom súbore je dĺžka číselných údajov rovnaká – 2 cifry celočíselnej časti, desatinná bodka a jedna desatinná cifra (teda napríklad teplota 8 °C bude zapísaná ako `08.0`) a teploty sú len kladné (bez znamienka).

```
1.  f = open("teploty 2.txt", "r")
2.  počet = 0
3.  s = f.read(5)
4.  súčet = 0
5.
6.  while s != "":
7.      súčet += float(s)
8.      počet += 1
9.      s = f.read(5)
10. print("Priemer teplôt", súčet/počet)
11.
12. f.close()
```

Je to funkcia `read()`, ktorá **číta vždy nasledujúci znak** (resp. skupinu znakov, **ak do argumentu zadáme číslo**) a pri ďalšom jej zavolaní **si pamätá, kde skončila** (kde „stála" naposledy). Teda `s = f.read(5)` znamená, že do premennej `s` sa uloží vždy päť nasledujúcich znakov – celá časť teploty, desatinná bodka, jedno desatinné miesto a medzera a, ako vieme, s medzerou pretypovávacia funckia nemá problém. Program skončí vtedy, ak `s` bude prázdny reťazec – teda, keď funkcia príde až na koniec súboru. **Ak by sme sa v programe rozhodli súbor čítať opäť, museli by sme ho najskôr zatvoriť a potom zas otvoriť, pretože funkcia `read()` už „stála" na konci súboru.**

Príklad 54

V textovom súbore **`súradnice.txt`** sa nachádza v každom riadku dvojica čísel, ktoré reprezentujú x-ovú a y-ovú súradnicu bodu. Vykreslite tieto body na plátno. Body, ktorých súradnice patria jednej „skupine“ (t. j. riadky nie sú oddelené voľným riadkom), spojte čiarou (prvý z druhým, druhý s tretím atď.). Animujte priebeh kreslenia – body (aj čiary medzi nimi) sa budú vykresľovať postupne v poradí, v akom sú zadané v súbore. Pre priložený súbor je vašou úlohou docieliť niečo takéto:



Nakreslite takýmto spôsobom svoje meno – vytvorte textový súbor, ktorý bude obsahovať súradnice bodov, ktorých vhodným pospájaním vzniknú jednotlivé písmená vášho mena.

Je to pomerne náročná úloha vyžadujúca si poriadne premyslieť, ako uchovávať jednotlivé hodnoty súradníc a ako tvoriť podmienku, kedy sa majú body spojiť a kedy nie. Vyskúšajte si, ako vždy, sami, a potom sa pozrite na možné riešenie a skúste ho optimalizovať (cvičíte si tým schopnosť čítať cudzí kód):

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 500, height = 500, bg = "red")
3.  canvas.pack()
4.
5.  def kresli(a, b, c, d):
6.      if a != "" or b != "":
7.          canvas.create_oval(a-5, b-5, a+5, b+5, fill = "blue", outline = "")
8.
9.      if c != "" or d != "":
10.         canvas.create_line(c, d, a, b, fill = "black")
11.     canvas.update()
12.     canvas.after(1000)
13.
14. f = open("súradnice.txt", "r")
15.
16. počítadlo = -1
17. pole_x = []
18. pole_y = []
19. medzera = 0
20. stĺpec = 0
21.
22. for i in f:
23.     for j in i:
24.         počítadlo += 1
25.         if j == " ":
26.             medzera = počítadlo
27.         elif j == "\n":
28.             stĺpec = počítadlo
29.     if i[:medzera] != "":
30.         pole_x += [int(i[:medzera])]
31.         pole_y += [int(i[medzera+1 : stĺpec])]
32.     else: # ak je tu medzera
33.         pole_x += [""]
34.         pole_y += [""]
35.     počítadlo = -1
36.     medzera = 0
37.
38. počet = 0
39. čiarka_x = čiarka_y = ""
40. for i in pole_x:
```

```
41.     počet += 1
42.     if i == "" or počet == 1:
43.         kresli(pole_x[počet-1], pole_y[počet-1], "", "")
44.     else:
45.         kresli(pole_x[počet-1], pole_y[počet-1], čiarka_x, čiarka_y)
46.     čiarka_x = pole_x[počet-1]
47.     čiarka_y = pole_y[počet-1]
48.
49. f.close()
```

Úloha 28

Napíšte program na kontrolu zátvoriek v texte (všetkých typu **() [] {}**). Vytvorte si textový súbor, kde v každom riadku bude „uzátvorkovaný" text, ktorému môžu chýbať opačné zátvorky. Cieľom programu bude „opraviť" text tak, že doplní v prípade potreby opačnú párovú zátvorku na koniec daného riadka. (Pozor, pamätajte na to, že textový súbor sa dá buď úplne prepísať (**"w"**) alebo dopisovať na jeho koniec (**"a"**). Musíte to spraviť inak – uložením celého obsahu súboru „niekam", to „niečo" upravovať a následne tým prepísať celý súbor).

Príklad – tento riadok:

Marienka (tá od susedov povedala Miškovi svojmu spolužiakovi], že ho má rada.

sa v textovom súbore má prepísať na tento:

Marienka (tá od susedov povedala Miškovi svojmu spolužiakovi], že ho má rada.)[

Program môžete doplniť tak, aby, ak bude chýbať ľavá zátvorka, doplnilo pravú na koniec riadka a ak bude chýbať pravá zátvorka, aby ľavú doplnilo na začiatok riadka. Napríklad takto:

[Marienka (tá od susedov povedala Miškovi svojmu spolužiakovi], že ho má rada.)

Úloha 29

K dispozícii máte textový súbor **sedí mucha na stene.txt**. V ňom je text známej detskej piesne, ktorá sa najskôr spieva normálne:

```
Sedí mucha na stene, na stene, na stene,
sedí mucha na stene, sedí a spí.
Sedí a buvinká, potvora malinká,
sedí mucha na stene, sedí a spí.
```

a potom sa sloha opakuje, ale vždy tak, že sa všetky samohlásky nahradia vždy iba jednou samohláskou – a, e, i, o, u. Napíšte program, ktorý do textového súboru dopíše takto vytvorených 5 sloh. Napríklad, prvý verš ďalšej slohy bude: **Sada macha na stana, na stana, na stana,**.

Úloha 30

Textový súbor **farby.txt** obsahuje v každom riadku jedno meno farby. Napíšte funkciu **do_riadkov(meno_súboru, šírka)**, ktorá vykreslí rad štvorčekov (veľkosti 30×30). Každý z týchto štvorčekov zafarbí príslušnou farbou zo súboru. Po vykreslení **šírka** štvorčekov pokračuje pod týmito v ďalšom rade štvorčekov s ďalšími farbami zo súboru. Takto pokračuje, kým sa neminú všetky farby zo súboru. Napríklad, pre priložený súbor (obrázok vľavo); ak by sme tento súbor vypisovali do 5 stĺpcov, dostali by sme (obrázok vpravo):

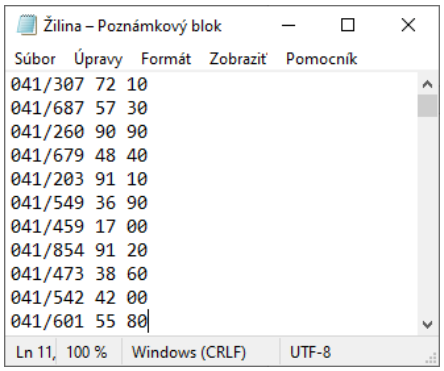
Úloha 31

V rámci celoeurópskeho procesu harmonizácie číslovacieho systému služieb telekomunikačných operátorov pristúpili v roku 2001 Slovenské telekomunikácie k prečíslovaniu všetkých telefónnych čísel. Dovtedy mali telefónne čísla 3- až 4-miestnu predvoľbu (Bratislava dvojmiestnu, ktorá sa zachovala doteraz) a samotné číslo tvorilo 5 alebo 6 cifier (teda číslo mohlo byť 7- až 10-miestne vrátane predvoľby). Po novom má každá predvoľba 3 cifry (okrem Bratislavy) a samotné číslo 7 cifier (Bratislava 8 cifier). Každé telefónne číslo teda musí byť 10-miestne.

K dispozícii máte dva textové súbory. Súbor **`predvoľby.txt`** obsahuje informácie o tom, ktoré predvoľby budú nahradené novou (napr. riadok: **`Bardejov 0935; 0936; 0938; 0937 - 054`** znamená, že všetky vymenované predvoľby budú nahradené predvoľbou **`054`**). Druhý súbor **`telefónne čísla.txt`** obsahuje 15-tisíc starých telefónnych čísel určených na prečíslovanie (predvoľby sú od samotného čísla oddelené spojovníkom). [16]

Napíšte program, ktorý postupne:

- prečíta a vhodným spôsobom si uchová informácie o prečíslovaní z textového súboru **`predvoľby.txt`**
- podľa týchto informácií spracuje všetky staré telefónne čísla nasledujúcim spôsobom:
  - starú predvoľbu nahradí novou
  - medzi číslom a predvoľbou nebude spojovník, ale lomka
  - ak je číslo krátke (spolu s predvoľbou má menej ako 10 miest), doplňte ho sprava nulami
  - aby číslo vyzeralo prehľadnejšie, povkladajte mu medzery
  - napríklad: z čísla **`0826-307721`** bude číslo **`041/307 72 10`**
- čísla každého okruhu (každej predvoľby) uložte zvlášť do textových súborov, ktoré pomenujete podľa príslušného mesta z textového súboru
- napríklad, pre okruh Žilina bude prvých 11 riadkov výsledného súboru **`Žilina.txt`** vyzerať takto:



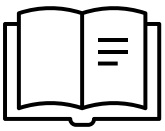
Úloha 32

Detská tajná reč spočíva v tom, že v slove za každou dvojicou písmen samohláska + spoluhláska nasleduje tá istá samohláska a **písmeno p.** Napríklad veta: **Povedz mi, prečo si dnes neprišiel?** znie v tajnej reči takto: **Popovepedz mipi, prepečopo sipi dnepes nepepripišiepel?** Napíšte program na takéto kódovanie a dekódovanie z a do tajnej detskej reči, ktorý načíta súbor s normálnym textom a vytvorí súbor s kódovaným textom a naopak. Uvažujte aj o náhrade dvojhlások (za dvojhláskami ia, ie, iu, ô bude nasledovať pa, pe, pu, po – ako v slove nepriš**ie**l → nepepripiši**epe**l).

Na záver si ešte zhrňme výhody použitia textového súboru:

- údaje sú, na rozdiel od klasického inputu zadávaného klávesnicou, **fyzicky uložené na pevnom disku,** t. j. **sú uložené trvalo, nezaniknú skončením vykonávania programu**
- textový súbor môže upravovať (otvárať a prepisovať) **aj človek, ktorý nie je programátor**
- pri väčšom množstve údajov zaručuje prehľad a **istotu, že sa nestratia**
- **môže s ním pracovať aj iný program** fungujúci na báze iného programovacieho jazyka

Myslíme si, že súborom úloh a príkladov, ktoré táto učebnica obsahuje, ste mali možnosť precvičiť si prácu – ako ste si sami všimli – nielen s textovým súborom, ale aj s rezmi polí, textových reťazcov a tiež algoritmizáciu – ako vhodne vytvoriť podmienky a cykly. Okrem úloh a príkladov, obsiahnutých v tejto učebnici a zvlášť v tejto kapitole, vrelo odporúčame precvičiť si úlohy zo zbierky **Maturujeme v Pythone** *(Kučera, Výbošťok, 2018).* Zbierka obsahuje 64 príkladov, ktoré bohato pokrývajú a rozširujú všetky oblasti programovania, ktoré boli v tejto učebnici spomenuté. **Ak ste sa v učebnici dostali po jej dôkladnom preštudovaní a naprogramovaní všetkých úloh až sem a programovanie podobných úloh a teória vám nerobí väčšie problémy, mali by ste byť kvalitne a dostatočne erudovaní na úrovni maturanta gymnázia z programovacej časti predmetu informatika, na ktorú sa na maturite kladie najväčší dôraz.**



ZHRNUTIE

- ✓ Textový súbor je obyčajný neformátovaný text.
- ✓ Textový dokument je súbor obsahujúci údaje o formátovaní textu, prípadne obrázky a iné prvky, vytvorený textovým procesorom – napr. MS Word.
- ✓ Textový súbor je postupnosť znakov uložená v trvalej pamäti počítača. Je to postupnosť riadkov a jednotlivé riadky sú postupnosťou znakov.
- ✓ Riadok je každý reťazec, ktorý je ukončený únikovou sekvenciou `\n`.
- ✓ Príklad volania funkcie na otvorenie súboru je `f = open("názov súboru.txt", "r", encoding = "utf-8")`.
- ✓ Atribút `"r"` otvorí súbor len na čítanie, `"w"` jeho obsah vymaže a prepíše novým a `"a"` umožňuje zápis reťazca na koniec súboru.
- ✓ Na zápis do súbor sa používa funkcia `print(s, file = f)`, kde `s` je textový reťazec, ktorý chceme zapísať a `f` je súbor, do ktorého chceme zapisovať.
- ✓ Funkciou `read()` čítame súbor po znakoch, táto funkcia si pamätá, ktorý znak čítala naposledy a po jej opätovnom zavolaní číta nasledujúci znak alebo skupinu znakov.
- ✓ Po práci s textovým súborom ho nesmieme zabudnúť zatvoriť funkciou `f.close()`.
- ✓ Výhody textového súboru sú, že údaje sú uložené natrvalo – fyzicky na pevnom disku. Takýto súbor môže upravovať aj človek, ktorý nie je programátor a s rovnakým súborom môže pracovať aj iný program, založený na inom programovacom jazyku.

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem textový súbor.
2. Vysvetlite rozdiel medzi textovým súborom a textovým dokumentom.
3. Čo sa považuje za riadok textového súboru?
4. O čom sme si povedali, že je to postupnosť riadkov?
5. Opíšte, ako by ste otvorili súbor so slovenskou diakritikou na dopisovanie na jeho koniec.
6. Opíšte, ako by ste otvorili súbor na jeho dopisovanie do stredu.
7. Opíšte, ako pracuje funkcia na čítanie znakov v textovom súbore.
8. Kedy vieme, že sme prečítali celý súbor (teda, že sme sa ocitli na konci súboru)?
9. Čo musíme spraviť, ak sa chceme z konca súboru dostať zas na jeho začiatok?
10. Vymenujte výhody používania textového súboru na uchovávanie údajov.

12 Udalosti plátna

CIELE

Žiak sa v tejto doplnkovej kapitole oboznámi s možnosťou ovládať modul tkinter pomocou myši a klávesnice. Získa aspoň základnú predstavu o udalosťami riadenom programovaní a po osvojení si základných princípov ovládania plátna bude schopný naprogramovať jednoduchú hru, ktorej vstupné a výstupné údaje sa zadávajú a zobrazujú na plátne a nie primárne na shelli.

Aby vám vaše doterajšie vedomosti už v ničom nebránili v tvorbe väčšej hry, v tejto kapitole si ešte ukážeme, ako ovládať plátno pomocou myši a klávesnice. Nasledovať bude ešte zopár úloh na precvičenie, ktorými (hlavne však sériou predchádzajúcich úloh) sa môžu učitelia tiež inšpirovať aj pri tvorbe maturitných zadaní.

12.1 Ovládanie myšou – klikanie*

Na grafické plátno sa dá aj „klikať" – budeme pod tým rozumieť **reakciu plátna na udalosť kliknutia.** Poďme si vyskúšať – čo je hlavným cieľom tejto kapitoly a čo budeme využívať – **získavanie súradníc kurzora myši** po jej kliknutí.

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 550, height = 300)
3.  canvas.pack()
4.
5.  def klik(parameter):
6.      canvas.create_text(parameter.x, parameter.y, text = "Ahoj")
7.
8.  canvas.bind("<Button-1>", klik)
```

Čo sme spravili? Vytvorili sme funkciu **`klik()`**, ktorá sa bude automaticky volať pri každom kliknutí do plochy. Nezabudli sme do hlavičky funkcie pridať jeden formálny parameter, inak by Python pri vzniku udalosti protestoval, že našu funkciu **`klik()`** chcel zavolať s jedným parametrom a my sme ho nezadeklarovali.

Teraz k samotnému parametru v našej funkcii **`klik()`**: tento parameter slúži na to, aby tkinter mohol nejakým spôsobom posielať informácie o udalosti. My vieme, že funkcia **`klik()`** sa zavolá vždy, keď sa niekam klikne, ale nevieme, kde presne do plochy sa kliklo. Práve na toto slúži tento parameter – z neho vieme vytiahnuť (okrem iného) napr. x-ovú a y-ovú súradnicu kliknutého miesta. Na naše účely teda parameter pokojne môžeme nazvať **`súradnice`**.

Kliknutie myšou do grafickej plochy vyvolá udalosť s menom **`"<Button-1>"`**. Druhý parameter metódy[99] **`bind()`** musí byť referencia na funkciu, ktorá má jeden parameter.

Zviazanie (**`bind`**) **udalosti s funkciou teda znamená, že každé vyvolanie udalosti** (kliknutie ľavým tlačidlom myši do grafickej plochy) **automaticky zavolá zviazanú funkciu.**

Udalosťou voláme akciu, ktorá vznikne mimo vykonávania programu a program môže na túto situáciu reagovať. Najčastejšie sú to udalosti pohybu a klikania myši, stláčania klávesov, časovača, rôznych zariadení, … V našom prípade potom môžeme nastaviť, čo sa má udiať pri ktorej udalosti. Tomuto sa zvykne hovoriť **udalosťami riadené programovanie** (event-driven programming) a funkciám, ktoré sú zviazané na nejakú udalosť, hovoríme **metódy**.[100]

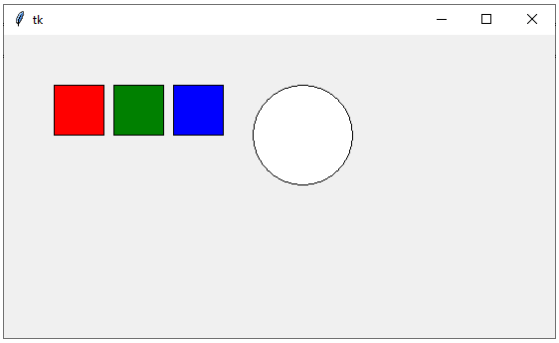
---

[99] *Asi si spomínate, že okrem funkcií a procedúr sme spomínali aj metódy – metódy sú „vlastne akoby" podprogramy v udalosťami riadenom programovaní a objektovo orientovanom programovaní. To, aké sú tam rozdiely a špecifiká, však rozoberať nebudeme.*

[100] *„Načierame" tu teda, ako sme už spomenuli, tak trochu aj do udalosťami riadeného programovania, čo už ale nie je cieľom tejto učebnice a ani kľúčovým a potrebným učivom programovania na stredných školách. Udalosti klikania myšou však chceme využiť a je to praktický a šikovný nástroj, ktorý vám „otvorí" možnosti pri tvorbe jednoduchej hry pomocou procedurálnej paradigmy, ktorej sa venujeme, odkedy sme sa naučili, čo sú podprogramy.*

PRÍKLAD 55

Skúste vytvoriť takýto program: máme tri „tlačidlá“ (štvorce – napríklad červený, zelený a modrý). Po kliknutí nech sa na plátne zobrazí kruh. Ak klikneme do oblasti niektorého zo štvorcov, nech sa kruh zafarbí takou farbou, na ktorý štvorec sme klikli. Ak klikneme mimo oblasti, nech sa zafarbí nabielo.

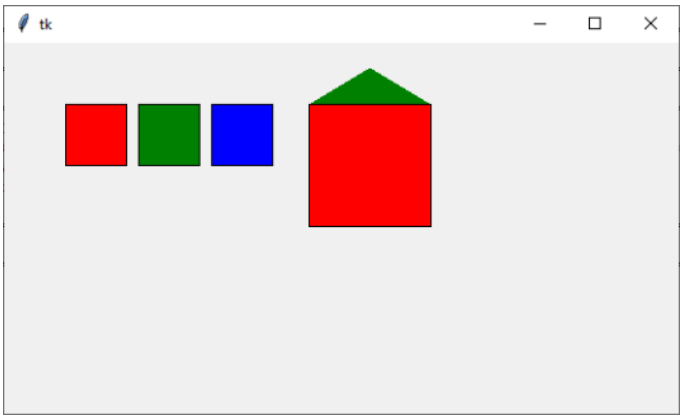


Možné riešenie:

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 550, height = 300)
3.  canvas.pack()
4.
5.  canvas.create_rectangle(50, 50, 100, 100, fill = "red")
6.  canvas.create_rectangle(110, 50, 160, 100, fill = "green")
7.  canvas.create_rectangle(170, 50, 220, 100, fill = "blue")
8.
9.  def klik(súradnice):
10.     farba = "white"
11.     if 100 > súradnice.x > 50 and 100 > súradnice.y > 50:
12.         farba = "red"
13.     elif 160 > súradnice.x > 110 and 100 > súradnice.y > 50:
14.         farba = "green"
15.     elif 220 > súradnice.x > 170 and 100 > súradnice.y > 50:
16.         farba = "blue"
17.
18.     canvas.create_oval(250, 50, 350, 150, fill = farba)
19.
20. canvas.bind("<Button-1>", klik)
```

PRÍKLAD 56

Teraz, keď už vám je asi jasné, ako sa dá pracovať so súradnicami kliknutia, skúste nakresliť domček. Ak kliknete ľavým tlačidlom, zafarbí sa stena, ak pravým tlačidlom, zafarbí sa strecha. Vedzte, že udalosť **"<Button-1>"** slúži na akciu ľavého tlačidla, **"<Button-2>"** na akciu kolieska myši a **"<Button-3>"** na akciu pravého tlačidla myši. Vyzerať to môže napríklad takto:

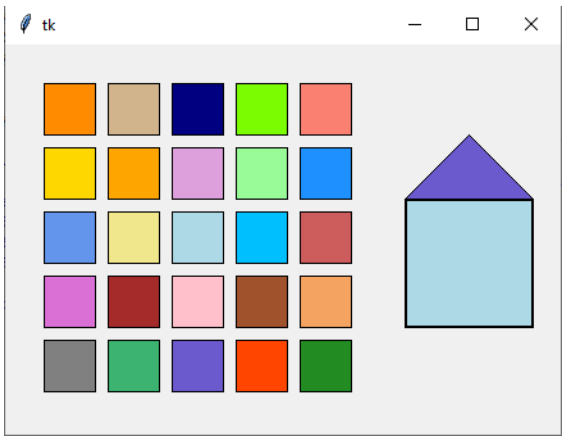


Predpokladáme, že prvé, čo vám napadlo, bolo skopírovať metódu, zmeniť jej názov, **rectangle** zmeniť na **polygon** a zviazať ju s akciou pravého tlačidla myši. Áno, je to riešenie, ale metódy sa odlišujú len v tom, aký tvar kreslia. Načo teda kopírovať dvakrát to isté? Pozrite sa na toto riešenie a rozmýšľajte, kde a ako sa urobili zmeny:

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 550, height = 300)
3.  canvas.pack()
4.
5.  canvas.create_rectangle(50, 50, 100, 100, fill = "red")
6.  canvas.create_rectangle(110, 50, 160, 100, fill = "green")
7.  canvas.create_rectangle(170, 50, 220, 100, fill = "blue")
8.
9.  def výber_farby(x, y):
10.     if 100 > x > 50 and 100 > y > 50:
11.         return "red"
12.     elif 160 > x > 110 and 100 > y > 50:
13.         return "green"
14.     elif 220 > x > 170 and 100 > y > 50:
15.         return "blue"
16.     return "white"
17.
18. def klik_stena(súradnice):
19.     farba = výber_farby(súradnice.x, súradnice.y)
20.     canvas.create_rectangle(250, 50, 350, 150, fill = farba)
21.
22.
23. def klik_strecha(súradnice):
24.     farba = výber_farby(súradnice.x, súradnice.y)
25.     canvas.create_polygon(250, 50, 350, 50, 300, 20, fill = farba)
26.
27. canvas.bind("<Button-1>", klik_stena)
28. canvas.bind("<Button-3>", klik_strecha)
```

Úloha 33

Porozmýšľajte sami, ako by ste riešili nasledujúci program. Funguje tak, ako predchádzajúci, kliknutie ľavým tlačidlom zafarbí stenu, kliknutie pravým zafarbí strechu.



Poradíme vám – pri vytváraní „farebnej palety“ určite využijete **ukladanie farieb do poľa** a pri zisťovaní, na ktorú ste klikli, určite využijete **cyklus v cykle.**

12.2 Ovládanie myšou – ťahanie

Poďme teraz vytvoriť útvar (napríklad kruh), ktorý sa bude pohybovať po plátne **podľa kurzora myši.**

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 800, height = 800)
3.  canvas.pack()
4.
5.  predchádzajúci = None
6.
7.  def ťahaj(súradnice):
8.      global predchádzajúci
9.      x, y = súradnice.x, súradnice.y
10.     canvas.delete(predchádzajúci)
11.     canvas.create_oval(x-10, y-10, x+10, y+10, fill = "red")
12.
13. canvas.bind("<Motion>", ťahaj)
```

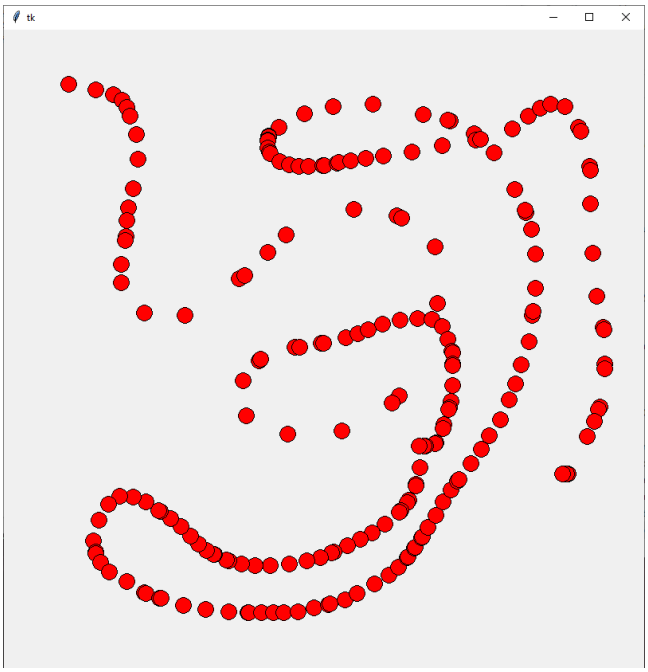
Tu už sa zrejme bez globálnej premennej nezaobídeme. Ako to funguje? Definovali sme si funkciu (v tomto prípade je to už metóda, ide tu totiž už o udalosťami riadené programovanie) **`ťahaj()`**, ktorá **podľa súradníc nakreslí vždy nový útvar a predchádzajúci zmaže.** Ako prvé si musíme inicializovať **prázdnu premennú `predchádzajúci`**, aby na začiatku „mal čo“ vymazať (aby nevypísal chybu, že premennú **`predchádzajúci`** nepozná). Uvedomme si, že **pri každej jednej zmene súradníc, pri každom pohybe myši čo i len o jeden pixel sa metóda vždy zavolá znova,** vykoná sa, premenné zaniknú. My však nechceme, aby zanikla premenná **`predchádzajúci`**, ale aby sa uložila. V tomto prípade, aj keď sa tomu snažíme väčšinou vyhýbať, je vhodné použiť **globálnu premennú.** Metóda vykoná, čo má, **zmení hodnotu globálnej premennej a jej lokálne premenné zaniknú.**

Možno si poviete – veď prečo nevymazať celé plátno vždy pred kreslením nového kruhu, dalo by sa to potom aj bez globálnej premennej. Áno, máte pravdu, pamätajte však, že ak by ste na plátne mali nakreslených viac útvarov, zmizli by všetky. Napriek tomu existuje **ešte jedna možnosť, ktorá je priamo na to určená**[101] – voliteľný atribút pri kreslení útvarov, tzv. **tag** (menovka, štítok).

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 800, height = 800)
3.  canvas.pack()
4.
5.  def ťahaj(súradnice):
6.      x, y = súradnice.x, súradnice.y
7.      canvas.delete("predchádzajúci")
8.      canvas.create_oval(x-10, y-10, x+10, y+10, fill = "red", tag = "predchádzajúci")
9.
10. canvas.bind("<Motion>", ťahaj)
```

Funguje to tak, že útvaru, ktorý kreslíme, priradíme štítok (tag) a potom príkazom **`canvas.delete()`** vymažeme **všetky útvary označené týmto štítkom,** ak také sú. Navyše, predchádzajúci spôsob nie je celkom správny – my sme sa pokúšali uložiť do premennej volanie funkcie bez návratovej hodnoty, čo síce „akýmsi zázrakom“[102] fungovalo, ale je to veľký prehrešok, a, čo si kvázi môžeme dovoliť v Pythone, môže byť (a zvyčajne aj je) v iných jazykoch **neprípustné.**

Keby sme chceli vykresľovanie kruhov podľa pohybu myši, len by sme odstránili vymazávanie predchádzajúceho kruhu.



[101] *Chceli sme vám najskôr načrtnúť problematiku tak, aby ste sa zamýšľali najprv nad tým, čo ste sa doteraz naučili a dokázali to spraviť na základe doterajších znalostí.*

[102] *To, ako a prečo to fungovalo, nebudeme rozoberať, súvisí to s objektmi a objektovo orientovaným programovaním. Žiaci by mali mať zafixované, že v „štandardnom“ procedurálnom programovaní sa do premennej volania funkcií bez návratových hodnôt* ***neukladajú.***

12.3 Ovládanie klávesnicou

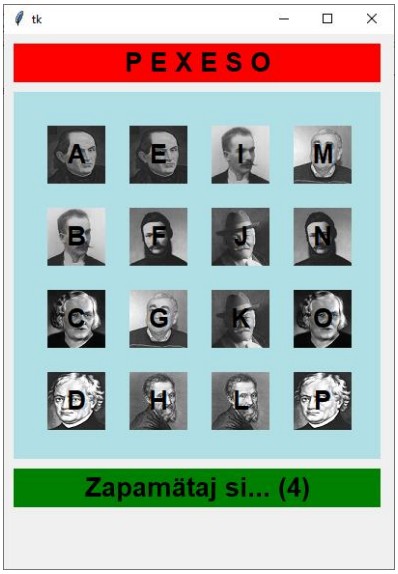
Plátno sa dá, samozrejme, ovládať aj klávesnicou. Nakreslime si štvorec a „hernými“ smerovými klávesmi (**a, s, d, w**) ním budeme po plátne pohybovať.[103]

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 800, height = 800)
3.  canvas.pack()
4.
5.  x, y = 20, 20
6.
7.  def posúvaj(x, y):
8.      canvas.delete("štvorec")
9.      canvas.create_rectangle(x - 20, y - 20, x + 20, y + 20, tag = "štvorec", fill = "red")
10.
11. def akcia_klávesu(kláves):
12.     global x, y
13.     if kláves.char == "a":
14.         x -= 20
15.     elif kláves.char == "s":
16.         y += 20
17.     elif kláves.char == "d":
18.         x += 20
19.     elif kláves.char == "w":
20.         y -= 20
21.     posúvaj(x, y)
22.
23. canvas.bind_all("<Key>", akcia_klávesu)
```

Skúsme však stlačiť napríklad **a** a **s** naraz, čo sa stane? Povedali by ste si – veď je to **cyklus s viacnásobným vetvením `elif`**, buď štvorec pôjde doľava alebo nadol, **podľa toho, ktorý kláves sme stlačili skôr.** Áno, v „klasickom“ programovaní to tak funguje, lenže tu sme sa dostali do **udalosťami riadeného programovania,** kde **vykonávanie programu nie je postupné, ale riadené rôznymi udalosťami.** Udalosťou je zatlačenie klávesu **a,** ale ďalšou, **nezávislou udalosťou,** je zatlačenie klávesu **s.** Tieto dve udalosti **vedia prebiehať aj naraz** – teda sa **naraz mení *x*-ová aj *y*-ová súradnica štvorca.**[104]

ÚLOHA 34

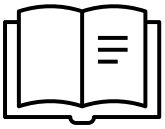
Pokúste sa naprogramovať hru pexeso (stačí rozmerov 4 × 4) na základe vedomostí, ktoré ste nadobudli, napríklad takto:



[103] *Dajú sa „oživiť“ aj iné klávesy – smerové tlačidlá, backspace, medzerník, klávesové kombinácie, ale nebudeme sa tu tomu venovať, lebo je to zložitejšie ako písmenkové klávesy. Ak veľmi chcete, vygooglite si.*

[104] *Odporúčame detailmi tejto kapitoly nezaťažovať už ani maturitné ročníky, je to tu len zo zaujímavosti a na oživenie. Vhodné je len, aby vedeli naprogramovať klikanie na plátno, nie je potrebné, aby detailne vedeli, na akom princípe to funguje – na základe tejto elementárnej vedomosti odporúčame formulovať aj niektoré maturitné zadania. Ťažiskom maturitných úloh však majú byť algoritmizačné úlohy – práca s poľom, znakovými reťazcami, textovým súborom, podmienky, cykly.*

Po spustení hry nechajte zobraziť všetky kartičky na určitý čas (10 sekúnd), potom ich nechajte skryť. Môže a nemusí byť ovládané klikaním myši. Pokojne len označte políčka písmenami a vyžadujte vstup od hráča cez shell. Po uhádnutí vypíšte na vhodné miesto do plátna gratuláciu, informujte hráča, koľký ťah už vykonal a koľko párov mu ostáva. Môžete vytvárať kartičky pomocou rôznych útvarov (**`rectangle`**, **`oval`**, **`polygon`**), alebo si vygooglite, ako sa vkladajú obrázky. Kreativite sa medze nekladú.



ZHRNUTIE

- ✓ Udalosťou voláme akciu, ktorá vznikne mimo vykonávania programu a program môže na túto situáciu reagovať.
- ✓ Zviazanie udalosti s metódou znamená, že každé vyvolanie udalosti automaticky zavolá zviazanú metódu.
- ✓ Funkciám, ktoré sú zviazané s udalosťou, hovoríme metódy.
- ✓ Udalosti vedia prebiehať aj naraz, čoho jasným dôkazom je pohybovanie útvaru do uhlopriečky po stlačení dvoch smerových kláves naraz, pričom akcie jednotlivých tlačidiel sú definované v cykle s viacnásobným vetvením (teda v cykle, v ktorom sa vykoná vždy len jedna vetva).

OTÁZKY NA ZOPAKOVANIE

1. Definujte pojem udalosť.
2. Aký je základný rozdiel medzi procedurálnym a udalosťami riadeným programovaním?
3. Opíšte, ako získate hodnoty súradníc kurzora myši po jej kliknutí.
4. Čo znamená zviazanie metódy s udalosťou?
5. Aký je rozdiel medzi funkciami a metódami?
6. Ako často sa volá metóda zviazaná s udalosťou ťahania myši?
7. Na jednoduchom príklade dokážte, že udalosti dokážu prebiehať aj naraz.

13 Tvorba jednoduchej hry

Ciele

Cieľom tejto poslednej kapitoly je, aby žiak vedel zúročiť a ešte viac si prehĺbiť všetky doteraz osvojené vedomosti, zručnosti a schopnosti – kreslenie, cykly, podmienky, práca s poľom, definície podprogramov a udalosti plátna – implementovaním jednoduchej, ale zato komplexne a na úrovni vyzerajúcej spoločenskej hry Logik.

Záverečná kapitola tejto učebnice bude vaším sprievodcom a pomocníkom pri tvorbe hry Logik. Pravidlá, ktoré budeme implementovať do programu, sú nasledujúce [18]:

- ✓ Na začiatku hry jeden z hráčov (v našom prípade počítač) tajne umiestni pod striešku ľubovoľnú variáciu[105] štyroch farebných špendlíkov (pričom rovnaká farba sa môže opakovať viackrát).[106]
- ✓ Druhý hráč sa túto variáciu snaží uhádnuť tým, že vytvára svoju variáciu štyroch špendlíkov.
- ✓ Za každý špendlík, ktorý hráč umiestnil so správnou farbou na správne miesto, pridá spoluhráč **čierny špendlík.**
- ✓ Za každý špendlík, ktorý hráč umiestnil so správnou farbou, ale na zlé miesto, pridá **biely špendlík.**
- ✓ Hádajúci hráč **nevie, za ktorý špendlík** boli pridelené hodnotiace špendlíky.
- ✓ Hráč má zvyčajne k dispozícii na výber zo 6 farieb (ak háda 4 pozície).
- ✓ Hra sa končí, ak:
  - hráč uhádne variáciu (víťazný pokus je ohodnotený **plným počtom čiernych špendlíkov**)
  - alebo vtedy, ak hráč **obsadí všetky riadky na hracej doske** bez toho, aby variáciu uhádol – vtedy prehrá.

13.1 Logická časť 1

Najdôležitejšie na celej podstate hry je naprogramovanie jej **logickej časti.** Tou sa môžete inšpirovať kódom nižšie, ktorý si ideme teraz rozobrať.

Ako prvé, importujeme modul **`random`**, keďže chceme, aby farby náhodne vyberal počítač namiesto spoluhráča. Vytvoríme si tiež pole (teraz s písmenami A, B, C, D, E, F, neskôr ho upravíme na reálne názvy farieb).

`pole_generované` bude uchovávať variáciu vygenerovanú počítačom, **`pole_hádané`** uloží vstup zo shellu zadaný hráčom.

Celý logický (vyhodnocovací) proces hry bude prebiehať v cykle **`while`** – hrať sa bude dovtedy, kým hráč neuhádne všetky pozície (my teraz hádame konkrétne 4 pozície zo 6 možných farieb – klasická verzia hry). Maximálny možný počet hádaní (spravidla 10) sme zatiaľ neobmedzovali (teda, že ak ani na 10. pokus hráč neuhádne variáciu, hra sa ukončí).

Teda, v cykle so štyrmi opakovaniami sa vyberú náhodne štyri písmenká, ktoré sa uložia do poľa **`pole_genrované`**. Necháme si ho (samozrejme, len na účely testovania, inak by hra nemala zmysel) vypísať na konzolu.

Následne prebieha cyklus **`while`**, dokedy sa čierna nerovná štyrom (teda, dokedy neuhádneme variáciu).

Prečo sme si tu definovali novú premennú – nové pole **`pole_upravené`**? Doň sme si nakopírovali položky poľa **`pole_generované`**, a to preto, pretože v nasledujúcich cykloch si budeme polia „pomocne

[105] *Je to variácia, lebo záleží na poradí farieb (nepoužívajte v tomto zmysle slovo kombinácia; v matematike to totiž znamená výber, kde nezáleží na poradí).*

[106] *Čo ani netreba spomínať, z pojmu variácia je to zrejmé; sme však na informatike, nie na matematike a aj pravidlá hry sú písané pre „nematematikov", teda pokojne sa môžeme v pravidlách rôznych hier stretnúť aj so spojením „vyberáte kombinácie, kde záleží na poradí", čo je však matematicky nezmysel.*

prepisovať“. Ešte predtým sme si ale vstup po znakoch uložili do poľa **pole_hádané** (my sme napríklad zadali **"ABCB"**, a teda v poli **pole_hádané** bude **["A", "B", "C", "B"]**).

Teraz budeme (najlepšie v dvoch „nezávislých“ – po sebe idúcich cykloch) porovnávať hodnoty a zisťovať zhody medzi nami zadanou a vygenerovanou variáciou. **Prednosť má zisťovanie, či na rovnakej pozícii je aj rovnaká farba.** Ak je, počítadlo **čierna** si navýšime o jeden a porovnávaný pár premeníme na inú hodnotu – napr. **"X"**.

Následne, v ďalšom cykle zisťujeme zhody farieb, ktoré **nie sú na správnom mieste.** Ak sa na danej **i**-tej pozícii v **poli_hádané** a na **j**-tej pozícii v **pole_upravené** **nenachádza** **"X"**, tak potom, ak sa tieto hodnoty rovnajú, je jasné, že nie sú na správnom mieste – pripíšeme si do počítadla **biela** jednotku a porovnávané hodnoty „zrušíme“ prepísaním na **"X"**.

Nakoniec pre hráča na konzolu vypíšeme, koľko čiernych a koľko bielych špendlíkov získal. Ak to sú tie spomínané 4 čierne špendlíky, cyklus **while** sa ukončí a vypíše sa **"Vyhral si."**. Ak nie, cyklus vynuluje počítadlá **čierna** a **biela**, vymaže **pole_hádané** aj **pole_upravené**, uloží doň nanovo prvky poľa **pole_generované**, opäť čaká na vstup hráča a pokračuje vo vyhodnocovaní zhody.

```
1.  import random
2.  výber = ["A", "B", "C", "D", "E", "F"]
3.
4.  pole_generované = []
5.  pole_hádané = []
6.  čierna = 0
7.
8.  for i in range(4):
9.      pomocný = random.choice(výber)
10.     pole_generované += [pomocný]
11.
12. print(pole_generované)
13.
14. while čierna != 4:
15.     čierna = 0
16.     biela = 0
17.     pole_upravené = []
18.     pole_hádané = []
19.
20.     for i in range(len(pole_generované)):
21.         pole_upravené += [pole_generované[i]]
22.
23.     vstup = input("Zadajte variáciu (napr. ADDE): ")
24.
25.     for i in vstup:
26.         pole_hádané += [i]
27.
28.     for i in range(len(pole_generované)):
29.         if pole_hádané[i] == pole_upravené[i]:
30.             čierna += 1
31.             pole_upravené[i] = "X"
32.             pole_hádané[i] = "X"
33.
34.     for i in range(len(pole_generované)):
35.         for j in range(len(pole_generované)):
36.             if pole_hádané[i] != "X":
37.                 if pole_upravené[j] != "X":
38.                     if pole_hádané[i] == pole_upravené[j]:
39.                         biela += 1
40.                         pole_hádané[i] = "X"
41.                         pole_upravené[j] = "X"
42.
43.     print("Pole hádaných:", pole_hádané)
44.     print("Pole upravených:", pole_upravené)
45.
46.     print("Čiernych špendlíkov:", čierna)
47.     print("Bielych špendlíkov:", biela)
48.
49. print("Vyhral si")
```

13.2 Grafická časť

Hra vlastne už funguje, lenže každá dobrá hra má používateľsky prívetivé grafické prostredie, ktoré sa teraz v rámci našich doteraz získaných skúseností pokúsime naprogramovať. V tejto časti tvorby hry si vytvoríme **samostatný program,** ktorý zaručí ukladanie farebnej variácie do poľa nie cez shell a `input()`, ale klikaním na plátne.

Bude to spočívať v naprogramovaní niekoľkých **podprogramov** (funkcií a metód), ktoré sa o toto všetko postarajú. My si teraz podrobne rozoberieme každý z nich.

V prvom rade tu vidíme nejaké importy – `winsound` slúži na zvukové oživenie hry – po kliknutí myšou necháme program pípnuť. Z grafického modulu `tkinter` zvlášť importujeme balíček `messagebox` – to sú klasické vyskakovacie okná operačného systému – informačné, s otázkou, s chybovým hlásením a iné.

Zvolenú farbu si dajme predvolene napríklad bielu (alebo takú, ktorej farby nie je žiaden špendlík). K vloženiu obrázka aj premennej `iba_raz` sa ešte dostaneme.

`voľba` bude pole (teraz napríklad dvanástich) farieb, ktoré si vytvoríme v cykle tak, že doň pridáme dvanásť núl. Vytvoríme si zároveň dvanásť pomocných „chlievikov“, do ktorých sa bude klikať.

Pripomeňme, že tu už ide o udalosťami riadené programovanie a v tomto prípade je jedno, ktorá metóda je kedy definovaná – môže sa pokojne „skôr“ definovaná metóda odvolávať na nejakú „neskôr“ definovanú. Neplatí tu striktne to, na čo sme upozorňovali, keď sme definovali **funkciu** na prevody číselných sústav (tá tretia – `zNdoN()` sa odvolávala na funkcie `z10do()` a `do10z()`, ktoré v tomto prípade museli byť nevyhnutne definované pred ňou).

Metóda `klik(súradnice)` je zviazaná s udalosťou kliknutia myši (riadok 135) a robí toto:

- ✓ Metóda potrebuje nielen **pristupovať k premenným** `zvolená_farba` a `voľba`, ale ich aj **meniť.** V tomto prípade si dovolíme zmeniť ich na **globálne premenné**.[107]
- ✓ Zvolená farba **prijme návratovú hodnotu zavolanej metódy** `kurzor(súradnice)`, ktorá na základe výberu farby **vráti jej názov.** Jej fungovanie (teda, ako sa vyberá farba) si vysvetlíme neskôr.
- ✓ Najdôležitejšie príde na rad teraz: metóda v cykle s 12 opakovaniami po každom kliknutí **zisťuje, či sme klikli do jedného z 12 chlievikov.** Ak áno a ak nie je zvolená biela farba (my sme si stanovili, že ak je zvolená biela, nie je zvolená „akoby žiadna“) a klikli sme do jedného z 12 chlievikov, vykreslí sa do daného chlievika krúžok reprezentujúci špendlík s danou zvolenou farbou. Do poľa `voľba` sa na dané miesto zapíše konkrétna farba (ak sme napríklad klikli do oblasti 6. chlievika, cyklus prebehne 6-krát, na 6. chlieviku sa zastaví, nakreslí mu krúžok s danou farbou a danú farbu uloží do poľa na 6. pozíciu a cyklus sa preruší príkazom `break` – pretože už nie je potrebné ďalej zbytočne zisťovať, kam sa kliklo, keď sme to už práve zistili. Ak je farba biela, teda nezvolená, nič sa nezapíše). Pre našu informáciu si pokojne pole s farbami môžeme nechať vypisovať na shell a tam účel tejto metódy pekne uvidíme.

Funkcia `zisťuj_pole()` zisťuje, či má pole `voľba` zapísané všetky farby – teda, **či neobsahuje nejakú nulu.** Ak je pole zaplnené farbami, vyvolá informačné okno, že farby boli úspešne zadané a túto hodnotu (pole `voľba`) vráti. Ak nie, vyvolá chybové hlásenie, ktoré nás vyzve, aby sme pole doplnili. Túto užitočnú funkciu budeme neskôr potrebovať na vhodnom mieste zavolať.

```
1.  import winsound
2.  import tkinter
3.  from tkinter import messagebox
4.  canvas = tkinter.Canvas(width = 800, height = 900)
```

[107] *Odteraz je vhodné povedať si, resp. „mať za svoje“, že* ***regulárne premenné sú v každom podprograme prístupné – ale iba na čítanie.*** *V kapitole, kde sa s podprogramami stretáva žiak prvýkrát, to spomínať neodporúčame (nechajte ho „v tom“, že to tak nie je – aby sa poriadne naučil definovať funkcii parametre a ukladať jej návratové hodnoty). Ak ich chceme aj modifikovať,* ***v danom podprograme*** *pred ne musíme napísať dohovorené slovo* `global`.

```
5.  canvas.pack()
6.
7.  zvolená_farba = "white"
8.  obrázok = tkinter.PhotoImage(file = "obrazky/otaznik.png")
9.  iba_raz = True
10.
11. voľba = []
12.
13. x, y = 30, 30
14.
15. for i in range (12):
16.     voľba += [0]
17.     canvas.create_rectangle(x, y, x+40, y+40)
18.     x += 40
19.
20.
21. def klik(súradnice):
22.     global zvolená_farba
23.     global voľba
24.     x, y = súradnice.x, súradnice.y
25.     zvolená_farba = kurzor(súradnice)
26.     a, b = 30, 30
27.     for i in range(12):
28.         if ((zvolená_farba != "white") and (x > a and y > b and x < a+40 and y < b+40)):
29.             canvas.create_oval(a, b, a+40, b+40, fill = zvolená_farba)
30.             voľba[i] = zvolená_farba
31.             break
32.         a += 40
33.     print("Pole farieb", voľba)
34.
35.
36. def zisťuj_pole():
37.     test = 0
38.     for i in range(len(voľba)):
39.         if voľba[i] == 0:
40.             test += 1
41.     if test == 0:
42.         messagebox.showinfo(title = "Úspešné.", message = "Farby boli úspešne zadané.")
43.         return voľba
44.     else:
45.         messagebox.showerror(title = "Chyba.",
46.                              message = "Nezvolil si všetky pozície. Doplň svoju voľbu!")
47.
```

Metóda **`ťahaj(súradnice)`** je zviazaná s udalosťou ťahania myši (riadok 136) a robí toto:

- ✓ Metóda je volaná **každou jednou zmenou súradníc** (každým pohybom myši – už sme si to bližšie opisovali v predchádzajúcej kapitole) a v tomto prípade vždy na mieste, kde sa nachádza kurzor, vykreslí obrázok (aký obrázok – tým sa zaoberá ďalšia funkcia) a starý (ten na predchádzajúcej pozícii) vymaže.

Metóda **`kurzor(súradnice)`** má meniť obrázok, ktorý neustále vykresľuje metóda **`ťahaj`** takýmto spôsobom:

- ✓ Ak sme klikli do „misky so špendlíkmi“ – sú to oblasti definované na konci tohto programu – teda **klik spadá do jednej z oblastí** (prienik súradníc), **do lokálnej premennej sa uloží zodpovedajúca farba,** ktorá sa po vykonaní činnosti podprogramu aj vráti.
- ✓ Ak sme do misky so špendlíkmi klikli (čiže nie je farba **`"white"`**), obrázok „kurzora“ (otáznik) sa zmení **na špendlík konkrétnej farby** (ako môžeme vidieť – pomocou formátovacieho reťazca – použije sa obrázok so špendlíkom konkrétnej farby, ktorý je ňou pomenovaný), kým farbu niekde „neodklikneme“. Simulujeme tým reálnu situáciu – ak si zoberieme špendlík z misky, buď ho umiestnime do (jedného z dvanástich) chlievika, alebo ho „odhodíme“ preč, teda po jeho odknutí „niekam“ sa kurzor opäť zmení na otáznik a je potrebné „zobrať“ si ďalší špendlík (kliknúť na jednu z misiek).

```
48. def ťahaj(súradnice):
49.     x, y = súradnice.x, súradnice.y
50.     canvas.delete("predchádzajúci")
51.     canvas.create_image(x, y, image = obrázok, tag = "predchádzajúci")
52.     canvas.update()
53.
```

```
54.
55. def kurzor(súradnice):
56.     global obrázok
57.     global iba_raz
58.     x, y = súradnice.x, súradnice.y
59.     farba = "white"
60.
61.     if (x>10 and y>610 and x<130 and y<730):
62.         farba = "orangered"
63.         iba_raz = True
64.     elif (x>130 and y>610 and x<250 and y<730):
65.         farba = "gold"
66.         iba_raz = True
67.     elif (x>250 and y>610 and x<370 and y<730):
68.         farba = "yellowgreen"
69.         iba_raz = True
70.     elif (x>370 and y>610 and x<490 and y<730):
71.         farba = "dodgerblue"
72.         iba_raz = True
73.     elif (x>10 and y>730 and x<130 and y<850):
74.         farba = "saddlebrown"
75.         iba_raz = True
76.     elif (x>130 and y>730 and x<250 and y<850):
77.         farba = "purple"
78.         iba_raz = True
79.     elif (x>250 and y>730 and x<370 and y<850):
80.         farba = "darkgrey"
81.         iba_raz = True
82.     elif (x>370 and y>730 and x<490 and y<850):
83.         farba = "black"
84.         iba_raz = True
85.     else:
86.         if iba_raz:
87.             farba = zvolená_farba
88.             iba_raz = False
89.
90.     if farba != "white":
91.         obrázok = tkinter.PhotoImage(file = f"obrazky/{farba}_spendlik.png")
92.         winsound.Beep(500, 200)
93.     else:
94.         obrázok = tkinter.PhotoImage(file = "obrazky/otaznik.png")
95.
96.     if not(iba_raz):
97.         obrázok = tkinter.PhotoImage(file = "obrazky/otaznik.png")
98.
99.     return farba
100.
```

Zvyšná časť programu vykresľuje „misky“ (oblasti so špendlíkmi, na ktoré ak klikneme, vyberieme si farbu). Na záver je vytvorené tlačidlo, ktoré je prepojené na metódu **`zisťuj_pole`** a zisťuje „platnosť“ poľa s farbami – ak sú nuly, pole je potrebné doplniť. **`zisťuj_pole`** je metóda, lebo je volaná po udalosti kliknutia na tlačidlo.

```
101.
102.
103. canvas.create_rectangle(10, 610, 130, 730, fill = "orangered", outline = "")
104. špendlíky_1 = tkinter.PhotoImage(file = "obrazky/orangered_spendliky.png")
105. canvas.create_image(70, 670, image = špendlíky_1)
106.
107. canvas.create_rectangle(130, 610, 250, 730, fill = "gold", outline = "")
108. špendlíky_2 = tkinter.PhotoImage(file = "obrazky/gold_spendliky.png")
109. canvas.create_image(190, 670, image = špendlíky_2)
110.
111. canvas.create_rectangle(250, 610, 370, 730, fill = "yellowgreen", outline = "")
112. špendlíky_3 = tkinter.PhotoImage(file = "obrazky/yellowgreen_spendliky.png")
113. canvas.create_image(310, 670, image = špendlíky_3)
114.
115. canvas.create_rectangle(370, 610, 490, 730, fill = "dodgerblue", outline = "")
116. špendlíky_4 = tkinter.PhotoImage(file = "obrazky/dodgerblue_spendliky.png")
117. canvas.create_image(430, 670, image = špendlíky_4)
118.
119. canvas.create_rectangle(10, 730, 130, 850, fill = "saddlebrown", outline = "")
120. špendlíky_5 = tkinter.PhotoImage(file = "obrazky/saddlebrown_spendliky.png")
121. canvas.create_image(70, 790, image = špendlíky_5)
122.
123. canvas.create_rectangle(130, 730, 250, 850, fill = "purple", outline = "")
124. špendlíky_6 = tkinter.PhotoImage(file = "obrazky/purple_spendliky.png")
```

```
125.  canvas.create_image(190, 790, image = špendlíky_6)
126.
127.  canvas.create_rectangle(250, 730, 370, 850, fill = "darkgrey", outline = "")
128.  špendlíky_7 = tkinter.PhotoImage(file = "obrazky/darkgrey_spendliky.png")
129.  canvas.create_image(310, 790, image = špendlíky_7)
130.
131.  canvas.create_rectangle(370, 730, 490, 850, fill = "black", outline = "")
132.  špendlíky_8 = tkinter.PhotoImage(file = "obrazky/black_spendliky.png")
133.  canvas.create_image(430, 790, image = špendlíky_8)
134.
135.  canvas.bind('<Button-1>', klik)
136.  canvas.bind('<Motion>', ťahaj)
137.
138.  tlačidlo = tkinter.Button(text = "Skontroluj", command = zisťuj_pole)
139.  tlačidlo.pack(side = tkinter.BOTTOM)
140.
```

Toto všetko sme programovali „len preto“, aby sme nemuseli farby zadávať ručne cez `input()`.

13.3 Logická časť 2

Vráťme sa teraz k logickej časti hry a poďme ju vylepšiť. Pridajme sem podmienku prehry – teda, aby sa hra neukončila len vtedy, ak uhádneme celú variáciu, ale aj vtedy, ak ju **neuhádneme ani na desiatykrát.**

Tiež sem pridajme **hru na levely,** my ich dáme na päť:

- prvý level bude vyberanie **dvoch** farieb zo **štyroch** (ABCD),
- druhý level bude vyberanie **troch** farieb z **piatich** (ABCDE),
- tretí level bude vyberanie **štyroch** farieb zo **šiestich** (ABCDEF),
- štvrtý level bude vyberanie **piatich** farieb zo **siedmich** (ABCDEFG),
- piaty level bude vyberanie **šiestich** farieb z **ôsmich** (ABCDEFGH),

pričom začneme prvým levelom, po jeho úspešnom ukončení (uhádnutí) prejdeme na ďalší level a ak uhádneme aj v poslednom, vyhrali sme, pričom v každom leveli nechajme maximálny možný počet pokusov 10, ak v ktorejkoľvek fáze (leveli) hry neuhádneme ani na 10. pokus, prehrali sme a hra sa ukončí.

Bodovanie nastavme takto:

- ✓ V každom leveli je možné získať maximálne 10 bodov (ak trafíme variáciu na prvýkrát),
- ✓ pričom každým ďalším pokusom získame o jeden bod menej (napr. na 6. pokus získame len 5 bodov),
- ✓ teda z celej hry môžeme získať:
  - najviac 50 bodov (ak by sme „podvádzali“, nechali vypisovať variácie na shell a trafili vždy hneď)
  - a najmenej 5 (ak by sme hypoteticky každý level prešli „s odretými ušami“ na posledný pokus).

Tento kód nepovažujeme za potrebné bližšie komentovať – poďme sa však na to pozrieť:

```
1.  import random
2.  výber = ["A", "B", "C", "D", "E", "F", "G", "H"]
3.
4.  pole_generované = []
5.  pole_hádané = []
6.  čierna = 0
7.
8.  print("\nZahraj si hru logik.\nHra má 5 úrovní (levelov) obťažnosti. V každom leveli hádaš n "
9.        "farieb z n+2. Farby ti generuje počítač a môžu sa opakovať. Tvojou úlohou je za čo "
10.       "najmenej pokusov uhádnuť farby aj ich pozície. Ak uhádneš pozíciu, získaš čierny bod, "
11.       "ak uhádneš len farbu, získaš biely bod. Ak uhádneš všetky pozície maximálne na 10. "
12.       "pokus, posúvaš sa na ďalší level. Ak prejdeš všetkých 5 levelov, vyhral si. Ak ani na "
13.       "10. pokus netrafíš, prehral si a hra sa ukončí. \nVeľa šťastia!\n")
14.
15. level = 1
16. výhra = 0
17. prehra = 0
18.
19. while level <= 5 and prehra == 0:
20.     pole_generované = []
21.     pokus = 0
22.
23.     body = výhra + 10
24.     print("****************************")
```

```
25.     print("Level:", level)
26.
27.     for i in range(level+1):
28.         pomocný = random.choice(výber[:level+3])
29.         pole_generované += [pomocný]
30.
31.     print(pole_generované)
32.
33.     if level+1 < 5:
34.         print(f"Vyberáš {level+1} farby z týchto: {výber}")
35.     else:
36.         print(f"Vyberáš {level+1} farieb z týchto: {výber}")
37.
38.     while čierna != level+1:
39.         čierna = 0
40.         biela = 0
41.         pole_upravené = []
42.         pole_hádané = []
43.
44.         for i in range(len(pole_generované)): pole_upravené += [pole_generované[i]]
45.
46.         print("Pokus č.", pokus+1)
47.
48.         vstup = input("Zadajte variáciu (napr. ADDE): ")
49.
50.         for i in vstup: pole_hádané += [i]
51.
52.         if len(pole_hádané) != level+1:
53.             print("Zadali ste neplatný vstup.")
54.         else:
55.             pokus += 1
56.
57.             for i in range(len(pole_generované)):
58.                 if pole_hádané[i] == pole_upravené[i]:
59.                     čierna += 1
60.                     pole_upravené[i] = "X"
61.                     pole_hádané[i] = "X"
62.
63.             for i in range(len(pole_generované)):
64.                 for j in range(len(pole_generované)):
65.                     if pole_hádané[i] != "X":
66.                         if pole_upravené[j] != "X":
67.                             if pole_hádané[i] == pole_upravené[j]:
68.                                 biela += 1
69.                                 pole_hádané[i] = "X"
70.                                 pole_upravené[j] = "X"
71.
72.             print("Pole hádaných:", pole_hádané)
73.             print("Pole upravených:", pole_upravené)
74.
75.             print("Čiernych špendlíkov:", čierna)
76.             print("Bielych špendlíkov:", biela)
77.
78.             if čierna == level+1:
79.                 body -= pokus -1
80.                 print(f"Vyhral si na {pokus}. pokus.")
81.                 print(f"Máš {body} bodov.")
82.                 výhra = body
83.             if pokus == 10 and čierna != level+1:
84.                 print("Prehral si.")
85.                 prehra = 1
86.                 break
87.     level += 1
88.
89. print("Koniec hry.")
```

13.4 Spojenie grafickej a logickej časti

V poslednej fáze tvorby hry poďme **vzájomne spojiť grafickú a logickú časť hry.** Z programu na grafické vytváranie poľa s variáciou farieb skopírujme všetko (Ctrl + A) a vložme na vhodné miesto do programu na samotné hranie hry, samozrejme, pred cyklus **`while`** a za informačným výpisom pravidiel hry. Upozorňujeme, **aby ste si spravili zálohu oboch programov.**

Ako prvé, dáme dohromady na začiatok programu **všetky importy.**

Keďže už sme pracovali s vyskakovacími oknami, pridáme sem ešte jedno – tzv. **simpledialog**, ktoré musíme tiež importovať. Doň pôjdu inštrukcie z hry, ktoré sme predtým vypisovali na shell. Všetky definované premenné, ktoré je potrebné mať vytvorené pred samotným vykonávaním programu, tiež umiestnime na vhodné miesto na začiatok. Pribudli nové, ich význam si budeme buď postupne rozoberať, alebo prídete sami na to, na čo slúžia.

```
1.  import winsound
2.  import random
3.  import tkinter
4.  from tkinter import messagebox
5.  from tkinter import simpledialog
6.
7.
8.  canvas = tkinter.Canvas(width = 500, height = 860, bg = "paleturquoise")
9.  canvas.pack()
10.
11. var = tkinter.IntVar()  # z internetu: zdroj č. [17]
12. tlačidlo = tkinter.Button(text = "Skontroluj", command = lambda: var.set(1))
13. tlačidlo.pack(side = tkinter.BOTTOM)
14.
15. messagebox.showinfo(title = "Vitajte v hre Logik",
16.                       message = "\nZahraj si hru logik.\nHra má 5 úrovní (levelov) obťažnosti. "
17.                                 "V každom leveli hádaš n farieb z n+2. Farby ti generuje "
18.                                 "počítač a môžu sa opakovať. Tvojou úlohou je za čo najmenej "
19.                                 "pokusov uhádnuť farby aj ich pozície. Ak uhádneš pozíciu, "
20.                                 "získaš čierny bod, ak uhádneš len farbu, získaš biely bod. Ak "
21.                                 "uhádneš všetky pozície maximálne na 10. pokus, posúvaš sa na "
22.                                 "ďalší level. Ak prejdeš všetkých 5 levelov, vyhral si. Ak ani "
23.                                 "na 10. pokus netrafíš, prehral si a hra sa ukončí. \nVeľa "
24.                                 "šťastia!\n")
25.
26. # výber = ["A", "B", "C", "D", "E", "F", "G", "H"]
27. výber = ["orangered", "gold", "yellowgreen", "dodgerblue",
28.          "saddlebrown", "purple", "darkgrey", "black"]
29.
30. pole_generované = []
31. pole_hádané = []
32. čierna = 0
33. level = 1
34. výhra = 0
35. prehra = 0
36. info = "Začni!"
37. zvolená_farba = "white"
38. obrázok = tkinter.PhotoImage(file = "obrazky/otaznik.png")
39. iba_raz = True
40. chlievik_y = 120
41. voľba = [0, 0]
42. x, y = 30, 30
43.
44.
```

Niektoré funkcie a metódy tu ostanú nakopírované takmer bez zmeny, napríklad **klik(súradnice)**, s tým, že sme tu trošku upravili rozmery – súradnice a počet chlievikov závisí od levelu (od 2 do 6).

Všimnime si aj tu napríklad, že **chlievik_y** je tá takzvaná regulárna premenná – **podprogram má k nej prístup, no nemôže ju meniť.** Premenná **zvolená_farba** a **voľba** sa modifikujú, preto sme ich nastavili na **global**.

```
45.
46. def klik(súradnice):
47.     global zvolená_farba
48.     global voľba
49.     x, y = súradnice.x, súradnice.y
50.     zvolená_farba = kurzor(súradnice)
51.     chlievik_x = 30
52.     for i in range(len(voľba)):
53.         canvas.create_rectangle(chlievik_x, chlievik_y, chlievik_x+30, chlievik_y+30)
54.         if ((zvolená_farba != "white") and (x > chlievik_x and y > chlievik_y
55.                                             and x < chlievik_x+30 and y < chlievik_y+30)):
56.             canvas.create_oval(chlievik_x, chlievik_y, chlievik_x+30, chlievik_y+30,
57.                                fill = zvolená_farba, width = 3)
58.             voľba[i] = zvolená_farba
59.             break
60.         chlievik_x += 40
```

```
61.     print("Pole farieb", voľba)
62.
63.
```

Metóda **`zisťuj_pole()`** zostala tiež takmer bez zmeny:

```
64.
65. def zisťuj_pole():
66.     global voľba
67.     test = 0
68.     print("Zadané farby:", voľba)
69.     if len(voľba) == 0:
70.         test += 1
71.     for i in range(len(voľba)):
72.         if voľba[i] == 0:
73.             test += 1
74.     if test == 0:
75.         messagebox.showinfo(title = "Úspešné.", message = "Farby boli úspešne zadané.")
76.         return voľba
77.     else:
78.         messagebox.showerror(title = "Chyba.",
79.                              message = "Nezvolil si všetky pozície. Doplň svoju voľbu!")
80.         return "hocičo čo nie je farba"
81.
82.
```

Metóda **`ťahaj(súradnice)`** tiež:

```
83.
84. def ťahaj(súradnice):
85.     x, y = súradnice.x, súradnice.y
86.     canvas.delete("predchádzajúci")
87.     canvas.create_image(x, y, image = obrázok, tag = "predchádzajúci")
88.     canvas.update()
89.
90.
```

Aj metódu **`kurzor(súradnice)`** sme naprogramovali tak, aby na nej už nebolo treba takmer nič meniť – dokonca ani pozície misiek so špendlíkmi. Pridali sme však podmienky, **aby sa v nižších leveloch nedalo klikať na misky, ktoré majú byť k dispozícii len vo vyšších leveloch:**

```
91.
92.  def kurzor(súradnice):
93.      global obrázok
94.      global iba_raz
95.      x, y = súradnice.x, súradnice.y
96.      farba = "white"
97.
98.      if (x>10 and y>610 and x<130 and y<730):
99.          farba = "orangered"
100.         iba_raz = True
101.     elif (x>130 and y>610 and x<250 and y<730):
102.         farba = "gold"
103.         iba_raz = True
104.     elif (x>250 and y>610 and x<370 and y<730):
105.         farba = "yellowgreen"
106.         iba_raz = True
107.     elif (x>370 and y>610 and x<490 and y<730):
108.         farba = "dodgerblue"
109.         iba_raz = True
110.     elif (x>10 and y>730 and x<130 and y<850 and level > 1):
111.         farba = "saddlebrown"
112.         iba_raz = True
113.     elif (x>130 and y>730 and x<250 and y<850 and level > 2):
114.         farba = "purple"
115.         iba_raz = True
116.     elif (x>250 and y>730 and x<370 and y<850 and level > 3):
117.         farba = "darkgrey"
118.         iba_raz = True
119.     elif (x>370 and y>730 and x<490 and y<850 and level > 4):
120.         farba = "black"
121.         iba_raz = True
122.     else:
123.         if iba_raz:
124.             farba = zvolená_farba
125.             iba_raz = False
126.
127.     if farba != "white":
```

```
128.         obrázok = tkinter.PhotoImage(file = f"obrazky/{farba}_spendlik.png")
129.         winsound.Beep(500, 200)
130.     else:
131.         obrázok = tkinter.PhotoImage(file = "obrazky/otaznik.png")
132.
133.     if not(iba_raz):
134.         obrázok = tkinter.PhotoImage(file = "obrazky/otaznik.png")
135.
136.     return farba
137.
138.
```

Definovali sme si ešte niekoľko grafických funkcií. Podstata grafickej funkcie **`kresli_úvod()`** pozostáva z hlavnej časti grafického programu, ktorú sme do funkcie zabalili, pridali sme kreslenie rôznych líšt a textov do plátna (názov hry, výpis levelu, bodov) a kreslenie zámky na kôpky so špendlíkmi, ktoré nemajú byť v danom leveli k dispozícii. **`level`** je regulárna premenná – funkcia má k nej prístup na čítanie a na základe toho sa rozhoduje, ktorá kôpka so špendlíkmi sa odokryje a ktorá ostane „zamknutá“.

```
139.
140. def kresli_úvod():
141.     global špendlíky_1, špendlíky_2, špendlíky_3, špendlíky_4
142.     global špendlíky_5, špendlíky_6, špendlíky_7, špendlíky_8, zámka
143.
144.     canvas.delete("all")
145.     canvas.create_rectangle(25, 65, 185, 105, fill = "red", outline = "")
146.     canvas.create_rectangle(20, 60, 180, 100, fill = "yellow", outline = "")
147.     canvas.create_rectangle(325, 65, 485, 105, fill = "red", outline = "")
148.     canvas.create_rectangle(320, 60, 480, 100, fill = "yellow", outline = "")
149.     canvas.create_oval(150, 10, 350, 50, fill = "green", outline = "")
150.     canvas.create_text(250, 30, text = "LOGIK", font = "Arial 20 bold")
151.     canvas.create_text(100, 80, text = "Level: " + str(level), font = "Arial 15 bold")
152.     canvas.create_text(400, 80, text = "Body: " + str(výhra), font = "Arial 15 bold")
153.     canvas.create_rectangle(0, 540, 500, 600, fill = "darkkhaki", outline = "")
154.     canvas.create_rectangle(25, 115, 265, 155, fill = "khaki", outline = "maroon", width = 3)
155.     canvas.create_rectangle(265, 115, 475, 155, fill="peachpuff", outline="maroon", width=3)
156.
157.     canvas.create_rectangle(10, 610, 130, 730, fill = "orangered", outline = "")
158.     špendlíky_1 = tkinter.PhotoImage(file = "obrazky/orangered_spendliky.png")
159.     canvas.create_image(70, 670, image = špendlíky_1)
160.
161.     canvas.create_rectangle(130, 610, 250, 730, fill = "gold", outline = "")
162.     špendlíky_2 = tkinter.PhotoImage(file = "obrazky/gold_spendliky.png")
163.     canvas.create_image(190, 670, image = špendlíky_2)
164.
165.     canvas.create_rectangle(250, 610, 370, 730, fill = "yellowgreen", outline = "")
166.     špendlíky_3 = tkinter.PhotoImage(file = "obrazky/yellowgreen_spendliky.png")
167.     canvas.create_image(310, 670, image = špendlíky_3)
168.
169.     canvas.create_rectangle(370, 610, 490, 730, fill = "dodgerblue", outline = "")
170.     špendlíky_4 = tkinter.PhotoImage(file = "obrazky/dodgerblue_spendliky.png")
171.     canvas.create_image(430, 670, image = špendlíky_4)
172.
173.     canvas.create_rectangle(10, 730, 130, 850, fill = "saddlebrown", outline = "")
174.     canvas.create_rectangle(130, 730, 250, 850, fill = "purple", outline = "")
175.     canvas.create_rectangle(250, 730, 370, 850, fill = "darkgrey", outline = "")
176.     canvas.create_rectangle(370, 730, 490, 850, fill = "black", outline = "")
177.
178.     zámka = tkinter.PhotoImage(file = "obrazky/zamknute.png")
179.
180.     canvas.create_image(70, 790, image = zámka)
181.     canvas.create_image(190, 790, image = zámka)
182.     canvas.create_image(310, 790, image = zámka)
183.     canvas.create_image(430, 790, image = zámka)
184.
185.     if level > 1:
186.         špendlíky_5 = tkinter.PhotoImage(file = "obrazky/saddlebrown_spendliky.png")
187.         canvas.create_image(70, 790, image = špendlíky_5)
188.
189.     if level > 2:
190.         špendlíky_6 = tkinter.PhotoImage(file = "obrazky/purple_spendliky.png")
191.         canvas.create_image(190, 790, image = špendlíky_6)
192.
193.     if level > 3:
194.         špendlíky_7 = tkinter.PhotoImage(file = "obrazky/darkgrey_spendliky.png")
195.         canvas.create_image(310, 790, image = špendlíky_7)
196.
197.     if level > 4:
```

```
198.          špendlíky_8 = tkinter.PhotoImage(file = "obrazky/black_spendliky.png")
199.          canvas.create_image(430, 790, image = špendlíky_8)
200.
201.
```

Ďalšia grafická funkcia **`kresli_záver()`** vypisuje hráčovi gratuláciu po ukončení levelu alebo informácie o prehre spolu s malou animáciou (kreslenie optického obrazca z čiarok). Na záver vykreslí štvorec, ktorý prekryje neaktuálnu časť plátna tak, **aby sa nemuselo celé vymazať a prekresliť nanovo.**

```
202.
203.  def kresli_záver():
204.      canvas.delete("all")
205.      anim_x = 107
206.      anim_y = 0
207.      canvas.create_text(250, 500, text = info, font = "Arial 30 bold")
208.      canvas.create_text(250, 550, text = "Máš " + str(výhra) + " bodov", font="Arial 30 bold")
209.
210.      for i in range(21):
211.          canvas.create_line(anim_x+anim_y+50, 150, anim_x+250, anim_y+150, fill = "red")
212.          canvas.create_line(anim_x+anim_y+50, 150, anim_x+50, -anim_y+350, fill = "green")
213.          canvas.create_line(anim_x+anim_y+50, 350, anim_x+250, -anim_y+350, fill = "blue")
214.          canvas.create_line(anim_x+anim_y+50, 350, anim_x+50, anim_y+150, fill = "orange")
215.          canvas.update()
216.          canvas.after(100)
217.          anim_y += 10
218.
219.      canvas.create_rectangle(0, 110, 500, 540, fill = "paleturquoise", outline = "")
220.      canvas.after(2000)
221.
222.
```

Grafická funkcia **`kresli_voľbu()`** po úspešnom zadaní farebnej variácie túto variáciu **prekreslí** (čím prekryje pomocné chlieviky) a napravo **vykreslí čierne a biele hodnotiace špendlíky.** Zároveň, ak sme ešte neuhádli a ak to nebol posledný pokus, **nakreslí ďalší riadok na hádanie** – ukladanie novej variácie.

```
223.
224.  def kresli_voľbu():
225.      canvas.create_rectangle(25, chlievik_y - 5, 265, chlievik_y + 35,
226.                              fill = "khaki", outline = "maroon", width = 3)
227.      canvas.create_rectangle(265, chlievik_y - 5, 475, chlievik_y + 35,
228.                              fill = "peachpuff", outline = "maroon", width = 3)
229.
230.      x = 270
231.      for i in pole_vyhodnotenie:
232.          canvas.create_oval(x + 10, chlievik_y + 10, x + 20, chlievik_y + 20,
233.                             fill = i, outline = "red", width = 1)
234.          x += 33
235.
236.      x = 30
237.      for i in pomôcka:
238.          canvas.create_oval(x, chlievik_y, x + 30, chlievik_y + 30,
239.                             fill = i, outline = "black", width = 3)
240.          x += 40
241.
242.      if čierna < level+1:
243.          canvas.create_rectangle(25, chlievik_y + 35, 265, chlievik_y + 75,
244.                                  fill = "khaki", outline = "maroon", width = 3)
245.          canvas.create_rectangle(265, chlievik_y + 35, 475, chlievik_y + 75,
246.                                  fill = "peachpuff", outline = "maroon", width = 3)
247.
248.      canvas.update()
249.      canvas.after(200)
250.
251.
```

`klik` a **`ťahaj`** je, samozrejme, potrebné zviazať s udalosťami klikania a ťahania myši.

```
252.
253.  canvas.bind('<Button-1>', klik)
254.  canvas.bind('<Motion>', ťahaj)
255.
256.
```

Všimnite si, že všetky grafické záležitosti hry sme si **vopred definovali do niekoľkých funkcií,** ktoré sme výstižne pomenovali a v nasledujúcej logickej časti hry ich budeme **len v požadovanej chvíli volať,** nech si „vykonajú svoju prácu“, prípadne meniť, privážať parametre alebo získavať návratové hodnoty.

Poďme opísať zmeny a doplnenia logickej časti hry. Hneď prvý príkaz v prvom cykle **while** je volanie funkcie **kresli_úvod()**, ktorá nakreslí všetky úvodné záležitosti hry.

Pribudli dve polia: **pole_vyhodnotenie** a **pomôcka**.

- ✓ **pole_vyhodnotenie** slúži len na to, aby funkcia **kresli_voľbu()** vedela nakresliť v cykle požadovaný počet čiernych a/alebo bielych špendlíkov. To sa ukladá do poľa podľa toho, akú majú premenné **čierna** a **biela** hodnotu.
- ✓ **pomôcka** je pole, do ktorého sa ukladá návratová hodnota funkcie **zisťuj_pole()** a na základe nej sa, ak je pole platné (ak je celé zadané), kopíruje obsah do **pole_hádané**, ktoré sa porovnáva s poľom **pole_upravené**. Keby sa totiž použilo len **pole_hádané** bez **pomôcka**, tak pri zadaní len niekoľkých farieb a stlačení tlačidla „Skontroluj“ by sa celé pole vymazalo a farby by bolo treba naklikať znova a tiež by sa nevedela variácia prekresliť (bez chlievikov) vo funkcii **kresli_voľbu()**. Vyskúšajte si to sami – nepoužiť pomocné polia **pole_upravené**, ktoré je kópiou poľa **pole_generované** a pole **pomôcka**, ktoré je kópiou poľa **pole_hádané**.

Namiesto vstupu zo shellu teraz chceme návratovú hodnotu premennej **zisťuj_pole()** – veď to bol hlavný dôvod programovania samostatnej grafickej časti. **Je to však (veľký) problém,** prečo? Uvedomte si, že **celý cyklus while je vlastne štruktúrované** (resp. procedurálne) **programovanie** a podprogramy (metódy), ktoré sú zviazané s klikaním alebo ťahaním myši, sú vlastne **udalosťami riadené.** Ešte nikdy doteraz sme sa nestretli s ničím takým – buď sa program vykonal celý štruktúrovane a procedurálne a maximálne čakal na „nejaký ten **input()**“ a pokračoval ďalej, alebo sme sa len hrali s animáciami, udalosťami po kliknutí myši, stlačení klávesu či ťahaní myši, nikdy sme ich však nespájali. **A to sa ani nesmie robiť – keď sa budete učiť objektovo orientované programovanie (ktoré má k udalosťami riadenému programovaniu veľmi blízko), zistíte, že činnosť programu nikdy NESMIE BYŤ RIADENÁ CYKLOM, ALE VŠETKO MUSIA BYŤ METÓDY, ktoré reagujú na udalosti alebo reagujú na seba navzájom.**

Napriek tomu existuje v Pythone taká konštrukcia, ktorá dovolí „umelo“ pozastaviť vykonávanie hlavných príkazov (štruktúrovaných) a ak nastane očakávaná udalosť, opäť v cykle pokračovať. Čo vlastne chceme? Chceme, aby na nás hlavný cyklus „počkal“, kým nestlačíme nejaké tlačidlo (čím „nasimulujeme“ a nahradíme čakanie cyklu na **input()**). My sme si to vygooglili [17], dovolili sme si to trošku upraviť a odporúčame vám vyskúšať si to:

```
1.  import tkinter
2.  canvas = tkinter.Canvas(width = 100, height = 100)
3.  canvas.pack()
4.
5.  var = tkinter.IntVar()
6.  tlačidlo = tkinter.Button(text = "Skontroluj", command = lambda: var.set(1))
7.  tlačidlo.pack(side = tkinter.BOTTOM)
8.
9.
10. print("Začiatok programu sa vykoná hneď.")
11.
12. tlačidlo.wait_variable(var)
13.
14. print("Po prvom čakaní.")
15.
16. tlačidlo.wait_variable(var)
17.
18. print("Po druhom čakaní.")
19.
20. tlačidlo.wait_variable(var)
21.
22. print("Koniec vykonávania programu.")
```

Ako vidíme, vykonávanie hlavných príkazov sa pozastaví **pred každým volaním metódy tlačidlo.wait_variable(var)**. Toto sme využili na riadku 293.

Tiež tu pribudol jeden test platnosti poľa (jeden test tu je zbytočný, účel spĺňal, keď hra bola v štádiu zadávania variácie cez shell, vedeli by ste to opraviť/vymazať?).

Nakoniec sa vynuluje **voľba** (pole, ktoré obsahuje toľko núl, koľko farieb sa háda) a globálna premenná **chlievik_y** sa posunie o 40 nadol, aby sa pri ďalšom pokuse dali špendlíky vkladať do ďalšieho riadka.

```
257. #*********************************************************
258.
259. while level <= 5:
260.     kresli_úvod()
261.     body = výhra + 10
262.     pole_generované = []
263.
264.     print("****************************")
265.     print("Level:", level)
266.
267.     for i in range(level+1):
268.         pomocný = random.choice(výber[:level+3])
269.         pole_generované += [pomocný]
270.
271.     print(pole_generované)
272.
273.     if level+1 < 5: print(f"Vyberáš {level+1} farby z týchto: {výber}")
274.     else: print(f"Vyberáš {level+1} farieb z týchto: {výber}")
275.
276.     pokus = 0
277.     prehra = 0
278.
279.     while čierna != level+1:
280.         info = "Zadaj znova."
281.         čierna = 0
282.         biela = 0
283.         pole_upravené = []
284.         pole_hádané = []
285.         pole_vyhodnotenie = []
286.         pomôcka = []
287.
288.         for i in pole_generované: pole_upravené += [i]
289.
290.         print("Pokus č.", pokus+1)
291.
292.         #vstup = input("Zadajte variáciu (napr. ADDE): ")
293.         tlačidlo.wait_variable(var)
294.
295.         pomôcka = zisťuj_pole()
296.
297.         if len(pomôcka) != level+1:
298.             print("Zadali ste neplatný vstup.")
299.         else:
300.             for i in range(level+1):
301.                 if (pomôcka[i] != "orangered" and pomôcka[i] != "gold"
302.                     and pomôcka[i] != "yellowgreen" and pomôcka[i] != "dodgerblue"
303.                     and pomôcka[i] != "saddlebrown" and pomôcka[i] != "purple"
304.                     and pomôcka[i] != "darkgrey" and pomôcka[i] != "black"):
305.                     skúška = 1
306.                 else:
307.                     skúška = 0
308.
309.         if len(pomôcka) != level+1 or skúška != 0:
310.             print("Zadali ste neplatný vstup.")
311.         else:
312.             for i in pomôcka: pole_hádané += [i]
313.             pokus += 1
314.
315.             for i in range(level+1):
316.                 if pole_generované[i] == pomôcka[i]:
317.                     čierna += 1
318.                     pole_upravené[i] = "X"
319.                     pole_hádané[i] = "X"
320.
321.             for i in range(level+1):
322.                 for j in range(level+1):
323.                     if pole_hádané[i] != "X":
324.                         if pole_upravené[j] != "X":
325.                             if pole_hádané[i] == pole_upravené[j]:
326.                                 biela += 1
```

```
                                pole_hádané[i] = "X"
                                pole_upravené[j] = "X"

            if čierna == level+1:
                body = body - pokus + 1
                print(f"Vyhral si na {pokus}. pokus.")
                info = f"Máš {body} bodov."
                print(info)
                výhra = body
                print()

            if pokus == 10 and čierna != level+1:
                info = "Prehral si."
                print(info)
                prehra = 1
                break

            for i in range(čierna): pole_vyhodnotenie += ["black"]
            for i in range(biela): pole_vyhodnotenie += ["white"]

            kresli_voľbu()

            print("Pole hádaných:", pole_hádané)
            print("Pole upravených:", pole_upravené)

            print("Čiernych špendlíkov:", čierna)
            print("Bielych špendlíkov:", biela)

            chlievik_y += 40
            voľba = [0]
            voľba = voľba * (level+1)

    if prehra == 1:
        chlievik_y = 555
        x = 210
        print("Prehral si.")
        level = "Koniec hry."
        canvas.create_text(105, 555, text = "Prehral si.", font = "Arial 10 bold")
        canvas.create_text(105, 580, text = "Správne farby boli:", font = "Arial 10 bold")
        winsound.PlaySound("obrazky/prehra.wav", winsound.SND_FILENAME)

        for i in pole_generované:
            canvas.create_oval(x, chlievik_y, x+30, chlievik_y+30,
                               fill = i, outline = "black", width = 3)
            x += 40

        canvas.update()
        canvas.after(15000)
        koniec = tkinter.PhotoImage(file = "obrazky/zaver.png")
        canvas.create_image(250, 430, image = koniec)
        break
    else:
        canvas.create_rectangle(0, 110, 500, 540, fill = "paleturquoise", outline = "")
        chlievik_y = 120

        if level < 5:
            winsound.PlaySound("obrazky/vyhra.wav", winsound.SND_FILENAME)
            info = "Výborne, uhádol si!"
            level += 1
            canvas.update()
            canvas.after(200)
            kresli_záver()
            voľba = []
            voľba = [0] * (level+1) #tu musí byť zátvorka
        else:
            winsound.PlaySound("obrazky/vyhra-zaver.wav", winsound.SND_FILENAME)
            info = "Gratulujeme, vyhral si!!!"
            level += 1
            canvas.update()
            canvas.after(5000)
            kresli_záver()
            koniec = tkinter.PhotoImage(file = "obrazky/zaver.png")
            canvas.create_image(250, 430, image = koniec)
            break

print("Koniec hry.")

```

Komentár od riadka 359:

Ak sme prehrali, kontrolná premenná **prehra** nadobudla hodnotu **1**. Potom **chlievik_y** posunieme na súradnicu 555, kde hráčovi prezradíme, aká mala byť správna variácia. Prehráme mu zvuk prehry a hru po 15 sekundách prekryjeme záverečným obrázkom.

Inak (ak sme vyhrali kolo), pole špendlíkov si zmažeme prekrytím obdĺžnikom, **chlievik_y** nastavíme na **120** (na prvý ďalší riadok) a:

- ak je **level** menší ako 5 (hra sa ešte neskončila), prehráme zvuk výhry a vypíšeme gratuláciu, pripočítame premennej **level** jednotku a zavoláme funkciu **kresli_záver()**, ktorá vykreslí animáciu a resetujeme pole **voľba**,
- ak je **level** rovný 5 (ak sme vyhrali), prehráme hráčovi záverečný zvuk výhry, tiež posunieme **level** o jednotku – aby sa mohol skončiť cyklus **while** a po 5 sekundách vykreslíme záverečný obrázok. (Je nevyhnutné v tomto prípade použiť príkaz **break**, keď už sme „vyskočili“ z cyklu **while** pripočítaním jednotky premennej **level**?)

Návrhy na optimalizáciu hry:

- ✓ Vedeli by ste, na základe toho, že už vieme, že nie je vhodné použiť cyklus **while** na riadenie celého udalosťami riadeného alebo objektovo orientovaného programu, prerobiť logickú časť hry do metód a tie riadiť udalosťami (napríklad pridaním ďalšieho tlačidla)?
- ✓ Aby sme tu využili všetky vedomosti, vrátane zápisu do textového súboru, doprogramujte do hry rebríček hráčov. Nech si hra po spustení vypýta vaše meno, na záver hry ho uloží aj s vašimi získanými bodmi do súboru zoradené podľa najlepších hráčov a desať najlepších hráčov na konci hry vypíše.

Multimediálne súbory použité v hre:

K dispozícii sú vám obrázky misiek so špendlíkmi, obrázky farebných špendlíkov kurzora, otáznik, zámka aj zvuky výhry a prehry – ich zdroje: [19], [20], [21], [22].

13.5 Námety na ďalšie hry

Jednoduchých hier, ktoré sa dajú takto implementovať, je niekoľko, napríklad:

- ✓ Hľadanie mín (Minesweeper)
- ✓ Hanojské veže
- ✓ tzv. Problém batožinového priestoru
- ✓ Sudoku

Ich pravidlá a algoritmy z dôvodu už aj tak veľkého rozsahu tejto učebnice nebudeme rozoberať. Navyše, (hlavne grafické) programovanie týchto hier odporúčame nechať na vtedy, keď sa budete učiť objektovo orientované programovanie, keďže sami vidíte, že čisto procedurálne programovanie „narazilo na svoje limity“ a už aj kód hry Logik začína byť trošku neprehľadný a dlhý. Na inšpiráciu na úplný záver ponúkame odkaz, kde je logická (a aj semigrafická) implementácia hry Hľadanie mín celkom pekne rozpísaná a vysvetlená [23]:

https://www.askpython.com/python/examples/create-minesweeper-using-python

14 Zdroje

Hlavné zdroje učebného textu:

- BLAHO, A.: *Programovanie v Pythone 2019/2020*. Dostupné na internete: https://input.sk/python2019/ (naposledy pristúpené 17.07.2022)
- BLAHO, A.: *Programovanie v Pythone*. 3. vydanie. [Údaj o vydavateľstve neuvedený], 2018. ISBN 978-80-8147-084-4. Dostupné na internete: https://input.sk/pdf/python.pdf (naposledy pristúpené 25.07.2022)

Námety na úlohy a príklady:

- **RNDr. Andrej Blaho, PhD.** (https://input.sk/, *didakticky upravené*):
  - úlohy č.: 8, 10, 11, 13, 14, 30
  - príklady č.: 8, 10, 11, 12, 13, 14, 15, 21, 22, 23, 24, 26, 28, 54

Webové zdroje (obrázky, schémy, kódy, definície):

[1] *10 Myths about Programming and Software Development.* Dostupné na internete: https://www.codingem.com/programming-myths/. (naposledy pristúpené 05.07.2022)

[2] *Vlajka Grónska.* Dostupné na internete: https://sk.wikipedia.org/wiki/Gr%C3%B3nsko#/media/S%C3%BAbor:Flag_of_Greenland.svg. (naposledy pristúpené 07.07.2022)
Vlajka Nórska. Dostupné na internete: https://sk.wikipedia.org/wiki/N%C3%B3rsko#/media/S%C3%BAbor:Flag_of_Norway.svg. (naposledy pristúpené 07.07.2022)
Vlajka Grécka. Dostupné na internete: https://sk.wikipedia.org/wiki/Gr%C3%A9cko#/media/S%C3%BAbor:Flag_of_Greece.svg. (naposledy pristúpené 07.07.2022)
Vlajka Česka. Dostupné na internete: https://sk.wikipedia.org/wiki/%C4%8Cesko#/media/S%C3%BAbor:Flag_of_the_Czech_Republic.svg. (naposledy pristúpené 05.07.2022)

[3] *Python Online – Koordinatengrafik*. Dostupné na internete: https://www.python-online.ch/gpanel.php?inhalt_links=gpanel/navigation.inc.php&inhalt_mitte=gpanel/while.inc.php. (naposledy pristúpené 08.07.2022)

[4] *File:Sin-cos-defn-I.png.* Dostupné na internete: https://en.m.wikipedia.org/wiki/File:Sin-cos-defn-I.png. (naposledy pristúpené 09.07.2022)

[5] *Wallis product.* Dostupné na internete: https://en.wikipedia.org/wiki/Wallis_product. (naposledy pristúpené 12.07.2022)

[6] *Leibniz formula for π*. Dostupné na internete: https://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80. (naposledy pristúpené 12.07.2022)

[7] *Deliteľnosť*. Dostupné na internete: https://sk.wikipedia.org/wiki/Delite%C4%BEnos%C5%A5. (naposledy pristúpené 14.07.2022)

[8] *Colors RGB*. Dostupné na internete: https://www.w3schools.com/colors/colors_rgb.asp. (naposledy pristúpené 15.07.2022)

[9] *Euklidov algoritmus – vysvetlenie*. Dostupné na internete: https://pdf.truni.sk/e-skripta/em6/data/e6b6ebdf-7e4c-4d60-9473-1b86e2c0338c.html?ownapi=1. (naposledy pristúpené 15.07.2022)

[10] *Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java*. Dostupné na internete: https://pythontutor.com/visualize.html#mode=edit. (naposledy pristúpené 18.07.2022)

[11] *Greatest common divisor*. Dostupné na internete: https://rosettacode.org/wiki/Greatest_common_divisor, upravené. (naposledy pristúpené 29.07.2022)

[12] *Python List VS Array VS Tuple*. Dostupné na internete: https://www.geeksforgeeks.org/python-list-vs-array-vs-tuple/. (naposledy pristúpené 16.07.2022)

[13] *Všetko o číselnej lotérii Loto 5 z 35.* Dostupné na internete: https://www.tipos.sk/loterie/loto-5-z-35/vsetko-o-ciselnej-loterii-loto-5-z-35. (naposledy pristúpené 25.07.2022)

[14] *Bubble Sort*. Dostupné na internete: https://www.youtube.com/watch?v=yIQuKSwPlro&ab_channel=KCAng. (naposledy pristúpené 17.07.2022)

[15] *Podprogram*. Dostupné na internete: https://sk.wikipedia.org/wiki/Podprogram. (naposledy pristúpené 18.07.2022)

[16] *Prečíslované telefónne predvoľby miest v SR (tabuľka)*. Dostupné na internete: https://tech.sme.sk/c/75517/precislovane-telefonne-predvolby-miest-v-sr-tabulka.html (naposledy pristúpené 30.07.2022)

[17] *Making Tkinter wait untill button is pressed*. Dostupné na internete: https://stackoverflow.com/questions/44790449/making-tkinter-wait-untill-button-is-pressed. (naposledy pristúpené 23.07.2022)

[18] *Logik (hra)*. Dostupné na internete: https://cs.wikipedia.org/wiki/Logik_(hra). (naposledy pristúpené 25.07.2022)

[19] *Free Win Sound Effects*. Dostupné na internete: https://mixkit.co/free-sound-effects/win/. (naposledy pristúpené 31.07.2022)

[20] *Download Red Push Pin transparent PNG*. Dostupné na internete: https://www.stickpng.com/img/icons-logos-emojis/pins/red-push-pin. (naposledy pristúpené 31.07.2022)

[21] *Push pins / Heap of colorful push pins*. Dostupné na internete: https://www.wallsheaven.com/wall-murals/push-pins-heap-of-colorful-push-pins.-top-view.-E309083271. (naposledy pristúpené 31.07.2022)

[22] *Free Vector Graphic Download. Lock and Key Collection*. Dostupné na internete: http://7428.net/2013/07/lock-and-key-collection.html. (naposledy pristúpené 31.07.2022)

[23] *Create Minesweeper using Python From the Basic to Advanced*. Dostupné na internete: https://www.askpython.com/python/examples/create-minesweeper-using-python. (naposledy pristúpené 31.07.2022)

Odporúčaná literatúra:

- KUČERA, P.: *Programujeme v Pythone – učebnice informatiky pre stredné školy*. 2016. Dostupné na internete: http://www.programujemevpythone.sk/. (naposledy pristúpené 31.07.2022)
- PECINOVSKÝ, R.: *Python – Kompletní příručka jazyka pro verzi 3.10*. První vydání. Praha : Grada Publishing, a. s., 2021. ISBN 978-80-8147-084-4