

# Programovanie v Pythone

Zbierka inovatívnych metodík pre stredné školy



EURÓPSKA ÚNIA  
Európsky sociálny fond  
Európsky fond regionálneho rozvoja



OPERAČNÝ PROGRAM  
ĽUDSKÉ ZDROJE



MINISTERSTVO  
ŠKOLSTVA, VEDY,  
VÝSKUMU A ŠPORTU  
SLOVENSKEJ REPUBLIKY



itAkadémia



*Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu  
v rámci Operačného programu Ľudské zdroje*

[www.minedu.sk](http://www.minedu.sk) [www.employment.gov.sk/sk/esf/](http://www.employment.gov.sk/sk/esf/) [www.itakademia.sk](http://www.itakademia.sk)

# **Programovanie v Pythone**

## **Zbierka inovatívnych metodík pre stredné školy**

Spracované v rámci národného projektu IT Akadémia – vzdelávanie pre 21. storočie

## **Programovanie v Pythone: Zbierka inovatívnych metodík pre stredné školy**

Spracované s finančnou podporou národného projektu IT Akadémia – vzdelávanie pre 21. storočie

Autori: Ján Guniš , Ľubomír Šnajder, Valentína Gunišová, Zuzana Tkáčová

Recenzenti: Gabriela Andrejková, Jana Plichtová

Editori: Ľubomír Šnajder, Ján Guniš

Vydavateľ: Centrum vedecko-technických informácií SR, Bratislava

Rok vydania: 2021

Vydanie : 1. vydanie

ISBN: 978-80-89965-82-3

EAN: 9788089965823

Bratislava 2021



Obsah podlieha licenci Creative Commons CC BY SA 4.0.

Dielo sa môže rozmnožovať, rozširovať, vystavovať dielo a odvodené diela za podmienky uvedenia autora. Je možné rozširovať odvodené diela len za podmienky použitia identickej licencie pre odvodené diela.

Ako citovať:

Guniš, J., Šnajder, Ľ., Gunišová, V. & Tkáčová, Z. (2021). Programovanie v Pythone: Zbierka inovatívnych metodík pre stredné školy. Bratislava: Centrum vedecko-technických informácií SR. Dostupné na Internet: <https://registracia.itakademia.sk/admin/theme/download/zim-python.pdf>

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu v rámci Operačného programu Ľudské zdroje.

## OBSAH

Obsah	4
Úvod	6
Konceptia výučby programovania v jazyku Python	7
Konceptia programovania v jazyku Python	19
01 Úvod do programovania, výpočty v konzole	40
02 Korytnačia grafika	54
03 Vlastné funkcie bez parametrov a bez návratovej hodnoty	65
04 Cyklus s pevným počtom opakovaní	80
05 Opakovanie I. + didaktický test	92
06 Vlastné funkcie s parametrami – kresliace úlohy	103
07 Vlastné funkcie s parametrami a výstupom – výpočtové úlohy	116
08 Chyby vo výpočtoch, ich rozpoznávanie a odstraňovanie	125
09 Podmienенý príkaz	133
10 Opakovanie II. + didaktický test	144
11 Reťazce	151
12 Reťazcové metódy, zložené a vnorené podmienky	161
13 Algoritmy s reťazcami	171
14 Odchytávanie výnimiek	178
15 Generovanie výnimiek	188
16 Opakovanie III. + didaktický test	199
17 Zoznamy a metódy zoznamov	205
18 Vytváranie a modifikácia zoznamov, použitie náhody	215
19 Algoritmy so zoznamami	226
20 Cyklus s podmienkou	234
21 Vnorené riadiace štruktúry	244
22 Opakovanie IV. + didaktický test	253
23 Grafické používateľské rozhranie – tlačidlá, textové polia, popisy	261

24 Grafické používateľské rozhranie – plátno	271
25 Komplexný projekt – výber problému, analýza a návrh riešenia problému	282
26 Komplexný projekt – implementácia riešenia problému	291
27 Komplexný projekt – finalizácia projektu + prezentácia a diskusia	296
Bibliografia	305
Príloha	306

## Úvod

Predmet informatika patrí k najdynamickejšim vyučovacím predmetom, ktorý by mal reflektovať súčasné poznatky v oblasti informatiky a digitálnych technológií a ich aplikácie v spoločnosti. K aktuálnym učebniciam informatiky však chýbajú metodické materiály, ktoré by usmerňovali učiteľov pri výučbe.

V rámci Národného projektu IT Akadémia – vzdelávanie pre 21. storočie (ďalej NPITA) sme sa primárne zamerali na tvorbu metodík pre učiteľa, ktorých súčasťou sú pracovné listy a pracovné súbory pre žiakov. Inováciu vyučovania informatiky chápeme nielen ako inováciu obsahu, ale aj inováciu pedagogických postupov využívajúcich napr. bádateľsky orientované vyučovanie, projektové vyučovanie, nástroje formatívneho hodnotenia.

V tejto publikácii uvádzame zbierku metodík pre vyučovanie programovania v jazyku Python na stredných školách, ktorá pokrýva vzdelávací štandard tematickej oblasti Algoritmické riešenie problémov uvedený v Inovovanom štátnom vzdelávacom programe so štvorročným a päťročným vzdelávacím programom (ďalej iŠVP) (Štátny pedagogický ústav, 2020).

V kapitole Konceptia vyučovania programovania v jazyku Python uvádzame pedagogické východiská tvorby metodík, obsahové zameranie metodík, štruktúru metodík a ich príloh, odporúčania k výučbe podľa týchto metodík.

V kapitole Konceptia programovania v jazyku Python uvádzame odporúčania pre správne písanie programov v jazyku Python a upozorňujeme na odlišné prístupy v porovnaní s inými programovacími jazykmi. Cieľom je, aby si učitelia a žiaci už od začiatku vytvárali správne návyky pri programovaní a písaní štandardného programového kódu v jazyku Python.

Na základe analýzy spätnej väzby z overovania metodík odporúčame, aby sa učiteľ vopred oboznámil s oboma koncepciami a „prijal za svoje“.

V ďalších 27 kapitolách uvádzame metodiky výučby 27 tém programovania v Pythone. Súčasťou publikácie je elektronická príloha s priečinkami pre učiteľa a pre žiaka, ktoré obsahujú pracovné súbory a pracovné listy s riešeniami úloh.

Obsah tejto publikácie je výsledkom štvorročnej autorskej práce štvorice autorov, ktorej súčasťou bolo dvojročné overovanie týchto materiálov na vybraných stredných školách zapojených do NPITA. Ďakujeme 44 stredoškolským učiteľom, ktorí v školských rokoch 2018/2019 a 2019/2020 overili naše metodiky vo svojej výučbe a tiež recenzentom doc. RNDr. Gabriele Andrejkovej, CSc. a RNDr. Jane Plichtovej, ktorí svojimi komentármi a pripomienkami prispeli ku kvalite finálnych verzií uvedených metodík a ich príloh. Veríme, že predložená zbierka metodík osloví veľkú skupinu učiteľov informatiky, ktorí začínajú alebo už vyučujú programovanie v jazyku Python a hľadajú komplexnú podporu ich výučby, ktorá je navyše pre nich voľne prístupná.

Autori

# KONCEPCIA VÝUČBY PROGRAMOVANIA V JAZYKU PYTHON

## PEDAGOGICKÉ VÝCHODISKÁ TVORBY METODÍK

**Výučba programovania** ako súčasť tematickej oblasti Algoritmické riešenie problémov má obzvlášť významné miesto vzhľadom k jej kľúčovej úlohe v oblasti rozvoja algoritmického myslenia, informatického myslenia, kritického uvažovania, kreativity žiakov a orientácii na tvorbu nástrojov využiteľných v ostatných oblastiach aj mimo informatiky. Pri tvorbe metodík v rámci NPITA sme vychádzali z uznávaných psychologických teórií učenia sa – **konštruktivismu** a **konštrukcionizmu**, ktoré sa premietajú, napr. do bádateľsky orientovaného vyučovania, problémového či projektového vyučovania.

## TYPY METODÍK

Vo vytvorenej sade metodík k výučbe programovania v Pythone sme použili nasledovné typy metodík:

- **Bádateľsky orientovaná metodika** (model 5E) – je zameraná na aktívne a čo najviac samostatné porozumenie informatickým konceptom a princípom žiakmi a rozvíjanie ich informatického myslenia a bádateľských spôsobilostí.
- **Projektovo zameraná metodika** – je zameraná na tvorivú činnosť žiakov, ktorej výsledkom sú hodnotné produkty prínosné pre autora a jemu blízku komunitu.
- **Metodika zameraná na zopakovanie a systemizáciu učiva** – je zameraná na upevnenie a systemizáciu učiva z rozsiahlejšieho celku. Súčasťou niektorých metodík je didaktický test s javovou analýzou.

### *Bádateľsky orientovaná metodika (model 5E)*

Bádateľsky orientovaná metodika (model 5E) je najčastejším typom metodík, ktoré sme použili v projekte NPITA. Metodika umožňuje žiakom, aby s určitou podporou učiteľa aktívne a čo najviac samostatne skúmali a objavovali nové informatické koncepty a princípy.

Na začiatku metodiky v časti Priebeh výučby uvádzame osnovu hodiny obsahujúcu názov jednotlivých fáz (Zapojenie, Skúmanie, Vysvetlenie, Rozpracovanie, Vyhodnotenie), ich približné časové trvanie a stručný popis priebehu výučby v uvedených fázach. V metodikách sú najviac časovo dotované fázy Skúmanie a Rozpracovanie, v ktorých sú žiaci dominantne aktívni. Aj v týchto fázach má učiteľ určitý priestor na svoju realizáciu, len jeho podpora je skôr individuálna na rozdiel od frontálneho charakteru vo fázach Zapojenie, Vysvetlenie a Vyhodnotenie.

Fázy modelu (učebného cyklu) 5E:

- **Zapojenie (Engage):**
  - Cieľom tejto fázy je motivovať žiakov pre skúmanie uvedenej oblasti a zistiť ich prvotné predstavy o skúmanej problematike. Veľmi často ide o frontálnu formu výučby vedenú učiteľom (uvedenie krátkeho príbehu, zadanie zaujímavej úlohy, diskusia). Na konci Zapojenia by mali žiaci dospieť k tomu, čo je zaujímavé/potrebné preskúmať, a tým získať vlastnú predstavu o cieľoch vyučovacej hodiny. Prvotné predstavy žiakov o skúmanej problematike vieme zistiť pomocou riešenia úloh a diskusie so žiakmi.
- **Skúmanie (Explore):**
  - V tejto fáze žiaci realizujú viaceré podnetné aktivity (často v skupinách), pomocou ktorých prichádzajú k rôznym predstavám a hypotézam ako funguje skúmaný systém (program, príkaz). Veľmi dôležitá je podpora od učiteľa (angl. scaffolding) formou využitia rôznych pomôcok (napr. hotových či nedokončených kódov programov, diagramov, tabuliek, návodov) a hlavne učiteľovým slovným usmernením či provokatívnymi otázkami, nie učiteľovým predkladaním hotových riešení. Učiteľ v tejto fáze prechádza pomedzi žiakov a sleduje ich prácu – v prípade nejasností im objasní zadanie úlohy, preformuluje úlohu alebo im pomôže doplňujúcimi otázkami. Úlohy v tejto fáze sú často formulované spôsobom „preskúmajte čo robí program/ako funguje systém“, „doplňte program, aby robil ...“, pri ktorých žiaci majú k dispozícii pracovné súbory – programy či údaje. Odpovede žiakov v pracovných listoch zachycujú aktuálny stav ich poznania, majú možnosť uviesť, že daný prvok učiva nevedia, čo súvisí s rozvíjaním ich metakognície. V každej metodike uvádzame kľúčové prvky, na ktoré je skúmanie zamerané. V tejto fáze nemusia žiaci nutne objaviť a preskúmať všetko s čím sa v metodike stretnú. Napr. stačí ak objavia, že reťazce majú metódy (na príklade dvoch metód). V tradičnej výučbe sa táto fáza vynecháva.
- **Vysvetlenie (Explain):**
  - V tejto fáze by si mali žiaci „poupratovať“ vo vlastnej hlave, uvedomiť si k čomu dospeli a formulovať svoje predstavy o tom ako funguje skúmaný systém, ktorý skúmali v predchádzajúcej fáze. Učiteľ vyzýva všetkých žiakov, aby vysvetlili ako rozumejú práve osvojovanému učivu. Diskutuje so žiakmi a koriguje ich predstavy o skúmanej problematike. Na konci žiackeho vysvetľovania zhrnie zažitú skúsenosť žiakov a pomenuje ich odborným slovníkom. Vo fáze Vysvetlenia by sme mali sledovať len zopár hlavných cieľov a nezaťažovať žiakov nepodstatnými detailmi. Preto v metodikách zámerne neuvádzame úplný zoznam príkazov a ich variácií. Niektoré príkazy prezradíme žiakom až keď nastane vhodná situácia, kedy sú potrebné (napr. `hideturtle()` či `delay(0)`). V tejto časti učiteľ môže použiť rôzne pomôcky (tabuľky, diagramy, grafy, zdrojové programové kódy) podporujúce pochopenia učiva rôznymi učebnými štýlmi. Túto fázu môže učiteľ doplniť o ďalšie informácie, ktoré môžu žiaci využiť vo fáze Rozpracovanie. Napríklad okrem metód reťazcov, ktoré sme objavili vo fáze Skúmanie, sú užitočné aj nasledujúce dve-tri metódy (pozor, v žiadnom prípade nenasleduje komplet manuál k reťazcom, možno len odkaz kde sa dá dozvedieť viac). V metodike uvádzame, o čo by mal učiteľ vysvetlenie žiakov doplniť. Z tejto časti by si mali žiaci spraviť výťah a ten zaznamenať do pracovných listov.



- **Rozpracovanie (Elaborate):**

- Podobne ako Skúmanie aj táto fáza je viac časovo dotovaná s viacerými úlohami pre žiakov. Cieľom tejto fázy je, aby si žiaci precvičili a prehĺbili osvojované učivo. Okrem analytických úloh typu „analyzujte program“, „nájdite a opravte chybu v programe“ predkladáme žiakom aj kreatívne úlohy typu „dopracujte/rozšírte program“, „vytvorte program“. Riešenia týchto úloh sú pre učiteľa zdrojom pre získanie predstavy o žiackych miskoncepciách. Aj v tejto fáze je možná vzájomná spolupráca žiakov.

- **Vyhodnotenie (Evaluate):**

- Aj keď počas hodiny učiteľ používa viaceré formy formatívneho hodnotenia (napr. spätnú väzbu k riešeniu úloh, diskusiu k žiackym vysvetleniam učiva), na záver hodiny by mal učiteľ poskytnúť žiakom príležitosť k vyjadreniu subjektívneho sebaaponímania úrovne osvojených poznatkov a získaných skúseností (napr. sebahodnotiacou kartou) či získaniu objektívnej spätnej väzby k úrovni osvojených poznatkov a získaných skúseností (napr. riešením úlohy didaktického testu). Je veľmi dôležité, aby učiteľ poskytol žiakom čo najskoršiu spätnú väzbu k ich úrovni osvojenia si nových poznatkov.

Fázy modelu 5E majú určené poradie, ktoré nie je možné meniť. Rovnako nie je možné niektorú fázu vynechať.

### *Projektovo orientovaná metodika*

Tento typ metodík sme použili v záverečných troch metodikách. Projektové vyučovanie je v odbornej literatúre chápané rôznymi autormi rôzne – niekedy ako metóda výučby (príp. komplexná metóda výučby), inokedy ako organizačná forma vyučovania a stretnúť sa môžeme aj označením ako výchovno-vzdelávacia stratégia.

Podľa (Čapek, 2015) **projekt** predstavuje sofistikovanú úlohu zameranú na praktický produkt s jedinečným riešením, ktoré vyžaduje od žiaka autorský vklad. Charakteristickým znakom projektového vyučovania je **prebratie zodpovednosti** za riešenie projektu samotnými žiakmi, ktorí majú možnosť samostatne rozhodnúť, ako celý projekt realizovať, čoho dôsledkom sú vzájomne **odlišné produkty** žiakov. Ďalším typickým znakom projektu je výrazná **medzipredmetovosť** (projekt využíva poznatky a zručnosti z rôznych predmetov, tematických oblastí alebo odborov) a taktiež **prepojenosť s realitou** (žiaci riešia konkrétny problém alebo produkt zo života, nie rutinnú nezáživnú úlohu nepoužiteľnú v praxi). Projektové vyučovanie tak vytvára priestor pre efektívnu integráciu poznatkov a prepojenie na rozvoj kľúčovým kompetencií žiaka. Umožňuje rozvíjať kreativitu a samostatnosť žiakov. Súčasťou projektu je potreba samostatného objavovania poznatkov žiakmi počas riešenia. Na základe uvedeného vyplýva, že projektové vyučovanie:

- nie je žiacky referát, pri ktorom žiaci vyhľadávajú na zadanú tému informácie na internete a spracúvajú ich do výslednej prezentácie alebo plagátu (nie je to reálny produkt zo života),
- nie je zadanie samostatnej alebo skupinovej aplikačnej úlohy po predošlom výklade učiteľa (žiaci nemusia objavovať nové poznatky),

- nie je zadanie úlohy, pri ktorej žiaci nemajú možnosť individuálnej voľby činnosti v rámci riešenia (výstup očakáva, že všetci žiaci budú kresliť, písať, programovať a pod.)
- nie je postupnosť krátkych učiteľom zadáných úloh, aj keď ich výsledkom môže byť v praxi užitočný produkt (žiak nedospeje k produktu na základne vlastného plánovania a rozhodovania).

Riešenie projektu pozostáva zo štyroch etáp (Turek, 2008):

- **Voľba témy projektu** – jej špecifikácia, určenie cieľov – produktov projektu,
- **Plánovanie riešenia projektu** – vypracovanie postupu, určenie čiastkových úloh, rozdelenie úloh medzi riešiteľov projektu, určenie termínov jednotlivých úloh projektu,
- **Riešenie projektu** – realizácia plánu projektu žiakmi, učiteľ je v roli pomocníka, oponenta, podnecovateľa atď.
- **Zverejnenie výsledkov riešenia projektu** – prezentácia výsledkov a zhodnotenie práce na projekte.

*Metodika zameraná na zopakovanie a systemizáciu učiva*

Tento typ metodiky použijeme pri rozsiahlejších témach pre lepšiu fixáciu a systemizáciu vedomostí a zručností žiakov.

Žiaci počas hodiny riešenia gradované úlohy, pokrývajúce prebrané učivo s prepojením na predchádzajúce učivo. Učiteľ poskytuje podporu žiakom, jednak individuálne usmerňujúcich dialógom či pomocnou úlohou, jednak hromadne upozornením či vysvetlením špecifického problému pri riešení danej úlohy.

Súčasťou metodiky okrem zadaní a riešení gradovaných úloh môže byť aj **zadanie a riešenie didaktického testu** (ako nástroja sumatívneho hodnotenia žiakov) včítanie jeho skórovania, ktorý bude administrovaný na nasledujúcej vyučovacej hodine.

## CHARAKTERISTIKA KONCEPTOV INFORMATICKÉHO MYSLENIA

Informatické myslenie sa často poníma ako súbor na predmete nezávislých schopností žiakov nevyhnutných pre vzdelávanie a život v 21. storočí.

**Informatické myslenie** (angl. computational thinking) môžeme podľa (BCS Barefoot, 2019) vymedziť ako efektívne riešenie problémov s pomocou počítača alebo bez neho. Informatické myslenie je komplexná schopnosť tvorená viacerými **konceptmi**. Viaceré významné vzdelávacie spoločnosti (napr. Computer Science Teachers Association, International Society for Technology in Education, Computing at School) navrhli vlastné rámce konceptov informatického myslenia. Pre naše výskumné a pedagogické účely sme vybrali šesticu konceptov navrhnutých Computing at School, a to konkrétne:

- **Logika** – predpovedaj, analyzuj  
(logicky vyvodzovať závery z pozorovaní a experimentov, analyzovať nejaký systém/program a predpovedať jeho správanie sa)
- **Algoritmy** – vytváraj kroky a pravidlá  
(zostavovať postupy činnosti pre nejakého vykonávateľa (algoritmy, programy, scenáre, storyboardy) a tiež využívať, upravovať, vylepšovať vytvorené postupy (algoritmy, programy, scenáre, storyboardy))
- **Dekompozícia** – rozdeľ na časti  
(rozdeľovať problém na ľahšie podproblémy, riešenie ktorých bude využiteľné pri riešení pôvodného problému)
- **Hľadanie vzorov** – rozpoznaj a využívaj podobnosti  
(rozpoznávať a určovať v systéme rovnaké/podobné časti/vlastnosti/pravidlá a využívať nájdené vzory pri rôznych činnostiach/riešení problému)
- **Abstrakcia** – vyber podstatné, odhľadni od menej podstatného  
(určovať, ktoré detaily/prvky/vlastnosti/vzťahy systému sú v danej situácii podstatné a ktoré môžeme zanedbať)
- **Vyhodnotenie** – rob rozhodnutia  
(určovať relevantné kritériá hodnotenia a podľa nich vyhodnocovať projekt/program/algoritmus)

V NPITA sme detailnejšie charakterizovali uvedené koncepty do podkonceptov, čo nám umožnilo v metodikách lepšie popísať, ktoré časti jednotlivých konceptov rozvíjame.

<b>logika</b> predikcia a analýza	<ul style="list-style-type: none"> <li>• (LOG1) využitím logických zdôvodnení <b>predpokladať správanie sa algoritmov</b> (návodov, súborov pravidiel, matematických výpočtov, napr.: cesta cez bludisko, hľadanie v slovníku, riešenie sudoku)</li> <li>• (LOG2) využitím logických zdôvodnení <b>predpokladať správanie sa jednoduchých programov</b></li> <li>• (LOG3) využitím logických zdôvodnení <b>detegovať a opravovať chyby</b> v programoch a algoritmoch</li> <li>• (LOG4) <b>vyvodzovať</b> (logicky zdôvodňovať) <b>závery z pozorovaní a experimentov</b> (aj myšlienkových)</li> <li>• (LOG5) logicky <b>zdôvodniť rozdelenie</b> algoritmu/programu/problému/objektu na menšie časti</li> <li>• (LOG6) logicky <b>zdôvodniť zmenu</b> algoritmu/programu</li> <li>• (LOG7) na základe výsledku <b>dedukovať</b> predchádzajúci stav/východiskový stav</li> <li>• (LOG8) z existujúcich pravidiel logicky <b>odvádzať iné pravidlá</b></li> </ul>
<b>algoritmy</b> vytváranie krokov a pravidiel	<ul style="list-style-type: none"> <li>• (ALG1) <b>rozpoznať</b> či postup/návod <b>je algoritmom</b> (algoritmus – zrozumiteľné/jednoznačné pravidlá/kroky/inštrukcie poskytujúce výsledky v konečnom čase, napr. aj postup experimentu, postup geometrickej konštrukcie, vedecká metóda – rámec výskumu) alebo nie je algoritmom (nejednoznačnosť postupu, nezrozumiteľnosť, výsledky nie sú v konečnom čase)</li> <li>• (ALG2) <b>vykonávať algoritmus</b> (bez použitia vlastností/schopností, ktoré vykonávateľ nemá, napr. simulovať činnosť robota, hrať hru podľa pravidiel)</li> <li>• (ALG3) <b>vytvárať vlastné algoritmy</b> riešiace <b>problém/časti problému</b> (postupnosti krokov na realizáciu nejakej činnosti vedúcej k cieľu, vytvárať scenáre a storyboardy, postup experimentu)</li> </ul>

	<ul style="list-style-type: none"> <li>• (ALG4) <b>vytvárať vlastné algoritmy</b>, ktoré <b>pracujú s množinou</b>/modifikujú množinu <b>dát</b> (napr. usporiadanie kníh na poličke, párovanie ponožiek po praní)</li> <li>• (ALG5) <b>využívať existujúce</b> (vlastné/cudzie) <b>algoritmy</b> v návrhu vlastných algoritmov (z algoritmov riešiacich podproblémy zostaviť algoritmus riešiaci problém)</li> <li>• (ALG6) <b>modifikovať existujúce/dotvárať nekompletné algoritmy</b> (uprav/doplň programový kód)</li> <li>• (ALG7) <b>vylepšovať existujúce algoritmy</b> (zlepšenie efektívnosti, rozšírenie algoritmu na väčšiu množinu vstupov, rozšírenie funkcionality)</li> <li>• (ALG8) <b>zapísať algoritmy v</b> konkrétnom formálnom <b>jazyku</b> (programovacím jazyku, zápis geometrickej konštrukcie, notový zápis)</li> </ul>
<b>dekompozícia</b> delenie na časti	<ul style="list-style-type: none"> <li>• (DEK1) lineárna dekompozícia – <b>lineárne rozdeliť</b> objekty/problémy/procesy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (detekcia prvkov/určenie objektov/správania sa v nejakej hre – pozadie/postava/udalosti, rozdeliť prácu pre členov tímu (rovnomerná záťaž, uvažovanie o prerekvizitách, časový harmonogram a pod.))</li> <li>• (DEK2) hierarchická dekompozícia – <b>hierarchicky rozdeliť</b> objekty/problémy/procesy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (vytvoriť pojmovú mapu, diagram, zobrazíť hierarchiu, aj podproblémy rozdeliť na menšie podproblémy, rozdeliť prácu pre členov hierarchického tímu ktorí delegujú prácu na podriadených)</li> <li>• (DEK3) rekurzívna dekompozícia – <b>rekurzívne rozdeliť</b> objekty/problémy/procesy na menšie, ale celku podobné časti tak, aby sa dali využiť pre dosiahnutie cieľa (rekurzia, rozdeľuj a panuj, matematická indukcia)</li> </ul>
<b>hľadanie vzorov</b> rozpoznaj a využívaj podobnosti	<ul style="list-style-type: none"> <li>• (VZO1) <b>rozpoznať časti</b> objektu/problému/procesu, ktoré majú rovnaké/podobné vlastnosti/pravidlá správania sa (semafor na križovatke/systém semaforov na križovatke, premenná v cykle, člen postupnosti, časti vety)</li> <li>• (VZO2) <b>určiť rovnaké/podobné vlastnosti/pravidlá správania sa častí</b> objektu/problému/procesu (vzor v obrázku, refrén v hudobnej skladbe, vlastnosti cyklov v prírode, zmena hustoty dopravy v priebehu dňa, zmena premennej v cykle ...)</li> <li>• (VZO3) <b>rozpoznať rovnaké/podobné vlastnosti/pravidlá správania sa v častiach rôznych</b> objektov/problémov/procesov (napr. časť jedného problému je časťou druhého problému)</li> <li>• (VZO4) <b>zovšeobecniť</b> riešenie podobných problémov na celú triedu, zovšeobecniť na základe konkrétnych prípadov (namiesto viacerých podprogramov použijeme jeden s viacerými parametrami),</li> <li>• (VZO5) <b>preniesť/použiť vzory/myšlienky/riešenia</b> z jedného problému na druhý problému (analógia, použitie cyklov, podprogramov, modulov, skrátenie zápisu dát obsahujúcich vzory, použitie stratégie riešenia problému, použitie schémy programu: vstup-spracovanie-výstup, schéma tvorby programu: vytvoriť a inicializovať objekty, definovať ich správania, nastaviť štartovací stav, nechať bežať podľa pravidiel)</li> </ul>
<b>abstrakcia</b> vyber	<ul style="list-style-type: none"> <li>• (ABS1) <b>určiť</b>, ktoré detaily/prvky/vlastnosti/vzťahy objektov/problémov/procesov sú v danej situácii <b>podstatné</b> a ktoré môžeme zanedbať (objem motora &lt;-&gt; farba jeho náteru, prezentovanie informácií &lt;-&gt; nástroj na tvorbu prezentácií)</li> </ul>

podstatné a zanedbaj menej podstatné	<ul style="list-style-type: none"> <li>• (ABS2) <b>z konkrétnych prípadov</b> (inštancií) objektov/problémov/procesov <b>abstrahovať</b> pojmy/vzťahy/postupy, (napr. opakovanie zadaný počet krát, na základe podmienky -&gt; cyklus; desktop, tablet, vstavaný počítač -&gt; počítač; funkcie 3-uholník, štvorec, hexagón -&gt; funkcia n-uholník)</li> <li>• (ABS3) <b>využiť podstatné</b> prvky objektov/problémov/procesov (používať/adaptovať/vytvárať modely, riešiť slovne zadané problémy, vytvárať plány (objektov, prostredí, činností, ...), vyjadriť formou tabuľky, grafu ..., navrhnúť testovacie dáta)</li> <li>• (ABS4) <b>identifikovať vzťahy medzi rôznymi abstrakciami</b> danej entity (napr. funkcia a spôsoby jej zadania: tabuľka, graf, matematická formula; konkrétne texty a čísla -&gt; dátové typy -&gt; objekty; zvýraznenie častí textu -&gt; štýly textu -&gt; generovanie obsahu dokumentu)</li> </ul>
vyhodnotenie rob rozhodnutia	<ul style="list-style-type: none"> <li>• (VYH1) <b>vybrať kritériá</b> pre vyhodnotenie priebehu alebo výsledkov projektu/programu/algoritmu/situácie (napr. rýchlosť vykonania, bezpečnosť systému, náročnosť na zdroje, efektívnosť algoritmu, kvalita zdrojového kódu)</li> <li>• (VYH2) <b>definovať vlastné kritériá</b> pre vyhodnotenie priebehu alebo výsledkov projektu/programu/algoritmu/situácie (napr. pomer ceny a rýchlosti prepravy, pomer ceny a výkonu hardvéru, hodnotiaci funkcia na efektívnosť postupu)</li> <li>• (VYH3) <b>posúdiť kvalitu/správnosť/efektívnosť/vhodnosť</b> objektu/systému/postupu/nástroja na základe <b>vybraných/definovaných kritérií</b> (napr. posúdiť efektívnosti algoritmov, posúdiť bezpečnosť systému, posúdiť správnosť dekompozície, posúdiť presnosť a úplnosť algoritmu/programu/postupu, testovať program/výrobok, posúdiť/dokázať pravdivosť tvrdenia)</li> <li>• (VYH4) <b>posúdiť kritéria</b> pre vyhodnotenie z pohľadu <b>relevantnosti a úplnosti</b> (napr. posúdenie kritérií hodnotenia projektu, ktoré navrhli žiaci; posúdiť testovacie dáta)</li> </ul>

## OBSAHOVÉ ZAMERANIE METODÍK

Vytvorená sada 27 metodík pokrýva tematickú oblasť **Algoritmické riešenie problémov** aktuálneho iŠVP. Okrem špecifických vedomostí a zručností z oblasti programovania sa v metodikách priebežne sa venujeme aj rôznym **stratégiám riešenia problémov** (napr. dekompozícia problému na podproblémy, hľadanie vzoru). Niektoré metodiky zasahujú aj do iných oblastí školskej informatiky či iných školských predmetov. Dbali sme však, aby schopnosť vyriešiť problém z inej oblasti **nebola postavená primárne na vedomostiach z inej oblasti**.

Rovnako sme sa snažili jednotlivé témy spracovať, aby boli v prijateľnej podobe pre žiakov strednej školy a obsahovo blízke rôznym záujmom žiakov.

Sadu metodík sme rozdelili do troch skupín:

- vizualizácia priebehu výpočtu pomocou korytnačej grafiky (modul turtle),
- konzolové aplikácie, riešenie konkrétnych problémov,
- grafické aplikácie (modul tkinter).

V úvodnej prvej metodike sa žiaci stretávajú s pojmom premenná (vo význame pomenovania hodnoty). V tejto časti nie je potrebné vysvetľovať pojem premenná, resp. technickú implementáciu tohto konceptu. Žiaci budú počas nasledujúcich hodín zbierať skúsenosti a postupne si tento koncept osvoja aj na abstraktnej úrovni.

V druhej až šiestej metodike využívame pri riešení problémov korytnačiu grafiku (modul turtle). Samotná korytnačia grafika slúži len na vizualizáciu priebehu výpočtu, na ľahšie pochopenie princípov a ľahšiu lokalizáciu chýb v programoch. Nie je teda cieľom, ale prostriedkom, ktorý navyše premostí programátorské skúsenosti žiakov zo základnej školy.

Ďalších šestnásť metodík je zameraných na konzolové aplikácie riešiace konkrétne problémy zamerané na prácu s textami, číslami, zoznamami dát.

Posledných päť metodík je venovaných tvorbe grafických aplikácií. Zaradili sme ich na koniec celého bloku preto, aby sme sa vyhli vytváraniu na vzhľad síce pekných, ale vnútorne „chudobných“ aplikácií. Grafické rozhranie je sprostredkovateľom medzi kvalitne navrhnutou a implementovanou aplikáciou (riešiacou nejaký problém) a jej používateľom.

Pre lepšiu orientáciu uvádzame zoznam tém, na ktoré sú zamerané jednotlivé metodiky:

**Tabuľka 1 Zoznam tém jednotlivých metodík**

oblasť	názov metodiky
vizualizácia priebehu výpočtu pomocou korytnačej grafiky (modul turtle)	01 Úvod do programovania, výpočty v konzole
	02 Korytnačia grafika
	03 Vlastné funkcie bez parametrov a bez návratovej hodnoty
	04 Cyklus s pevným počtom opakovaní
	05 Opakovanie I. + didaktický test
	06 Vlastné funkcie s parametrami – kresliace úlohy
konzolové aplikácie, riešenie konkrétnych problémov	07 Vlastné funkcie s parametrami a výstupom – výpočtové úlohy
	08 Chyby vo výpočtoch, ich rozpoznávanie a odstraňovanie
	09 Podmienený príkaz
	10 Opakovanie II. + didaktický test
	11 Reťazce
	12 Reťazcové metódy, zložené a vnorené podmienky
	13 Algoritmy s reťazcami
	14 Odchyťávanie výnimiek
	15 Generovanie výnimiek
	16 Opakovanie III. + didaktický test
	17 Zoznamy a metódy zoznamov
	18 Vytváranie a modifikácia zoznamov, použitie náhody
	19 Algoritmy so zoznamami
	20 Cyklus s podmienkou
	21 Vnorené riadiace štruktúry
	22 Opakovanie IV. + didaktický test
grafické aplikácie (modul tkinter)	23 Grafické používateľské rozhranie – tlačidlá, textové polia, popisy
	24 Grafické používateľské rozhranie – plátno
	25 Komplexný projekt – výber problému, analýza a návrh riešenia problému
	26 Komplexný projekt – implementácia riešenia problému
	27 Komplexný projekt – finalizácia projektu + prezentácia a diskusia

## ŠTRUKTÚRA ZBIERKY INOVATÍVNYCH METODÍK A PRÍLOHY

### ŠTRUKTÚRA METODIKY

Metodika je didaktickým materiálom pre učiteľa, ktorý mu umožní čo najkvalitnejšie naučiť žiakov vybranú tému. Využíva metódy aktívneho a bádateľského vyučovania, obsahuje prvky formatívneho hodnotenia aj nástroje na vyhodnotenie výsledkov výučby. Pre úspešnú implementáciu vytvorenej inovatívnej metodiky poskytujeme učiteľom čo najväčšiu podporu pri príprave, priebehu aj vyhodnotení výučby. Obzvlášť u učiteľa, ktorý sa s daným učivom či spôsobom výučby stretáva prvýkrát, je vysoké riziko, že hodina nedopadne podľa jeho a ani nášho očakávania. Preto je dôležité, aby si dôsledne preštudoval metodiku a ďalšie materiály uvedené v jej prílohe a následne ich využil vo svojej výučbe.

V rámci NPITA používame nasledovnú štruktúru metodiky:

**Informačný list** (stručné informácie o metodike, jej metadáta):

- Tematický celok / téma,
- Stupeň školy / Odporúčaný ročník / Rozsah ,
- Požiadavky na vstupné vedomosti a zručností,
- Ciele – žiakom osvojované vedomosti a zručností – štandardy z iŠVP a za nimi špecifické kognitívne ciele, ktoré boli pre výučbu danej témy identifikované),
- Ciele – žiakom rozvíjané spôsobilosti – koncepty a podkoncepty informatického myslenia,
- Riešený didaktický problém – známe miskoncepce žiakov, nesprávne metodické praktiky, zdôvodnenie použitia metodiky,
- Dominantné vyučovacie metódy a formy,
- Príprava učiteľa a pomôcky – pomôcky pre učiteľa a žiakov,
- Diagnostika splnenia vzdelávacích cieľov – diagnostické nástroje, pomocou ktorých učiteľ zistí mieru splnenia vzdelávacích cieľov.

**Úvod:**

- Kontext zasadenia danej hodiny do kontextu predchádzajúcej a nasledujúcej hodiny,

**Priebeh výučby:**

- Osnova výučby (realizovanej, napr. podľa modelu 5E),
- sekvencia gradovaných úloh, ktoré žiaci riešia/skúmajú individuálne, či v skupine využitím rôznych aktivizujúcich metód,
- vyhodnotenie výsledkov výučby.



## PRÍLOHY

Pre učiteľa poskytujeme súbor **zim-python.pdf** so zbierkou metodík (tento dokument) a prílohu **zim-python.zip** obsahujúcu:

- priečink **ucitel**
  - súbor **pracovny\_zosit\_riesene\_ulohy.docx** – pracovný zošit s pracovnými listami, ktoré obsahujú zadania a riešenia úloh pre jednotlivé kapitoly,
  - priečink **pracovne\_subory\_riesenia/** – obsahujúci podpriečinky **01** až **27** obsahujúce pythonovské zdrojové kódy riešení úloh, excelovské tabuľky pre vyhodnotenie výsledkov vyučovania, prípadne prezentácie k výučbe,
  - priečink **testy/** – obsahujúci priečinky **test1**, **test2**, **test3**, **test4** so zadaniami a riešeniami didaktických testov s prípadnými pracovnými súbormi.
- priečink **ziak**
  - súbor **pracovny\_zosit.docx** – pracovný zošit pre žiakov s pracovnými listami, ktoré obsahujú zadania úloh pre jednotlivé kapitoly,
  - priečink **pracovne\_subory/** – obsahujúci podpriečinky **01** až **27** obsahujúce pythonovské zdrojové kódy pracovných súborov prípadne referenčné materiály či inštruktážne listy.

Odporúčame, aby učiteľ na začiatku výučby žiakom poskytol súbor **pracovny\_zosit.docx** spolu s obsahom priečinku **ziak**. Zošit obsahuje zadania všetkých úloh pre výučbu, a tiež ďalšie úlohy na precvičenie učiva. Na konci každej témy je uvedené stručné zhrnutie nových poznatkov – Vedomosti v kocke. Aj keď poskytujeme elektronickú verziu pracovného zošitu, odporúčame používať tlačенú verziu, do ktorej si môžu žiaci robiť svoje poznámky rukou, napr. hlbšie analyzovať zadané problémy. Za každou úlohou v pracovnom liste žiaka, kde sa predpokladá, že žiak niečo vytvorí, je voľné miesto označené ako „Riešenie:“. Tento priestor by mali žiaci využiť na analýzu riešených úloh, ako priestor pre poznámky k riešeniu úlohy, prípadne na uvedenie častí kódov, ktoré žiaci alebo učiteľ považujú za dôležité. Zo spätnej väzby z overenia metodík vyplynulo, že elektronická verzia pracovného listu spomaľuje prácu žiakov (prepínanie okien, pomalšie písanie na klávesnici než na papieri a pod.)

## ODPORÚČANIA K VÝUČBE PROGRAMOVANIA PODĽA METODÍK

Ak vaši žiaci už absolvovali stredoškolský základný kurz programovania alebo ho absolvovali v inom jazyku než Python, neodporúčame realizovať výučbu podľa uvedených metodík. Nami navrhnuté metodiky sledujú ciele definované v iŠVP. Rovnaké alebo veľmi podobné ciele sledoval aj kurz, ktorí žiaci už absolvovali v inom jazyku. Absolvovaním výučby podľa uvedených metodík by žiaci dosiahli rovnaké či veľmi podobné ciele len inými prostriedkami. Odporúčame preto pokračovať v tom, čo ste začali v základnom kurze, rozvíjať vedomosti a zručnosti žiakov viac do hĺbky, zamerať sa na riešenie problémov a pod.

Štandardne jedna metodika pokrýva rozsah jednej vyučovacej hodiny (vzhľadom na jednohodinovú dotáciu informatiky uvedenú v iŠVP). V metodike plánujeme vyučovaciu hodinu ako 40-minútovu (5 minút je rezerva na administratívne záležitosti).

Sme si vedomí toho, že úroveň a schopnosti žiakov prichádzajúcich na SŠ môžu byť značne rozdielne. Podstatnú úlohu tú zohráva úroveň vyučovania informatiky na ZŠ, z ktorej žiak prichádza.

V každej metodike je definované minimum, ktoré by mal byť schopný dosiahnuť každý žiak. Do metodik sme zaradili aj úlohy, zaradenie ktorých ponechávame na učiteľovi. Ak to časová dotácia dovoľuje (žiaci majú dobré základy zo ZŠ alebo je navýšená dotácia hodín informatiky), odporúčame aj tieto úlohy do výučby zaradiť. Zámerne sme sa vyhli označeniu týchto častí „nálepkou“ typu nepovinné, dobrovoľné, náročnejšie úlohy a pod. Takto označený obsah majú žiaci tendenciu ignorovať. Tieto časti sme preto označili neutrálnym „Riešte podľa pokynov učiteľa“.

Jednotlivé metodiky na seba nadväzujú, preto neodporúčame meniť navrhnuté poradie. Odporúčame učiteľom, aby si vopred prezreli aj nasledujúce metodiky (nie len tú, podľa ktorej momentálne vedú výučbu). Učiteľ takto dostane prehľad o celkovej koncepcii výučby podľa tejto zbierky metodík.

Dôraz nekladíme na ovládanie programovacieho jazyka, ale na rozvíjanie kompetencií riešiť problémy. Z prvkov samotného programovacieho jazyka sa snažíme žiakom ukázať len nevyhnutné minimum. Aj napriek tomu, že jazyk umožňuje viaceré prístupy, vždy uprednostňujeme a používame len jeden. Učiteľa však upozorňujeme na viaceré možné prístupy, aby bol pripravený vhodne reagovať na prípadné otázky žiakov.

Jednotlivé koncepty z oblasti programovania zavádzame postupne. Najskôr žiak s konceptom pracuje, zbiera prvé skúsenosti bez toho, aby sme tento koncept formálne definovali. O niekoľko vyučovacích hodín neskôr (v nasledujúcich metodikách), keď žiak nadobudne dostatok skúseností prichádza porozumenie princípom na vyššej, abstraktnej úrovni. Tento prístup postupného „dozrievania“ konceptov programovania v čo najviac relevantných situáciách pokladáme pre žiaka za prirodzenejší a zaujímavejší ako bezduché drilovanie konceptov programovania na nezmyselných školských úlohách s nízkym motivačným potenciálom. Pokiaľ nie je v metodike uvedené inak, neodporúčame učiteľom, aby tento proces urýchlili.

## KONCEPCIA PROGRAMOVANIA V JAZYKU PYTHON

V tejto kapitole uvádzame **konceptné informácie odborného charakteru** k sérii metodík pre základný kurz programovania. Uvedené informácie sa týkajú nielen samotného jazyka Python, ale aj prechodu na programovací jazyk Python vo výučbe. V texte upozorňujeme aj na niektoré rozdielne prístupy k programovaniu, ktoré sa používajú v jazyku Python.

### PROGRAMOVACÍ JAZYK PYTHON

Jazyk Python je moderný a živý programovací jazyk. Programovacím jazykom série metodík je jazyk Python, vo verzii 3.x. POZOR, verzie 2.x a 3.x sú vzájomne nekompatibilné. Odporúčame použiť najnovšiu stabilnú verziu jazyka Python (<https://www.python.org/>). V metodikách používame verziu 3.6, resp. novšiu. V starších verziách jazyka Python budú niektoré časti kódov nefunkčné, resp. ich interpreter vyhlási za syntakticky chybné (napr. formátovanie reťazcov pomocou f-strings).

Jazyk Python má svoje špecifiká, ktoré sme zohľadňovali pri tvorbe metodík a výbere úloh. Dôsledkom je, že mnohé z typicky programátorských úloh školskej informatiky v metodikách nenájdete, pretože ich nemá zmysel v Pythone riešiť. Na druhej strane v metodikách nájdete úlohy, ktoré sa vo výučbe programovania v inom programovacom jazyku nepoužívali.

Napr. úlohy na algoritmy usporadúvania sme vynechali. Zoznamy majú zabudovanú metódu na usporadúvanie a je zbytočné, aby žiaci programovali jej menej efektívnu implementáciu. POZOR! Týmto nehovoríme, že v Pythone nie je potrebné sa zaoberať algoritmami usporadúvania. Hovoríme len to, že pre žiakov v základnom kurze to nie je potrebné.

Na druhej strane môžeme zaradiť prvky, ktoré sa bežne v základnom kurze neobjavujú. Napr. prácu s výnimkami. Namiesto zdĺhavého testovania, či akciu možno realizovať, je v Pythone jednoduchšie odchytať výnimky po realizácii akcie. Podobne, namiesto „tichého“ predpokladania vhodných vstupov, v prípade nevhodného vstupu by program mal vyhodíť relevantnú výnimku.

Vo všeobecnosti preto neodporúčame prechod na programovací jazyk Python vo vyučovaní realizovať spôsobom, že riešenia úloh, ktoré sme doteraz používali, prepíšeme do jazyka Python. Python to síce umožňuje, ale nevyužijeme jeho silu. Prechod na jazyk Python je pri tomto prístupe ťažké rozumne zdôvodniť.

Uvedená séria metodík je nezávislá na konkrétnom editore kódu, resp. vývojovom prostredí. V ďalšej časti textu uvádzame niekoľko možností aké vývojové nástroje použiť, výhody a nevýhody jednotlivých nástrojov. Odporúčame, aby si učiteľ vybral jedno z vývojových prostredí, toto žiakom predstavil a následne ho so žiakmi používal. Neodporúčame učiť žiakov používať viaceré vývojové prostredia, pretože zvládnutie každého z nich si vyžaduje nejaký čas, ktorý je vhodnejšie venovať samotnému programovaniu, resp. riešeniu konkrétnych problémov využitím programovania.

Licenčné podmienky vo všetkých prípadoch umožňujú použitie vo výučbe aj v domácej príprave žiaka.

## POŽIADAVKY NA SOFTVÉR, VÝVOJOVÉ PROSTREDIE

Uvádzame stručný prehľad vývojových prostredí. Nepopisujeme konkrétne programy, resp. verzie programov. Ak si zvolíte konkrétne prostredie, môžete nájsť odlišnosti s tým, čo je uvedené nižšie. Pri každej z možnosti uvádzame odkazy na niektoré stránky s príslušným softvérom. Tieto zoznamy si nekladú za cieľ byť kompletne.

Aj keď séria metodík nie je závislá na konkrétnom vývojovom prostredí odporúčame prvú možnosť (Lokálna inštalácia jazyka Python a lokálna inštalácia vývojového prostredia) a niektoré z komplexnejších vývojových prostredí. Na základe našich skúseností odporúčame lokálnu inštaláciu najnovšej stabilnej verzie jazyka Python a lokálne inštalované vývojové prostredie PyCharm Edu (<https://www.jetbrains.com/pycharm-edu/>).

Odporúčame, aby mali žiaci na prvej hodine inštalované všetky potrebné aplikácie a aby boli aplikácie pripravené na prácu.

Ak sa rozhodnete pre lokálnu inštaláciu vývojového prostredia (napr. PyCharm Edu) vytvorte žiakom projekt na testovanie. PyCharm ho pri nasledujúcom spustení automaticky použije a nebude vyžadovať jeho vytvorenie od žiakov. Odporúčame, aby žiaci vytvárali a pracovali s projektom (teda nie otvárali konkrétny \*.py súbor) a všetky súbory udržiavali v projektoch. Projekt je v podstate priečinok so súbormi, s ktorými žiak pracuje a so súbormi, v ktorých si vývojové prostredie uchováva nastavenie a vlastnosti projektu.

Na prvej hodine žiaci nebudú vytvárať žiadne súbory, využijú len konzolu jazyka. Aby si žiaci mohli aj doma precvičiť preberané učivo, je potrebné, aby si na domácich počítačoch inštalovali rovnaké aplikácie ako v škole. Využiť môžu návod na inštaláciu uvedený v súbore `01\PyCharm_instalacia.pdf`.

## LOKÁLNA INŠTALÁCIA JAZYKA PYTHON A LOKÁLNA INŠTALÁCIA VÝVOJOVÉHO PROSTREDIA

- Výhody:
- žiak pracuje na lokálnom počítači a má k dispozícii silné nástroje a podporu vývojového prostredia,
  - editor vývojového prostredia ponúka možnosť dopĺňania zdrojového kódu (auto-completion), vie prepájať zdrojový kód z viacerých súborov (správa projektov, modulov), čísluje riadky zdrojového kódu (dôležité pre lokalizáciu chyby pri hlásení interpretera),
  - vývojové prostredie obsahuje ladiace nástroje,
  - v prostredí je prístupná konzola, ktorú môže žiak využiť na experimentovanie a objavovanie,
  - súbory so zdrojovými kódmi (projekty) je jednoduché kopírovať (ako priečinky so súbormi), napr. medzi školským a domácim počítačom,
  - editor dokáže už počas samotného písania kódu upozorniť na syntaktické chyby a niektoré možné behové chyby.

- Nevýhody:
- je potrebná lokálna inštalácia (jazyk, prostredie) na každom počítači v učebni, prípadne aj na domácom počítači žiaka,
  - vývojové prostredie môže byť pre začiatočníka, najmä zo začiatku neprehľadné a komplikované (niektorí producenti poskytujú aj odľahčenú verziu profesionálnych vývojových prostredí, napr. Pycharm Edu),
  - ak žiak pracuje na viacerých počítačoch (doma, v škole) súbory so zdrojovými kódmi je potrebné kopírovať medzi počítačmi alebo využiť nejaký cloudový systém.
- Softvér:
- interpret jazyka Python:
    - Python – <https://www.python.org/downloads/>,
  - vývojové prostredie:
    - Pycharm Edu – <https://www.jetbrains.com/pycharm-edu/>,
    - Visual Studio Code (<https://code.visualstudio.com/>) + VS Code Python extension (<https://marketplace.visualstudio.com/items?itemName=ms-python.python>),
    - Thony – <https://thonny.org/> (interpret jazyka Python je súčasťou prostredia).

## LOKÁLNA INŠTALÁCIA JAZYKA PYTHON

- Výhody:
- žiak pracuje na lokálnom počítači a má k dispozícii nástroje jednoduchého vývojového prostredia,
  - editor ponúka možnosť dopĺňania zdrojového kódu (auto-completion), ale len pre štandardné knižnice,
  - vývojové prostredie obsahuje jednoduché ladiace nástroje,
  - súčasťou (v samostatnom okne) je konzola, ktorú môže žiak využiť na experimentovanie a objavovanie,
  - súbory so zdrojovými kódmi je jednoduché kopírovať (ako súbory, nie ako projekty),
  - prostredie je pomerne jednoduché, obsahuje len základné nástroje.
- Nevýhody:
- je potrebná lokálna inštalácia (jazyk) na každom počítači v učebni, prípadne aj na domácom počítači žiaka,
  - ak žiak pracuje na viacerých počítačoch (doma, v škole) súbory so zdrojovými kódmi je potrebné kopírovať medzi počítačmi,
  - prostredie neprepája zdrojové kódy z viacerých súborov (správa projektov, modulov), neumožňuje číslovanie riadkov v zdrojovom kóde (Python < 3.8).
- Softvér
- interpret jazyka Python:
- Python – <https://www.python.org/downloads/>

## ONLINE PROSTREDIE – CLOUDOVÉ SLUŽBY

- Výhody:
- nie je potrebná lokálna inštalácia,
  - súbory so zdrojovými kódmi je možné zdieľať alebo publikovať,
  - prostredie je pomerne jednoduché, obsahuje len základné nástroje,
  - súbory so zdrojovými kódmi sú prístupné z každého počítača v sieti (po prihlásení).

- Nevýhody:
- počítač, na ktorom žiak pracuje, musí byť pripojený do siete internet,
  - žiak pracuje na vzdialenom počítači a k dispozícii má len jednoduchý editor (nie vývojové prostredie),
  - ladiace nástroje vo väčšine prípadov nie sú k dispozícii,
  - konzola nie je prístupná, POZOR! s konzolou budeme pomerne často pracovať,
  - niektoré prostredia neumožňujú použitie modulu turtle, POZOR! s modulom turtle budeme priamo pracovať v nasledujúcich metodikách.

Softvér: vývojové prostredie:

- <https://repl.it/>
- <https://trinket.io/>

## ŠPECIÁLNE NÁSTROJE

Jazyk Python má niektoré prvky, ktoré môžu byť pre človeka zvyknutého na nejaký iný programovací jazyk mierne mätúce. Nie sú to len ojedinelé prípady, na ktoré môžeme výnimočne naraziť, ale dosť často sa tieto odlišnosti cielene využívajú ako výhoda jazyka. Predpokladáme, že pri vlastnom štúdiu rôznych zdrojov alebo v otázkach žiakov na ne môžete naraziť. Aby sme vám uľahčili prácu, tak niektoré z nich aj s vysvetlením uvádzame v nasledujúcej časti. Pokiaľ sa z daným prvkom počas výučby nestretnete, nemusíte ho žiakom vysvetľovať. V opačnom prípade zvážte, či vysvetlenie nepodať len konkrétnemu žiakovi namiesto celej skupiny.

Aj keď vývojové prostredia umožňujú krokovanie programov, niekedy pre lepšie pochopenie pomôže vizualizácia priebehu výpočtu. Pre tento prípad odporúčame online prostredie <http://pythontutor.com/>, ktoré môže učiteľ využiť a demonštrovať žiakom priebeh realizácie programu (pozri obrázok nižšie).



Obrázok 1 Vizualizácia priebehu vykonávania programu v prostredí pythontutor.com

Prostredie umožňuje vytvoriť trvalý odkaz na program a neskôr ho sprostredkovať žiakom na experimentovanie s programom.

**Poznámka:**

Pri premietaní cez dataprojektor na plátno sa môže ukázať, že vopred nastavená veľkosť písma je nedostatočná a žiaci v zadných laviciach už text nevidia prečítať.

V prípade použitia vývojového prostredia, ktoré je súčasťou inštalácie jazyka Python, nastavíme požadovanú veľkosť písma v menu „Option | Configure IDLE | Fonts/Tabs“.

V prípade prostredia PyCharm Edu nastavíme požadovanú veľkosť písma v menu „File | Settings... | Editor | Font“.

V prípade online prostredí využijeme možnosť zväčšenia obsahu stránky (zoom) pomocou kolieska myši a klávesu Ctrl.

## PRAVIDLÁ PRE PÍSANIE ZDROJOVÉHO KÓDU

Jazyk Python je dosť benevolentný k tomu, ako písať zdrojové kódy programov (POZOR, nemýľme si to so syntaxou, ktorá je presne definovaná). Programátori si postupom času vytvorili pravidlá ako zdrojový kód písať. Ak ich nebudeme dodržiavať, naše programy síce budú fungovať správne, ale budú pre iných programátorov ťažšie čitateľné a ťažšie pochopiteľné. Preto sa aj my v našich metodikách snažíme tieto pravidlá rešpektovať. Spomínané pravidlá sú zverejnené ako súčasť Python Enhancement Proposals, skrátene PEPs (<https://www.python.org/dev/peps/>). Pravidlá pre písanie zdrojového kódu programov nájdeme pod číslom 8: PEP 8 -- Style Guide for Python Code (<https://www.python.org/dev/peps/pep-0008/>)<sup>1</sup>. Z množstva odporúčaní vyberáme:

- Pre odsadenie zdrojového kódu programu používajme 4 medzery (nie tabulátor). Niektoré editory sú nastavené tak, že po stlačení tabulátora doplnia automaticky 4 medzery.
- Obmedzme dĺžku riadkov na 79 znakov. Pri dokumentačných reťazcoch a komentároch na 72 znakov.
- Pri vytváraní **identifikátorov** im **dajme význam** (=popisný názov) a nepoužívajme len jednopísmenkové pomenovanie.
- Pre názvy premenných a funkcií používajme **malé\_znaky\_s\_podčiarkovníkom**.
- Pri názvoch tried používajme **ĎaviePísmo**.
- Pre názvy konštánt používajme **VEĽKÉ\_PÍSMENÁ**. Technicky Python konštantu nepozná. Je to premenná, ktorej hodnotu nemeníme. Ide teda len o zaužívanú konvenciu.
- Pri názvoch modulov používajme malé písmena a pokiaľ možno len jedno slovo.
- Import modulov umiestňujme na začiatok súboru.
- Definície funkcií umiestňujme spolu pri sebe.
- Používajme dokumentačné reťazce (docstrings) na dokumentovanie funkcií a modulov.
- Používajme dva prázdne riadky na oddelenie definícií funkcií.

---

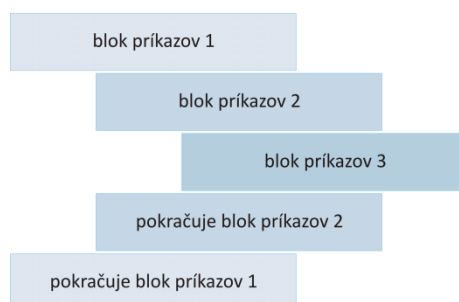
<sup>1</sup> Online kontrolu kódu či spĺňa odporúčanie PEP8 môžeme realizovať aj online na <http://pep8online.com/>.

- Príkazy najvyššej úrovne vrátane volaní funkcií umiestňujeme v spodnej časti programu.
- Do každého riadku píšme len jeden príkaz. Program bude čitateľnejší a ak v danom riadku bude chyba, je len jeden príkaz, ktorý je potrebné skontrolovať.
- Nepoužívame globálne premenné vo funkciách (výnimkou je napr. konštanta na úrovni modulu). Toto pravidlo čiastočne porušíme:
  - Pri kreslení pomocou korytnačky, kde funkcie používajú grafické pero (korytnačku) vytvorené v hlavnom programe.
  - Pri tvorbe grafického rozhrania (tkinter). Štandardne by sa v tomto prípade využil objektový prístup a inštančné premenné. Keďže objektové programovanie nie je súčasťou iŠVP využijeme globálne premenné (premenné prepojené s prvkami grafického rozhrania).
- Nevytvárajme funkcie s viac ako 15 riadkami (15 riadkov sa ešte vojde na jednu obrazovku a 15 riadkovú funkciu dokážeme ešte vnímať ako celok). Dlhšie funkcie môžu svedčiť o nedostatočnej dekompozícii problému a je pomerne ťažké takéto rozsiahle funkcie testovať.

Tieto pravidlá sme uviedli preto, aby sme vysvetlili naše odporúčania ohľadom písania zdrojového kódu programu. Na aplikáciu jednotlivých pravidiel vás upozorníme v metodike vtedy, keď sa daná situácia vyskytne a budeme predpokladať, že ju sprostredkujete žiakom. Informácie z tejto časti sprostredkujte žiakom priebežne tak, ako to bude vyžadovať situácia. Dobrým argumentom prečo pri písaní programov dodržiavať tu uvedené odporúčanie je fakt, že program píšme len raz, ale čítame ho „tisíckrát“. Moderné vývojové prostredia často obsahujú funkciu typu „Reformat code“, ktorú sa oplatí používať.

## ODSADZOVANIE KÓDU

Python nepoužíva žiadne obalovacie znaky (napr. zátvorky `{ }`) alebo kľúčové slová (napr. `begin`, `end`) na určenie bloku programu (telo cyklu, telo funkcie a pod). Blok je definovaný odsadením (pozri obrázok nižšie).



**Obrázok 2 Hierarchia blokov programu určená odsadením**

Pre odsadenie (=určenie bloku) stačí 1 medzera. Nech sa rozhodneme akokoľvek, musíme to v celom programe jednotne dodržiavať. Pre odsadenie sa odporúčajú 4 medzery. Toto odporúčanie



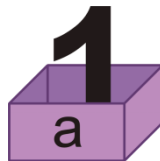
v metodikách dodržiavame (viď. ukážka). Všimnime si, že riadok pred začiatkom bloku končí znakom dvojbodka.

```
def oslovenie(vek):  
    if vek < 18:  
        vysledok = 'chlapec/dievča'  
    else:  
        vysledok = 'pán/pani'  
    return vysledok
```

## PREMENNÉ

Python pracuje s premennými inak ako ostatné jazyky, napr. jazyk Pascal. Keď v jazyku Pascal priradíme do premennej hodnotu, alokuje sa v pamäti blok, v ktorom je daná hodnota uložená. Môžeme si to predstaviť ako škatuľku s rovnakým menom ako má premenná.

```
var a: integer = 1;
```



V Pythone chápeme premenné skôr ako visačky, ktoré pomenúvajú hodnoty (nie škatuľky s hodnotami). Ak priradíme do premennej hodnotu, Python si pre túto hodnotu vytvorí pomenovanie.

```
a = 1
```



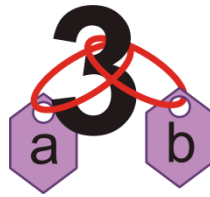
Ak zmeníme hodnotu premennej, Python si týmto menom pomenuje inú hodnotu.

```
a = 3
```



Pôvodná hodnota môže v pamäti zostať (môže byť referencovaná aj z inej premennej), ale už sa k nej nevieme dostať pomocou pôvodného pomenovania `a`. Ak priradíme rovnakú hodnotu do inej premennej, Python k tejto hodnote priradí ďalšie meno.

```
b = 3
```



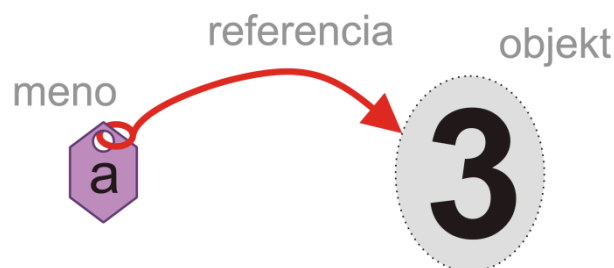
Voľne preložené z:

<http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html#other-languages-have-variables>.

V tejto sérii metodík chápeme **premenné vo význame pomenovania hodnoty**. Technicky vzaté, všetky pythonovské premenné obsahujú referencie na objekty (v Pythone je všetko objektom).

Ak Python vykoná príkaz `a = 3`, stane sa nasledovné:

1. vytvorí sa objekt reprezentujúci hodnotu 3, (alebo sa tento objekt nájde v pamäti, ak už existuje)
2. vytvorí sa meno `a` (ak meno už existuje, zruší sa jeho referencia na pôvodný objekt),
3. zvýši sa počet referencií na nový (alebo nájdený) objekt o 1,
4. vytvorí sa referencia mena `a` na novovytvorený objekt.



Analógia s krabičkou môže priniesť viacero problematických interpretácií, preto odporúčame už od začiatku pracovať s premennou v zmysle pomenovania hodnoty.

## ZMENA HODNOTY PREMENNEJ

Ak chceme zmeniť hodnotu, na ktorú odkazuje nejaké meno, môžeme to spraviť nasledovne:

```
x = povodna_hodnota
x = nova_hodnota
```

V prípadoch, keď potrebujeme pôvodnú hodnotu nejakou aktualizovať (napr. zvýšiť hodnotu o 1) použijeme spôsob:

```
x = x + 1
```

Tento príkaz sa dá zapísať skráteno a mnoho zdrojov aj používa skrátený zápis v tvare<sup>2</sup>:

```
x += 1
```

<sup>2</sup> Neodporúčame tento skrátený zápis zavádzať štandardne pre všetkých žiakov. Je náročnejší na pochopenie a niektorí žiaci s ním môžu mať problém.

Vo väčšine prípadov sú tieto zápisy ekvivalentné. Ale nie vždy je tomu tak. Porovnajme napr. aktualizáciu zoznamu:

```
a = [1]
b = a
print(a)      #[1]
print(b)      #[1]
a = a + [2]
print(a)      #[1, 2]
print(b)      #[1]
```

```
a = [1]
b = a
print(a)      #[1]
print(b)      #[1]
a += [2]
print(a)      #[1, 2]
print(b)      #[1, 2]
```

V prípade, že je možná zmena hodnoty objektu na mieste (v tomto prípade zoznamu), Python v prípade skráteneho zápisu (`+=`) uprednostní zmenu hodnoty pôvodného objektu.

Ak zmena hodnoty objektu možná nie je alebo sa použije neskrátený zápis, Python vytvorí nový objekt s novou hodnotou a pôvodné meno previaže s novým objektom. Takéto správanie nás môže prekvapiť, ak túto zmenu realizujeme vo funkcii. Porovnajme:

```
def pridaj(a):
    a = a + [2]

zoznam = [1]
print(zoznam)  #[1]
pridaj(zoznam)
print(zoznam)  #[1]
```

```
def pridaj(a):
    a += [2]

zoznam = [1]
print(zoznam)  #[1]
pridaj(zoznam)
print(zoznam)  #[1, 2]
```

Premenná je referenciou na objekt reprezentujúci nejakú hodnotu. Táto filozofia sa uplatňuje aj pri funkciách a pri ich volaní. Pri odovzdávaní parametrov funkcii sa funkcii odovzdá referencia na objekt. V jazykoch ako Pascal sme uvažovali parametre volané hodnotou alebo parametre volané odkazom. V Pythone sú parametre volané referenciou na objekt. Akákoľvek zmena objektu (ak je objekt meniteľného typu, viď. nasledujúca kapitola) vo funkcii sa prejaví aj v hlavnom programe.

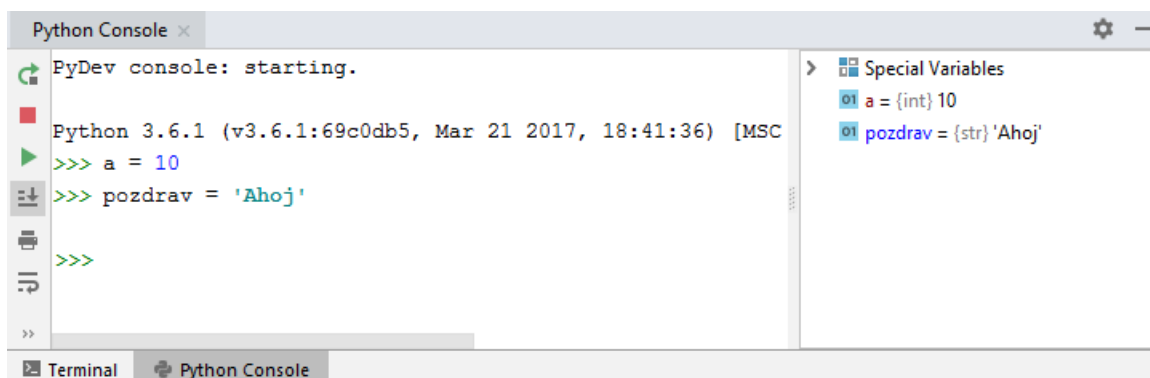
## MENITEĽNÉ A NEMENITEĽNÉ TYPY

V Pythone existujú dva druhy typov (presnejšie objektov): meniteľné a nemeniteľné. Uvádzame štandardné typy a tučným sú zvýraznené tie, s ktorými sa v našich metodikách môžete stretnúť:

- meniteľné (mutable) – **zoznam**, množina, slovník
  - obsah premennej je možné meniť, napr. do zoznamu je možné pridať hodnotu, odstrániť hodnotu, zmeniť poradie hodnôt a pod.
  - ak funkcii pošleme v parametroch referenciu na takúto hodnotu a funkcia túto hodnotu modifikuje, zmena sa prejaví aj v hlavnom programe,
- nemeniteľné (immutable) – **logický typ**, **celé číslo**, **reálne číslo**, **n-tica**, **reťazec**,
  - obsah premennej nie je možné zmeniť, výsledkom nejakej operácie je teda nový objekt s novou hodnotou.

## PREMENNÁ V KONZOLE

Pri experimentovaní v konzole vo vývojovom prostredí PyCharm je možné neustále sledovať hodnoty premenných – v časti konzolového okna. Odporúčame na tento fakt žiakov upozorniť. Žiaci tak získajú lepšiu predstavu o tom, čo sa s premennou počas manipulácie s ňou deje.



## FORMÁTOVANIE REŤAZCOV

Python používa niekoľko spôsobov na formátovanie reťazcov (vytváranie reťazcov pre výstup alebo ďalšie spracovanie, ktoré sa vyskladajú z konštantných hodnôt, hodnôt premenných a návratových hodnôt funkcií). Pri štúdiu iných zdrojov na ne môžete naraziť.

Predpokladajme, že potrebujeme vypísať informáciu typu: „Priemer známok žiaka Mrkvička je: 1.7.“. Ak meno žiaka (Mrkvička) a jeho priemer (1,672) máme uložený v premenných `meno` a `priemer`, môžeme to spraviť nasledovnými spôsobmi:

```
print('Priemer známok žiaka ' + meno + ' je: ' + str(round(priemer,1)) + '.')
print('Priemer známok žiaka %s je: %3.1f.' % (meno, priemer))
print('Priemer známok žiaka {} je: {:.31f}.'.format(meno, priemer))
print(f'Priemer známok žiaka {meno} je: {priemer:{3}.{2}}.')
#alebo
print(f'Priemer známok žiaka {meno} je: {round(priemer, 1)}.'
```

...

Priemer známok žiaka Mrkvička je: 1.7.  
Priemer známok žiaka Mrkvička je: 1.7.  
Priemer známok žiaka Mrkvička je: 1.7.  
Priemer známok žiaka Mrkvička je: 1.7.  
Priemer známok žiaka Mrkvička je: 1.7.

Posledná z možností (posledné dva výpisy) „f-strings“ sa objavila vo verzii Pythonu 3.6. Tento zápis je vo všeobecnosti najkratší a najprehľadnejší. Preto tento spôsob formátovania reťazcov používame aj v našich metodikách.

Reťazce môžeme uzatvoriť do dvojice apostrofov ('retazec') alebo úvodzoviek ("retazec"). Technicky v tom nie je žiadny rozdiel. V slovenčine skôr natrafíme na texty obsahujúce úvodzovky než apostrofy. Aby sme sa vyhli problematike, keď potrebujeme pracovať s textom obsahujúcim úvodzovky, v metodikách texty uzatvárame do apostrofov. Navyše, pri programovaní sa častejšie využíva anglická klávesnica a na nej sa apostrof napíše stlačením jedného klávesu.

---

## VÝPIS DO KONZOLY – PRINT

V Pythone vo verzii 2.x je `print` výrazom. Vo verzii 3.x bol nahradený funkciou. Rozdiel v použití je nasledovný:

Python 2.x

```
print 'Výsledok: ', 2+2
```

Python 3.x

```
print('Výsledok: ', 2+2)
```

Hodnoty, ktoré chceme vypísať do konzoly, sa vo verzii 3.x uvádzajú v zátvorkách. V metodikách používame verziu Python 3.x. Túto poznámku uvádzame len z dôvodu, že je pomerne veľa online zdrojov, ktoré používajú verziu Python 2.x, na ktoré môžete vy alebo vaši žiaci pri štúdiu naraziť.

---

## IMPORT MODULOV

Interpreter jazyka Python používa štandardné zabudované funkcie (<https://docs.python.org/3/library/functions.html>). Ak však potrebujeme použiť matematické funkcie, funkcie na vykresľovanie obrázkov alebo nejakú ďalšiu funkcionálnosť, tie nie sú štandardnou súčasťou Pythonu, ale sú uložené v moduloch (knižniciach). V tomto kurze programovania v Pythone budeme používať funkcie a konštanty z viacerých modulov (`math`, `turtle`, `random`, `time`, `tkinter`, ...). Pri importe modulu uprednostňujeme zápis<sup>3</sup>: `import <meno modulu>`, napr. `import turtle` alebo `import math`. K jednotlivým metódam, vlastnostiam či konštantám pristupujeme cez **bodkovú notáciu**, napr. `turtle.left(90)`, `turtle.position()`, `math.sqrt(16)`, `math.pi`.

Existuje aj iná možnosť importovania modulu `from <meno modulu> import *`, napr. `from turtle import *`. Týmto spôsobom importu sa skrátí zápis programu, napr. `left(90)`. Pri použití viacerých modulov sa však zhorší prehľadnosť zápisu programu, lebo nebude jasné, ktorému modulu prislúchajú uvedené funkcie. Tento spôsob importu neodporúčame.

Ak chceme zistiť zoznam funkcií, tried a konštant daného modulu, v konzole po importovaní modulu zapíšeme `dir(<meno modulu>)`, napr. `dir(turtle)`.

Ak chceme zistiť viac informácií o vybranej funkcii importovaného modulu, v konzole po importovaní modulu zapíšeme `help(<meno funkcie>)`, napr. `help(turtle.position)` (všimnime si,

---

<sup>3</sup> Existuje aj ďalšie spôsoby použitia príkazu `import`. V metodikách základného kurzu ich však nepoužívame.

že aj keď je `position()` funkciou, pri vyvolaní nápovede zátvorky neuvádzame). V prostredí pokročilejšieho vývojového prostredia je zobrazenie nápovedy k funkcii štandardnou funkcionalitou prostredia (napr. klávesová skratka `Ctrl+q` v PyCharmEdu). Python v tomto prípade využíva dokumentačné reťazce (viď text ďalej), ktoré do príslušného modulu alebo funkcie zaradil jej autor.

Zoznam užitočných modulov nájdeme napr. na <https://docs.python.org/3/library/index.html>, alebo na <https://pypi.python.org/pypi>.

## SLUČKA UDALOSTÍ – MAINLOOP()

Pri grafických programoch využívajúcich modul `turtle` či `tkinter` sa pre stále zobrazenie okna, zachytávanie a spracovávanie udalostí (napr. klávesnice, myši) a občerstvenie obsahu okna používa slučka udalostí reprezentovaná metódou `mainloop()`. Keďže predstavuje nekonečnú slučku, uvádza sa ako posledný príkaz grafického programu.

Niektoré zdroje (manuály, tutoriály) využívajúce modul `turtle` alebo `tkinter` v prostredí IDLE túto slučku nepoužívajú. Keďže prostredie IDLE je samotné vytvorené pomocou `tkinter`, volá slučku udalostí samo. Dôsledkom je, že naše programy aj bez volania tejto slučky budú fungovať správne – okno s výsledkom zostane zobrazené. Mimo IDLE tieto programy síce zbehnú správne, ale ihneď po ukončení vykresľovania sa grafické okno s výsledkom uzatvorí.

Program ako taký by mal byť nezávislý od prostredia, v ktorom sa spúšťa. V metodikách preto vždy v grafických programoch uvádzame aj slučku udalostí `mainloop()`.

## FUNKCIE A VIAC NÁVRATOVÝCH HODNÔT

V niektorých zdrojoch môžete naraziť na situáciu, keď funkcia vracia viac návratových hodnôt. V skutočnosti z hodnôt oddelených čiarkou Python automaticky vytvorí n-ticu. Porovnajme nasledovné funkcie. Napriek tomu, že v časti `return` sú uvedené viaceré hodnoty, obidve funkcie vracajú rovnaký výsledok, jednu hodnotu typu n-tica. Keďže do obsahu základného kurzu nie sú n-tice zaradené, túto konštrukciu v metodikách nepoužívame.

```
def funkcia1():
    return 1, 2, 3

def funkcia2():
    return (1, 2, 3)

print(funkcia1())      # (1, 2, 3)
print(type(funkcia1())) # <class 'tuple'>

print(funkcia2())      # (1, 2, 3)
print(type(funkcia2())) # <class 'tuple'>
```

## ANONYMNÁ PREMENNÁ V CYKLE

Ak v tele cyklu nepoužijeme premennú, môžeme v hlavičke cyklu uviesť namiesto riadiacej premennej označenej písmenami aj znak `_`, ktorý zastupuje tzv. anonymnú premennú.

```
for _ in range(5):  
    print('Ahoj')
```

Ak ju však použijeme v tele cyklu, tak sa správa ako premenná. Zápis kódu:

```
for _ in range(5):  
    print(_)
```

je ekvivalentný so zápisom kódu:

```
for i in range(5):  
    print(i)
```

Poznámka: Znak `_` ma význam aj v konzole, resp. v IDLE, kde zastupuje poslednú vypočítanú hodnotu.

```
>>> 2 + 5  
7  
>>> _ + 3  
10  
>>> _  
10
```

Túto informáciu uvádzame len pre učiteľa. Žiakov ňou zaťažovať nemusíme a ani v metodikách túto funkcionality nepoužívame.

## BEHOVÉ CHYBY A OŠETROVANIE VÝNIMIEK

Python pristupuje k behovým chybám trochu inak ako iné jazyky. Zatiaľ čo niektoré jazyky (alebo zaužívané prístupy) uprednostňujú postupné testovanie či s danými hodnotami možno akciu vykonať, v Pythone akciu vykonáme a potom zistíme, či akcia prebehla bez problémov.

Aj z tohto dôvodu sme do metodík zaradili problematiku výnimiek, ich odchyťovanie a generovanie.

Na začiatku tohto dokumentu sme uviedli, že dôraz kladieme na riešenie problémov. Sú situácie, kedy problém nie je možné vyriešiť. Problém môže nastať pre konkrétne hodnoty (napr.: výpočet obsahu štvorca pre zadanú zápornú dĺžku strany, pozri príklad v časti Dokumentačné reťazce). Človek vtedy zareaguje tak, že vyhlási, že problém sa nedá vyriešiť. Analogicky by mal zareagovať aj program, ktorý rieši rovnaký problém. Z tohto dôvodu sme zaradili aj problematiku generovania výnimiek. Uvažovať o výnimkách ako o špeciálnych stavoch, do ktorých sa môže program počas realizácie výpočtu dostať, vyžaduje analytické myslenie a uvažovanie o všetkých prípadoch nielen tých „očakávaných“.

Týmto rozhodnutím naplňame ciele iŠVP, podľa ktorého by mal žiak napr.:

- rozpoznávať, kedy program pracuje nesprávne,
- hľadať chybu v programe a opraviť ju,
- zisťovať, v ktorých prípadoch program pracuje nesprávne.

## VYTVORENIE ZOZNAMU S VOPRED NASTAVENÝMI HODNOTAMI

Ak potrebujeme v Pythone vytvoriť zoznam a naplniť ho nejakou preddefinovanou hodnotou, spravíme to nasledovne:

```
zoznam = [0] * 5  
  
print(zoznam) # [0, 0, 0, 0, 0]
```

Všimnime si, že ide o podobný koncept ako používame pri reťazcoch.

```
pozdrav = 'Ahoj ' * 5  
  
print(pozdrav) # Ahoj Ahoj Ahoj Ahoj Ahoj
```

Pozor, ale na prípad, ak by touto hodnotou mala byť hodnota meniteľného typu. Vytvorme zoznam zoznamov nasledovne:

```
zoznam = [[]] * 5  
print(zoznam) # [[], [], [], [], []]  
  
zoznam[1].append(3)  
print(zoznam) # [[3], [3], [3], [3], [3]]
```

Python vytvorí prázdny zoznam a referenciu naň opakovane vkladá do výsledného zoznamu. Zo všetkých pozícií je teda referencovaný jeden a ten istý objekt. V tomto prípade použime na vytvorenie radšej generátorovú notáciu (pozri aj kapitolu Generátorová notácia zoznamu).

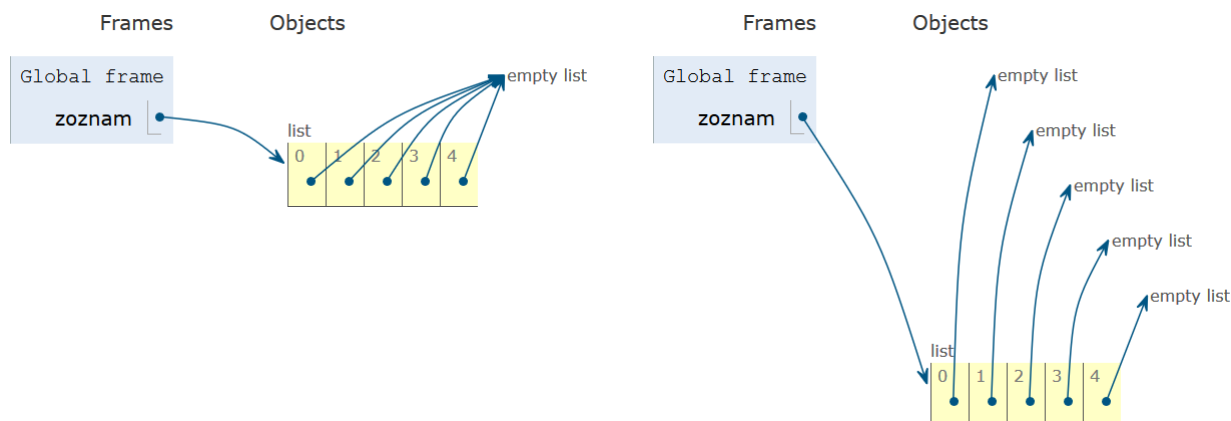
```
zoznam = [[] for i in range(5)]  
print(zoznam) # [[], [], [], [], []]  
  
zoznam[1].append(3)  
print(zoznam) # [[], [3], [], [], []]
```

Vizualizujme si uvedené prístupy prostredníctvom <http://www.pythontutor.com/>.

```
zoznam = [[]] * 5
```

```
zoznam = [[] for i in range(5)]
```





Vnorené dátové štruktúry v metodikách nepoužívame. Uvádzame to však preto, aby sme od začiatku učili žiakov správne postupy a nemuseli ich neskôr opravovať (napr. v nejakom pokračovaní základného kurzu programovania).

## GENERÁTOROVÁ NOTÁCIA (ZOZNAMU)

V prípadoch, keď potrebujeme vytvoriť zoznam s nejakými hodnotami alebo filtrovať dáta v zozname, využijeme namiesto cyklu `for` efektívnejšie riešenie pomocou generátorovej notácie zoznamu. Napriek náročne znejúcemu názvu ide o jednoduchý koncept.

```
cisla = [cislo for cislo in range(1, 11)]
print(cisla)      # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

mocniny = [cislo ** 2 for cislo in range(1, 11)]
print(mocniny)    # [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

neparne_mocniny = [mocnina for mocnina in mocniny if mocnina % 2 != 0]
print(neparne_mocniny) # [1, 9, 25, 49, 81]
```

Tento prístup využívame aj v metodikách a odporúčame ho používať aj vo výučbe.

Analogicky môžeme vytvárať aj slovníky alebo množiny (n-tice len konštrukciou typu `tuple(i for i in range(5))`). Tieto dátové štruktúry, ale v metodikách nespomíname.

## DEFINÍCIA A POUŽITIE FUNKCIÍ

Pri definícii funkcií uprednostňujeme vytváranie funkcií s návratovou hodnotu pred funkciami, ktoré výsledok vypíšu. Dôvodom je univerzálnosť a znovu použiteľnosť riešenia. Všimnime si dôsledok nášho odporúčania v nasledujúcom kóde.

```
def ciferny_sucet(cislo):
    sucet = 0
    while cislo > 0:
        sucet = sucet + cislo % 10
        cislo = cislo // 10
    return sucet
```

```
def stastne_cislo(datum_narodenia):  
    vysledok = datum_narodenia  
    while vysledok > 10:  
        vysledok = ciferny_sucet(vysledok)  
    return vysledok  
  
print(stastne_cislo(23061912)) #stastne cisla Alana Turinga
```

Ak by funkcia `ciferny_sucet()` výsledok vypísala, už by sme ju nevedeli využiť vo funkcii `stastne_cislo()`.

Tento princíp nemožno uplatňovať absolútne. Výnimkou sú napríklad funkcie, ktorých úlohou je primárne niečo vypísať, vykresliť, zapísať do súboru a pod.

Ďalším pravidlom, ktoré v metodikách dodržiavame je, že všetky hodnoty, s ktorými funkcia manipuluje dostane v parametroch. Funkcia tak nepristupuje ku globálnym premenným a nie je závislá na kontexte, kde je umiestnená. Výnimku z tohto pravidla sme uviedli v časti „Pravidlá pre písanie zdrojového kódu“.

## DOKUMENTAČNÉ REĽAZCE

Vo funkciách v metodikách uvádzame dokumentačné reťazce funkcií. Používame formát reStructuredText (<http://docutils.sourceforge.net/docs/user/rst/quickref.html>, <https://www.python.org/dev/peps/pep-0287/>). Dokumentačné reťazce uľahčujú pochopenie použitia funkcie bez nutnosti analýzy jej kódu. Dokumentačné reťazce sú uzatvorené v trojitých apostrofoch a umiestňujú sa hneď pod hlavičku funkcie.

```
def obsah_stvorca(dlзка_strany):  
    ''' Pre zadanú dĺžku strany štvorca vypočíta obsah štvorca  
  
    :param dlзка_strany: dĺžka strany štvorca, dlзка_strany je nezáporné číslo  
    :type dlзка_strany: int | float  
    :return: obsah štvorca  
    :rtype: int | float  
    :raises TypeError: ak dlзка_strany nie je číslo  
    :raises ValueError: ak dlзка_strany je záporná  
    '''  
    try:  
        if dlзка_strany < 0:  
            raise ValueError('Dĺžka strany nesmie byť záporná!')  
    except TypeError:  
        raise TypeError('Hodnota dĺžky strany musí byť číslo!')  
  
    return dlзка_strany ** 2
```

Pre takto dokumentovanú funkciu vývojové prostredie dokáže vygenerovať nasledovnú nápoved':

```
C:/Users/Jan Guniš/.PyCharmEdu2019.3/config/scratches/scratch_164.py
def obsah_stvorca(dlzska_strany: Union[int, float]) -> Union[int, float]
```

Pre zadanú dĺžku strany štvorca vypočíta obsah štvorca

Params: dlzska\_strany – dĺžka strany štvorca, dlzska\_strany je nezáporné číslo

Returns: obsah štvorca

Raises: `TypeError` – ak dlzska\_strany nie je číslo

`ValueError` – ak dlzska\_strany je záporná

Aj žiaci by mali svoje funkcie dokumentovať. Cieľom nie je to, aby žiaci vytvárali dokonalé dokumentačné reťazce, ale aby dokázali o svojom návrhu riešenia uvažovať, stručne vysvetliť čo funkcia rieši a povedať, kedy funkcia nevie spraviť to, čo od nej očakávame. Jednoduchšia verzia dokumentačného reťazca môže obsahovať len slovný popis činnosti funkcie.

```
def obsah_stvorca(dlzska_strany):
    ''' Pre zadanú dĺžku strany štvorca vypočíta obsah štvorca.

    Dĺžka musí byť zadaná ako nezáporné číslo.
    '''
    ...
```

Z úsporných dôvodov v texte metódič dokumentačné reťazce funkcií neuvádzame. V samotných programoch (\*.py) dokumentačné reťazce uvedené sú.

## NÁVRH GRAFICKÉHO ROZHRAANIA

Pri návrhu grafického rozhrania programov využívame modul `tkinter`.

Kedže objektové programovanie nie je podľa iŠVP súčasťou základného kurzu, nevytvárame nové triedy pre grafické rozhranie.

Grafické rozhranie predstavuje prostredníka medzi programom a používateľom. Samotný program by mal byť použiteľný a funkčný aj bez grafického rozhrania (použitie bude trochu náročnejšie, pretože použijeme konzolu). Dôvod je jednoduchý a praktický. Ešte pred tým ako začneme vytvárať grafické rozhranie, vieme otestovať a odladiť program. Grafické rozhranie tak vytvárame nad správne fungujúcim programom a je len prostredníkom medzi programom a jeho používateľom. Ak nastane chyba, vieme ju ľahko odhaliť. Program vieme odladiť bez grafického rozhrania, ktoré potom budujeme nad správne fungujúcim programom.

Uvažujme jednoduchú aplikáciu na výpočet druhej mocniny čísla<sup>4</sup>. Samotný program vyzerá nasledovne:

<sup>4</sup> Definovať funkciu pre druhú mocninu je kontraproduktívne. V tomto prípade sme ju definovali kvôli jej jednoduchosti, aby sme demonštrovali schému tvorby grafického rozhrania.

```
def umocni(x):
    return x ** 2
```

Správnosť funkcie vieme bez problémov otestovať. Grafické rozhranie pre náš „miniprogram“ vytvoríme podľa nasledovnej schémy:

```
import tkinter
from tkinter import messagebox
```

import potrebných modulov,  
tkinter pre tvorbu grafického  
rozhranie,  
messagebox pre zobrazenie  
chybových správ,

```
def umocni(x):
    """ Vráti druhú mocninu čísla

    :param x: číslo pre výpočet druhej mocniny
    :type x: int | float
    :return: druhá mocnina čísla x
    :rtype: int | float
    :raises ValueError: ak zadaná hodnota nie je číslo
    """
    try:
        x = float(x)
    except ValueError:
        raise ValueError('Zadaná hodnota nie je číslo')
    return x ** 2
```

samotné jadro programu,  
funkcia umocni() je funkčná aj bez  
grafického rozhrania a nezávislá na  
okolitom kontexte,

hodnoty, s ktorými pracuje dostane  
v parametroch,

ak pre dané hodnoty nie je schopná  
vypočítať požadované, generuje  
výnimku,

výsledok vracia ako svoju návratovú  
hodnotu,

```
def zobraz_mocninu():
    ''' Spostredkuje prepojenie medzi funkciou umocni()
    a používateľským rozhraním'''
    try:
        mocnina = umocni(vstup.get())
    except ValueError as chyba:
        messagebox.showerror('Chyba!', chyba)
    else:
        vystup.set(mocnina)
```

funkcie, ktoré prepájajú prvky  
grafického rozhrania s jadrom  
programu:

- čítajú dáta zo vstupných  
políček grafického  
rozhrania,
- prečítané dáta posielajú na  
spracovanie funkciám  
programu,
- preberajú výsledky funkcií  
programu,
- zobrazujú výsledky (príp.  
chybové správy) v grafickom  
okne,

```
okno = tkinter.Tk()
okno.title('Umocňovač')
```

vytvorenie grafického okna,  
nastavenie jeho parametrov,

```
tkinter.Label(okno, text='Zadaj číslo:').grid(row=0,
column=0)
vstup = tkinter.StringVar()
tkinter.Entry(okno, textvariable=vstup).grid(row=0,
column=1)

tkinter.Button(okno, text='Vypočítaj mocninu',
command=zobraz_mocninu).grid(row=1, column=0)

tkinter.Label(okno, text='Číslo na druhú:').grid(row=2,
column=0)
```

vytvorenie grafických prvkov a ich  
vloženie do grafického okna,  
prepojenie grafických prvkov (napr.  
tlačidlá) s relevantnými funkciami,

```
vystup = tkinter.StringVar()
tkinter.Entry(okno, textvariable=vystup).grid(row=2,
column=1)

tkinter.Button(okno, text='Koniec',
command=okno.destroy).grid(row=3, column=0)
```

```
tkinter.mainloop()
```

spustenie

slučky

udalostí

mainloop().

Modul `tkinter` umožňuje použiť niekoľko rôznych manažérov rozmiestnenia obsahu. V metodikách používame manažér `grid`. Pri rozmiestnení jednotlivých prvkov grafického rozhrania uvádzame čísla riadka a stĺpca, kde sa má prvok zobraziť.

Prvky grafického rozhrania majú množstvo nastavení (farby, zarovnanie, typy písma atď.) V metodikách sa tejto problematike primárne nevenujeme, aby sme zbytočne neodvádzali pozornosť žiakov od podstatných, koncepčných vecí.

## ŽIACKE CHYBY

Vo výučbe programovania v Pythone sme zaregistrovali nasledovné žiacke chyby:

### *Syntaktické chyby*

- zámena znakov pre operácie, `:` namiesto `/`, `.` namiesto `*`,
- použitie desatinnej čiarky namiesto desatinnej bodky,
- nesprávne odsadenie blokov príkazov (rôzne počty medzier v jednom dokumente),
- zabudnuté odsadenie,
- neuvedenie dvojbodky na konci „hlavičky“ príkazu,

### *Behové chyby*

- neošetrenie vstupných hodnôt (delenie 0, odmocňovanie záporného čísla, zadanie reťazca či desatinného čísla do výrazu, kde sa očakáva celé číslo a pod.),
- chýbajúca konverzia typov dôsledkom čoho program končí s chybou,
- nerozpoznanie situácie, kde môže dôjsť k „nelogickému“ výpočtu, program tak generuje nezmyselný výsledok namiesto generovania zmysluplnej výnimky,
- nerozpoznanie situácie, kde môže dôjsť k vyhodneniu výnimky, program tak generuje „nezrozumiteľnú“ systémovú chybu namiesto zrozumiteľnej chybovej správy,

### *Logické chyby*

- neošetrenie podmienok riešiteľnosti úlohy,

- neošetrenie okrajových podmienok (kritických hodnôt) vo vytvorenom programe,
- použitie desatinnej čiarky namiesto desatinnej bodky,
  - keďže v Pythone hodnoty oddelené čiarkou predstavujú jeden zo spôsobov ako vytvoriť n-ticu, táto zámena nespôsobí syntaktickú, ale logickú, resp. behovú chybu,
- predstava, že telo cyklu sa ukončí prázdny voľným riadkom,
- nezohľadnenie špecifik aritmetiky počítača – nekritické použitie programu s údajovým typom `float` pri riešení rozhodovacích problémov,

#### *Iné chyby*

- nepopisné názvy identifikátorov (premenné, funkcie) z dôvodu kratšieho zápisu,
- chýbajúce dokumentačné reťazce funkcií,
- chýbajúce komentáre k jednotlivým častiam programov,
- nevhodná alebo žiadna dekompozícia problému, výsledkom je tzv. „špagetový kód“, týmto pojmom sa označuje dlhý kód, v ktorom nie je viditeľná logická štruktúra, dôsledkom je nezrozumiteľnosť a následné problematické udržiavanie kódu,
- neefektívne použitie sekvencie `if` príkazov namiesto ich vnorenia,
- paradoxne s predchádzajúcim aj neefektívne vnáranie sekvencie `if-ov`, ide napr. o prípady testovania vhodnosti hodnoty vstupujúcej do funkcie, ak hodnota nie je vhodná, funkcia vyhadzuje výnimku, ďalší test preto nemusí byť vnorený,
- nepoužitie jazykových prostriedkov Pythonu – nekritické prepísanie programu z Pascalu alebo z Javy jedna k jednej do syntaxe Pythonu, pričom Python umožňuje elegantnejšie riešenie,
  - tento problém sa týka skôr tých, ktorí majú skúsenosti s iným programovacím jazykom,
- použitie globálnych premenných, namiesto lepšej dekompozície problému alebo použitia parametrov funkcií,
- otváranie a následné editovanie súborov namiesto otvorenia projektu (tento problém sa týka vývojových prostredí, ktoré pracujú s projektom, nielen s individuálnym súborom),
  - v tomto prípade môžeme prísť o kontext, v ktorom sme konkrétny súbor vytvorili (napr. nastavenie interpretera, väzby na iné súbory),
- chýbajúca alebo nesprávna inicializácia premenných,
- funkcie na vykreslenie objektov (n-uholníky, šípky ...) zmenia stav grafického pera (pozíciu, natočenia a pod.), takže pri opakovanom použití funkcie je problematické určiť stav kresliaceho pera, odporúčame aby funkcie „vrátili“ kresliace pero do stavu, v akom bolo pri ich volaní,

- ak by mal byť počet parametrov príliš dlhý (napr. body lomenej čiary), je výhodnejšie ich „obaliť“ do nejakého kontajnera (napr. zoznam),
- podceňovanie korytnačej grafiky, v metodikách je pre nás korytnačia grafika nástrojom pre lepšie pochopenie konceptov programovania a hľadanie chýb v programe, korytnačia grafika nie je cieľom, navyše, niektoré úlohy sa efektívnejšie riešia v relatívnej grafike (napr. kreslenie hviezdičky).

# 01 ÚVOD DO PROGRAMOVANIA, VÝPOČTY V KONZOLE

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
<b>Algoritmické riešenie problémov:</b> <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>pomocou premenných,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>používať premennú vo výrazoch,</li> <li>tabuľkový kalkulátor – výpočty v tabuľkách, odkaz na bunku v tabuľke.</li> </ul>	
<b>Ciele</b>	
Žiakom osvojované vedomosti a zručnosti	Žiakom rozvíjané spôsobilosti
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>používať matematické výrazy pri vyjadrovaní vzťahov,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti.</li> </ul> <b>Pomocou premenných:</b> <ul style="list-style-type: none"> <li>identifikovať zo zadania úlohy, ktoré údaje musia byť zapamätané, resp. sa menia,</li> <li>riešiť problémy, v ktorých si treba zapamätať a neskôr použiť zapamätané hodnoty vo výrazoch.</li> </ul> <b>Hľadanie a opravovanie chýb:</b> <ul style="list-style-type: none"> <li>rozlišovať chybu pri realizácii od chyby v zápise.</li> </ul> <b>Konzola jazyka Python:</b> <ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>identifikovať chybu a typ chyby (syntaktická, behová, logická) v zápise,</li> <li>vytvárať a používať premennú,</li> <li>používať vhodné, popisné názvy premenných.</li> </ul>	<b>Koncepty informatického myslenia</b>  <b>Algoritmy:</b> <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (výpočet ceny nákupu, výpočet vzdialenosti).</li> </ul> <b>Dekompozícia:</b> <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia – lineárne rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (rozdelenie výpočtu na menšie kroky).</li> </ul> <b>Abstrakcia:</b> <ul style="list-style-type: none"> <li>(ABS3) využiť podstatné prvky problémov (riešiť slovne zadané problémy).</li> </ul>
<b>Riešený didaktický problém</b>	





Majoritné programovacie jazyky používané v školskej informatike často obmedzujú žiaka tým, že mu neposkytujú možnosť riešiť náročnejšie problémy (nízky strop), neposkytujú mu možnosť riešiť rôznorodejšie problémy zasahujúce aj do iných oblastí poznania (úzke steny) alebo sú pre žiaka nezaujímavé pretože ich považuje za staré, nemoderné a v praxi nepoužiteľné. Jazyk Python má potenciál tieto problematické body eliminovať.

Programovanie sa často chápe ako samostatná, s ostatných oblastami disjunktná oblasť. V tejto metodike ukazujeme, že programovanie je formalizovaný nástroj na riešenie problémov.

<i><b>Dominantné vyučovacie metódy a formy</b></i>	<i><b>Príprava učiteľa a pomôcky</b></i>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• frontálna a individuálna forma.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/01/</b> tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu a návod na inštaláciu interpretera jazyka Python a vývojového prostredia.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/01/</b> návod na inštaláciu interpretera jazyka Python a vývojového prostredia.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i><b>Diagnostika splnenia vzdelávacích cieľov</b></i>	
Sebahodnotiaci test v pracovnom zošite.	

## Úvod

Toto je prvá metodika zo série 27 metodík (=27 vyučovacích hodín + 4 hodiny na didaktické testy + 1 hodina na prezentáciu projektov), ktoré sú určené pre základný kurz programovania. Uvedená séria metodík pokrýva celú oblasť Algoritmické riešenie problémov. Jednotlivé metodiky na seba nadväzujú a neodporúčame meniť ich poradie. Odporúčame učiteľom, aby sa oboznámili aj s nasledujúcimi metodikami a získali tak prehľad o celkovej koncepcii tejto série metodík.

Aj keď táto séria metodík nie je závislá na konkrétnom vývojovom prostredí, odporúčame lokálnu inštaláciu jazyka Python a lokálnu inštaláciu pokročilejšieho vývojového prostredia (napr. PyCharm Edu).

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

**Poznámka:**

Pracovný list je jedným z výstupov žiaka. Odporúčame, aby si žiaci jednotlivé vypracované pracovné listy odkladali. Neskôr ich môžu využiť pri opakovaní učiva. Každý pracovný list na konci obsahuje časť „Vedomosti v kocke“ kde sú stručne uvedené poznatky, na osvojovanie ktorých je metodika zameraná.

**Poznámka:**

Pri niektorých úlohách uvádzame riešenie – zdrojový kód programu. Ak program spôsobí výpis na konzolu, nejakú zmenu alebo sme k nemu uviedli komentár, uvádzame to svetlým písmom za znakom #. Napr.:

```
>>>33 * 3 # 99
```

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – diskusia so žiakmi na tému programovanie.
- **Skúmanie (8 minút)** – skúmanie ako Python (v konzole) vyhodnocuje pripravené výrazy (úloha 1 z pracovného listu).
- **Vysvetlenie (5 minút)** – vysvetlenie predchádzajúcich zistení, riešenie úloh (úlohy 2 až 6 z pracovného listu).
- **Rozpracovanie (18 minút)** – riešenie náročnejších úloh (úlohy 7 a 8 z pracovného listu).
- **Vyhodnotenie (4 minúty)** – vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 5 MIN)

V tejto fáze by mali žiaci diskutovať. Ak uznáte vy alebo vaši žiaci niektoré prvky diskusie za dôležité, môžu si ich žiaci zaznamenať do pracovného listu. Na základe iŠVP pre 2. stupeň ZŠ by žiaci mali mať predstavu o tom, čo je to programovanie, aké činnosti v sebe zahŕňa a aký je význam programovania z pohľadu riešenia problémov a z pohľadu uplatnenie človeka v živote. Napriek tomu odporúčame výučbu začať aspoň krátkou diskusiou o programovaní a jeho význame.

### **Poznámka:**

Alternatívne môžeme túto časť realizovať vo dvojiciach. Každý dvojici prideliť jednu z otázok (tém). Po určenom čase každá dvojica informuje zvyšok triedy o svojom závere.

Posledné dve otázky sú formulované negatívne zámerne. Z odpovedí na tieto otázky môžeme zistiť, čo a prečo žiakom spôsobuje problémy a cielene sa na tieto skutočnosti zamerať.

Diskusiu môžeme podnietiť nasledovnými otázkami:

- Kto je to programátor?
- Čo robí programátor?
- Čo je to program?
- Prečo vytvára programátor programy? Sú programy užitočné? Zdôvodnite svoju odpoveď.
- Je užitočné vedieť programovať? Zdôvodnite svoju odpoveď.
- Je programovanie ťažké? Zdôvodnite svoju odpoveď.
- Je programovanie nudné? Zdôvodnite svoju odpoveď.

a pod.

Odporúčame diskusiu viesť smerom k tomu, že programovanie je zaujímavá a vysoko kreatívna činnosť. Vedieť programovať vytvára predpoklady na uplatnenie sa v živote pretože vedieť programovať znamená vedieť riešiť problémy. Na druhej strane by si žiaci mali uvedomiť,

že programovanie je intelektuálne náročná zručnosť, ktorá sa nedá naučiť zo dňa na deň (rýchle riešenie neexistuje). Na to, aby sa z človeka stal dobrý programátorom je potrebný dlhodobý a pravidelný tréning.

**Poznámka:**

Programátor sa niekedy chápe vo význame kóder, t. j. ten, kto píše zdrojový kód programu, ten, kto implementuje návrh riešenia. Vo vyučovaní školskej informatiky je význam slova programátor komplexnejší. Je to analytik, ktorý analyzuje zadaný problém a navrhuje riešenie. Kóder, ktorý riešenie implementuje. Tester, ktorý otestuje správnosť navrhnutého riešenia a v neposlednom rade dizajnér, ktorý vytvára rozhranie pre komunikáciu výsledného programu s jeho používateľom.

Učiteľ stručne predstaví ciele vyučovacej hodiny žiakom: oboznámime sa s jazykom Python a naučíme sa riešiť jednoduché problémy pomocou jazyka Python. Neodporúčame túto časť výučby zbytočne natahovať. Cieľom je motivovať žiakov a nie úplne prediskutovať túto tému.

**SKÚMANIE (CCA 8 MIN)**

V tejto fáze prechádzame od rozprávania o programovaní k samotnému programovaniu. Žiaci by mali spustiť učiteľom vybrané vývojové prostredie. Žiakom stručne opíšeme prostredie (nezachádzajme zbytočne do detailov, ktoré sú v tomto momente nepodstatné). V tejto metodike si vystačíme len s konzolou jazyka Python. Žiaci pracujú s úlohou 1 z pracovného listu. Učiteľ v tejto fáze do priebehu hodiny nezasahuje.

Kľúčové poznatky, ktoré by si mali žiaci z tejto časti odniesť sú:

- konzolu môžeme využiť na jednoduché testy a výpočty,
- hodnoty môžeme pomenovať a použitím mena opakovane používať,
- pri programovaní môžeme robiť chyby a na niektoré chyby nás jazyk Python upozorní.

**Poznámka:**

Konzola je interaktívne rozhranie jazyka Python, v ktorom môžeme písať a spúšťať príkazy, resp. krátke časti programov. Predstavuje užitočný nástroj, ktorý žiaci využijú aj na ďalších hodinách. V konzole vieme testovať (overovať si) časti programov bez toho, aby sme ich museli vkladať do samostatných súborov a tie následne spúšťať.

K histórii zadaných príkazov a výrazov sa dostaneme klávesom šípka hore.

Konzolu v tejto fáze vnímame a používame ako pokročilejšiu kalkulačku, ako priestor pre experimentovanie a skúmanie. Experimentovanie a skúmanie v konzole budeme často používať aj v nasledujúcich metodikách. Konzola nemá tlačidlá, výrazy je preto potrebné písať. Na jednej strane to pôsobí ako nevýhoda, na strane druhej nie sme limitovaní obmedzeným zoznamom tlačidiel.

Najskôr necháme žiakov, aby otestovali konzolu jazyka Python a zistili, čo pomocou tejto „kalkulačky“ vedia vypočítať. Žiaci by mali zapísať a v konzole vyhodnotiť niekoľko výrazov. Do tabuľky v pracovnom liste si žiaci najskôr zapíšu svoju predpoveď výsledku, potom výraz v konzole vyhodnotia a do tabuľky zapíšu skutočnú reakciu jazyka Python v konzole. Na tomto mieste odporúčame upozorniť žiakov, že toto nie je test a ich predpovede nebudeme hodnotiť ako správne, resp. nesprávne (podobne aj v nasledujúcich metodikách). V tejto fáze nám ide len o skúmanie. K výsledkom žiackeho

skúmania sa vrátíme v časti „Vysvetlenie“. V tejto etape záznamy žiakov nevyhodnocujeme ani nekomentujeme.

**Poznámka:**

Túto časť môžu žiaci realizovať vo dvojiciach. Žiaci vo dvojiciach môžu vzájomne komunikovať a lepšie tak pochopiť skúmané javy. Aj diskusia medzi skupinami je vhodná.

Cieľom žiakov nie je to, aby sa zápis v stĺpci „moja predpoveď“ zhodoval so zápisom v stĺpci „skutočnosť“.

**Úloha 1** Skúmajte ako Python vyhodnotí nasledovné výrazy. Najskôr si do tabuľky zapíšte svoju predpoveď a potom si ju overte v konzole jazyka Python. Vyhodnotenie každého z výrazov spustíte klávesom Enter. Dodržte poradie výrazov.

	výraz	moja predpoveď	skutočnosť
1.	1 + 2		3
2.	3 + 4 * 5.0		23.0
3.	1 / 0		ZeroDivisionError: division by zero
4.	10 +* 20		SyntaxError: invalid syntax
5.	x = 10		pomenovanie hodnoty 10 menom x
6.	x		10
7.	x + 5		15
8.	sucet = x + 8		pomenovanie hodnoty 18 (=x + 8 = 10 + 8) menom sucet
9.	sucet		18

**Úloha 2** Skúmajte ako Python vyhodnotí nasledovné výrazy. Najskôr si do tabuľky zapíšte svoju predpoveď a potom si ju overte v konzole jazyka Python. Vyhodnotenie každého z výrazov spustíte klávesom Enter. Dodržte poradie výrazov.

Riešte podľa pokynov učiteľa

	výraz	moja predpoveď	skutočnosť
10.	z		NameError: name 'z' is not defined
11.	z = 15		pomenovanie hodnoty 15 menom z
12.	z		15
13.	z = z + 5		pomenovanie hodnoty 20 (=z + 5 = 15 + 5) menom z, zvýšenie hodnoty premennej z o 5
14.	z		20

Toto je prvotná skúsenosť žiakov s interakciou s jazykom Python. Pri prvých dvoch výrazoch sledujeme, či žiaci vedia interpretovať jednoduché aritmetické výrazy, či si uvedomujú prioritu operátorov a či rozlišujú medzi celým a reálnym číslom (číslo s desatinnou bodkou). Ďalšie dva výrazy sú zámerne chybné, aby žiaci získali skúsenosť s chybou. Ďalších päť výrazov je zameraných na intuitívne chápanie konceptu premennej vo význame pomenovania hodnoty pre jej neskoršie použitie.

Výraz 10. spôsobí chybu a cieľom je, aby si žiaci uvedomili problém použitia neinicializovanej premennej. Rovnako nás bude zaujímať interpretácia operácie „=“, ak sa na oboch stranách výrazu

nachádza rovnaké meno premennej. Predposledný príkaz priradenia obsahuje rovnakú premennú na oboch stranách. V matematike tento zápis vyjadruje rovnicu, ktorá nemá riešenie. V programovaní je význam iný. Menom pomenujeme hodnotu, ktorá je výsledkom výrazu na pravej strane.

**Poznámka:**

Python pracuje s premennými inak ako ostatné jazyky, napr. jazyk Pascal. Viď. Konceptia programovania v jazyku Python, časť Premenné. Odporúčame žiakov viesť k jednoduchému modelu pojmu premenná – premenná je pomenovanie hodnoty pre jej neskoršie použitie.

**Poznámka:**

Vedte žiakov k tomu, aby písali zdrojové kódy správne. Ich zdrojové kódy budú prehľadnejšie, ľahšie čitateľnejšie a pochopiteľnejšie. Viď. Konceptia programovania v jazyku Python časť Pravidlá pre písanie zdrojového kódu. Na tieto pravidlá vás budeme priebežne upozorňovať.

## VYSVETLENIE (CCA 5 MIN)

V tejto fáze sa vraciame k poznámkam žiakov z predchádzajúceho experimentovania. Žiaci by mali vysvetliť, čo zistili. Je vhodné, aby sa do diskusie zapojilo čo najviac žiakov. Učiteľ len usmerňuje a moderuje žiakov, aby vzájomne komunikovali svoje zistenia. Zatiaľ nevysvetľuje.

V časti „Skúmanie“ sme sa zamerali na vyhodnocovanie aritmetických výrazov, na prioritu operátorov, na chyby pri programovaní a na použitie premenných. Stačí, ak na intuitívnej úrovni žiaci budú schopní reprodukovať nasledovné tvrdenia:

- konzolu jazyka Python vieme použiť ako kalkulačku na jednoduché výpočty,
- pri výpočtoch môžeme použiť celé aj reálne čísla,
- pri výpočtoch máme možnosť pomenovať konkrétnu hodnotu a cez toto meno s touto hodnotou ďalej pracovať,
- pomenovať vieme aj výsledok vyhodnotenia nejakého výrazu,
- meno, ktorým hodnotu pomenujeme sa ľahšie pamätá než konkrétna hodnota,
- pri programovaní môžeme robiť chyby:
  - chyby typu  $(1 / 0, z)$  – **rozumiem**, čo chceš spraviť, **ale neviem** to spraviť (behová chyba),
  - chyby typu  $(10 + * 20)$  – **nerozumiem**, čo chceš spraviť (syntaktická chyba).

Ak žiaci riešili aj úlohu 2 očakávame, že budú vedieť vysvetliť aj zistenia z úlohy 2. Ak túto úlohu žiaci neriešili, vysvetlenie je úlohou učiteľa. Vysvetlenie by malo obsahovať:

- použiť meno, ktorým sme žiadnu hodnotu nepomenovali je chyba,
- symbol = sa pri programovaní nepoužíva vo význame rovnosti, slúži na pomenovanie hodnoty,

- ak výraz `s =` obsahuje rovnaké meno premennej na ľavej aj na pravej strane, vyhodnotí sa výraz na pravej strane a výsledná hodnota sa pomenuje menom uvedeným na ľavej strane.

Ak uznáte za vhodné, niektoré koncepty môžete žiakom prepojiť s predchádzajúcimi poznatkami:

- kalkulačky majú funkciu ulož do pamäte, s hodnotou uloženou v pamäti sa dá neskôr pracovať,
- meno bunky v tabuľke (alebo adresa bunky v tabuľke) v prostredí tabuľkového kalkulátora slúži ako pomenovanie hodnoty, vo výrazoch v bunkách tabuľky sa toto meno nahradí hodnotou uloženou v bunke tabuľky.

Túto časť môžeme zakončiť krátkou rekapituláciou toho, čo žiaci zistili. Čo Pythonovská kalkulačka dokáže. S akými typmi hodnôt dokáže pracovať. Ako reaguje na „nekorektné“ požiadavky, ako môžeme využívať premenné a pod. Je vhodné, ak si nové poznatky žiaci poznačia do svojich pracovných listov.

## ROZPRACOVANIE (CCA 18 MIN)

Pri spoločnom riešení nasledujúcich úloh využijeme zistenia žiakov z predchádzajúceho skúmania. V úvodnej časti sme žiakov smerovali k tomu, že programovanie je nástroj na riešenie problémov. Vyššie uvedené výrazy (Úloha 1 a úloha 2) sú možno riešeniami nejakých problémov, v tejto chvíli ale nevieme akých. Otočme teraz prístup. Začnime problémom a hľadáme jeho riešenie. Žiaci pracujú s úlohami 3 až 9 z pracovného listu.

V nasledujúcich úlohách by si mali žiaci osvojovať skúsenosti so zápisom výrazov v programovacom jazyku. Odporúčame, aby žiaci výrazy do konzoly vpisovali.

**Úloha 3** V pokladnici kina sme kúpili 33 vstupeniek. Jedna vstupenka stojí 3 € a 50 centov. Koľko € sme zaplatili za vstupenky?

Riešenie:

```
>>>33 * 3.50 # 115.50
```

Predpokladáme, že pri riešení tejto úlohy nevzniknú nejaké závažnejšie problémy. Možno žiaci vypočítajú cenu nákupu v centoch (namiesto v €) alebo namiesto desatinnej bodky zadajú desatinnú čiarku.

### Poznámka:

Pozor. Výraz `33 * 3,50` je syntakticky správny (nespôsobí pri vyhodnotení chybu). Python ho však vyhodnotí nasledovne:

najskôr vynásobí `33 * 3`, výsledkom je teda `99, 50`

hodnoty oddelené čiarkou predstavujú jeden zo spôsobov ako vytvoriť n-ticu (tuple), takže výsledkom je n-tica vo výpise ktorej sú uvedené aj zátvorky: `(99, 50)`.

Ak by sme na začiatku použili výraz `33 * 3,5`, tak výsledok bude `(99, 5)`.

O niečo zaujímavejšia je nasledovná úloha.

**Úloha 4** V pokladnici kina sme kúpili 23 vstupeniek pre spolužiakov, ktorí mali záujem zúčastniť sa predstavenia. Jedna vstupenka stojí 3 € a 50 centov. Neskôr sa rozhodlo pre návštevu kina ďalších 9 spolužiakov, ktorým sme vstupenky dodatočne dokúpili. Koľko € sme zaplatili za vstupenky celkom?

Riešenie:

```
>>>23 + 9 * 3.5 # 54.5 - chybný výsledok
```

alebo niektoré z nasledovných riešení (prípadne nejaké ďalšie, logicky správne riešenia)

```
>>>(23 + 9) * 3.5 # 112.0
>>>23 * 3.5 + 9 * 3.5 # 112.0
>>>32 * 3.5 # 112.0
```

Ak uvedenú chybu nespravlia žiaci, môžeme ju zámerne spraviť my. Dôvodom je upozorniť žiakov na chyby, ktoré sú neodmysliteľnou súčasťou práce programátora (a nie len jeho). Toto je pomerne ľahko odhaliteľná chyba, ale neskôr budeme robiť (i keď neúmyselne) aj ťažšie odhaliteľné chyby. Zároveň by si žiaci mali uvedomiť, že tento typ chýb (logická chyba) je v porovnaní s predchádzajúcimi typmi (syntaktická, behová) ťažšie odhaliteľný. Python nás na ňu totiž neupozorní. Prístup, ktorého výsledkom je posledné z uvedených správnych riešení, môže v budúcnosti spôsobovať problémy. Časť výpočtu  $(23 + 9)$  realizoval žiak mimo počítača a tento krok je náchylnejší na chyby.

**Úloha 5** Pomocou meracieho pásma sme odmerali rozmery telocvične: šírka 15 m a 32 cm, dĺžka 22 m a 12 cm. Koľko plechoviek farby budeme potrebovať na premalovanie podlahy telocvične, ak farba z jednej plechovky vystačí na  $14 \text{ m}^2$ ?

Riešte podľa pokynov učiteľa

Riešenie:

```
>>>(15.32 * 22.12) / 14 # 24.2056
```

alebo

```
>>>(15 + 32 / 100) * (22 + 12 / 100) / 14 # 24.2056
```

Výsledok je potrebné správne interpretovať (25 plechoviek). Nikto asi nepredpokladá, že si môžem kúpiť len časť farby v plechovke.

Všimnime si, že pri takomto prístupe (celý výpočet v jednom riadku) sa nám zadávaný zdrojový kód začína neúmerne predlžovať. Začína byť neprehľadný a preto náchylný na chyby. Odporúčame aj žiakov na tieto skutočnosti upozorniť. Na tomto mieste sa vrátíme k výsledkom skúmania (Úloha 1) k časti manipulácie s premennou (výrazy 5. – 9.) V nasledujúcich úlohách predpokladáme použitie premenných z dôvodov sprehladnenia zápisu výrazov.

**Úloha 6** V stánku rýchleho občerstvenia predávajú: hotdog za 55 centov, hamburger za 1 € a 20 centov a hranolčeky



za 70 centov. Predavač je síce dobrý kuchár, ale zlý počtár. Pomôžme mu, aby vedel cenu nákupu rýchlejšie vypočítať.

Koľko stojí nákup:  $2 \times \text{hotdog}$ ,  $5 \times \text{hamburger}$  a  $3 \times \text{hranolčeky}$ ?

Pomôcka: Ak si niektoré hodnoty pomenujete, pomôže vám to pri výpočtoch.

Riešenie:

```
>>>2 * 0.55 + 5 * 1.2 + 3 * 0.7 # 9.2
```

alebo prehľadnejšie riešenie, kde si hodnoty pomenujeme:

```
>>>hotdog = 0.55
>>>hamburger = 1.2
>>>hranolceky = 0.7
>>>2 * hotdog + 5 * hamburger + 3 * hranolceky # 9.2
```

Predavač si musí pamätať nie len to, koľko ktoré jedlo stojí, ale aj to, koľko z ktorého jedla zákazníkovi predáva. Zatiaľ čo cena je pomerne stabilná, počty predaných kusov sa v každom nákupe menia. Čo ak by sme pamätanie ceny jedla nechali na počítač?

Na tomto mieste odporúčame žiakov upozorniť na vhodné pomenovanie premenných (zmysluplný názov, slova\_s\_podciarkovníkom, bez medzier).

#### Poznámka:

Odporúčame už od začiatku viesť žiakov k tomu, aby si názvy premenných, resp. identifikátory vytvárali v kontexte problému, ktorí riešia. Jednopísmenkové identifikátory (resp. novovytvorené skratky) môžu šetriť čas pri písaní, ale značne sťažujú porozumenie programu. Rovnako neodporúčame vytvárať identifikátory na základe anglických slov. Takto sa identifikátory, najmä začiatovníkom, môžu pliesť s príkazmi jazyka Python.

Nový výraz je prehľadnejší a je pravdepodobné, že pri ňom urobíme menej chýb. Tento krok je zároveň propedeutikou k stratégii riešenia problémov – dekompozícia.

#### Poznámka:

Dekompozícia problému na menšie podproblémy je jednou zo základných stratégií riešenia problémov. Analýzou komplexného a zložitého problému identifikujeme menšie, jednoduchšie časti (podproblémy). Nájdeme ich riešenie a ich syntézou vytvoríme riešenie pôvodného problému. Túto stratégiu budeme v nasledujúcich metodikách používať často.

Opäť sa vrátíme k výsledkom skúmania, presnejšie k posledným dvom výrazom. Vedeli by sme využiť tento nový poznatok v nasledujúcej úlohe?

**Úloha 7** Táto úloha je pokračovaním úlohy 6.

Dodávateľ surovín do stánku zvýšil cenu hranolčekov o 5 %.

Koľko bude stáť nákup:  $1 \times \text{hotdog}$ ,  $3 \times \text{hamburger}$  a  $4 \times \text{hranolčeky}$ ?

Riešenie:

```
>>>1 * hotdog + 3 * hamburger + 4 * hranolceky * 1.05 # 7.09
```

alebo prehľadnejšie riešenie, kde využijeme pomenované hodnoty:

```
>>>hranolceky = 0.7 * 1.05
>>>1 * hotdog + 3 * hamburger + 4 * hranolceky # 7.09
```

alebo ešte lepšie riešenie, kde dôsledne využijeme pomenovanie hodnôt:

```
>>>hranolceky = hranolceky * 1.05
>>>1 * hotdog + 3 * hamburger + 4 * hranolceky # 7.09
```

Predpokladáme, že žiaci nepoužijú posledné z riešení. Ak bude dodávateľ meniť ceny surovín, bude pomerne ťažké neurobiť vo výraze chybu. Navyše si musíme pamätať jednotlivé ceny.

Šikovnejšie je upraviť cenu konkrétneho jedla a ďalej počítať s upravenou cenou – využitím pomenovania novej hodnoty. Odporúčame žiakov viesť k tomuto postupu.

**Úloha 8** Táto úloha je pokračovaním úloh 6. a 7.

Riešte Koľko zaplatí zákazník za 3 hamburgery?

podľa

pokynov

učiteľa

Riešenie:

```
>>>3 * hamburger
3.5999999999999996
```

Výsledok nie je chybný. Počítač pracuje správne a výpočty v Pythone sú tiež správne. Takýto výsledok je dôsledkom prevádzania čísiel medzi dvojkovou a desiatkovou sústavou. My ho však budeme interpretovať ako 3,6.

Riešenie tejto úlohy je triviálne. Zaujímavejšia je interpretácia výsledku úlohy. Výsledkom nie je očakávaná hodnota 3,6 ale hodnota 3,5999999999999996.

Žiakov zrejme prekvapí „chybné“ správanie sa počítača. Výsledok však nie je chybou, ale dôsledkom interpretácie čísiel v počítači. Zatiaľ čo mi pracujeme v desiatkovej sústave, počítač čísla kóduje v sústave dvojkovej. Čísla, ktoré majú konečný rozvoj v jednej sústave, môžu byť v inej sústave vyjadrené ako čísla s nekonečným rozvojom. Pri použití takýchto čísiel vo výpočtoch sa prejaví uvedené „chybné“ počítanie počítača. Tejto problematike nie je na tomto mieste potrebné venovať veľa času. Ide len o prvotnú skúsenosť žiakov s počítaním pomocou počítača.

Výsledok interpretujeme ako 3,6. Jednou z možností ako predísť tejto situácii je napr. uvažovanie cien v centoch.

**Úloha 9** Na čerpacej stanici predávajú benzín, naftu a LPG. Ceny pre dnešný deň boli stanovené nasledovne:

benzín: 1,234 €/liter,

nafta: 1,109 €/liter,

LPG: 0,520 €/liter.

Ak je kupujúci stálym zákazníkom, dostane zľavu 5%.

1. Koľko má zaplatiť nový zákazník, ktorý natankoval 48,9 litrov benzínu?
2. Koľko má zaplatiť stály zákazník, ktorý natankoval 55 litrov nafty?
3. Koľko má zaplatiť stály zákazník, ktorý natankoval 45,9 litrov benzínu a 41,2 litrov LPG?

Riešenie:

```
>>>benzin = 1.234
>>>nafta = 1.109
>>>lpg = 0.520
>>>po_zlave = 0.95
>>>48.9 * benzin # 60.3426, zaokrúhlene 60.34 €
>>>55 * nafta * po_zlave
# 57.945249999999994, zaokrúhlene 57.95 €
>>>(45.9 * benzin + 41.2 * lpg) * po_zlave
# 74.16136999999999, zaokrúhlene 74.16 €
```

**Úloha 10** Zvuk sa vo vzduchu šíri rýchlosťou približne  $340 \text{ m} \cdot \text{s}^{-1}$ . Svetlo sa šíri rýchlosťou približne  $299\,792\,458 \text{ m} \cdot \text{s}^{-1}$ .

Riešte Ako ďaleko od nás udel blesk, ak sme jeho zvuk začuli 14,2 s po tom, ako sme ho videli?

podľa  
pokynov  
učiteľa

Riešenie:

Ak rýchlosť svetla neuvažujeme.

```
>>>v_zvuk = 340
>>>v_zvuk * 14.2 # 4828.0, výsledok v metroch
```

Ak rýchlosť svetla uvažujeme.

```
>>>v_zvuk = 340
>>>v_svetlo = 299792458
>>>t_svetlo = (v_zvuk * 14.2) / (v_svetlo - v_zvuk)
# 1.6104492780560695e-05
>>>v_svetlo * t_svetlo # 4828.005475527545 metrov
```

Výsledky sú síce rôzne, ale rozdiel sa prejavil až na úrovni milimetrov. Prakticky môžeme tento rozdiel zanedbať. V reálnej situácii teda nemusíme brať rýchlosť svetla do úvahy.

Toto je takmer typická úloha zo školskej fyziky. To, čo ju robí trochu inou je zmienka o rýchlosti svetla. Nechajme žiakov diskutovať o tom, či je potrebné uvažovať aj o rýchlosti svetla a prečo?

Pre potvrdenie výsledku diskusie spravme dva výpočty. V jednom rýchlosť svetla neuvažujeme, teda blesk sme videli v rovnakom čase ako nastal. V druhom prípade uvažujeme aj rýchlosť svetla, teda blesk sme videli s nejakým časovým oneskorením po tom, ako nastal. V oboch prípadoch vychádzajme zo známeho vzťahu pre výpočet rýchlosti:  $v = s / t$ , resp. po úprave:  $s = v \cdot t$ .

#### Poznámka:

Odporúčame sa opäť vrátiť k vytváraniu názvov premenných. Pri viacslóvných premenných odporúčame používať malé písmená s podčiarkovníkom napr. `vyska_cloveka`, `celkovy_priemer_znamok`.

Riešenie v prípade, ak rýchlosť svetla neuvažujeme:

```
>>>v_zvuk = 340
>>>v_zvuk * 14.2 #4828.0, výsledok v metroch
```

Riešenie v prípade, ak uvažujeme aj rýchlosť svetla bude o niečo náročnejšie:

$$s = v_{\text{Svetlo}} \cdot t_{\text{Svetlo}}$$

$$s = v_{\text{Zvuk}} \cdot t_{\text{Zvuk}}, \text{ pričom } t_{\text{Zvuk}} = t_{\text{Svetlo}} + 14,2$$

$$\text{po dosadení a úprave: } t_{\text{Svetlo}} = (v_{\text{Zvuk}} \cdot 14,2) / (v_{\text{Svetlo}} - v_{\text{Zvuk}})$$

$$\text{pre dráhu platí: } s = v_{\text{Svetlo}} \cdot t_{\text{Svetlo}}$$

```
>>>v_zvuk = 340
>>>v_svetlo = 299792458
>>>t_svetlo = (v_zvuk * 14.2) / (v_svetlo - v_zvuk) # 1.6104492780560695e-05
>>>v_svetlo * t_svetlo # 4828.005475527545, výsledok v metroch
```

Výsledky sú síce rôzne, ale rozdiel sa prejavil až na úrovni milimetrov. Prakticky môžeme tento rozdiel zanedbať. V reálnej situácii teda nemusíme brať rýchlosť svetla do úvahy.

## VYHODNOTENIE (CCA 4 MIN)

V záverečnej časti hodiny požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal a nie klasifikácia známku. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

### Sebahodnotiaci test

1.	Ktoré z uvedených výrazov <b>spôsobia</b> pri vyhodnocovaní <b>chybu</b> ? a) 20 / 15 - 15      b) 20 / (15 +/- 10)      c) 20 / -5      d) 20 -/ 5
2.	Aký je <b>výsledok</b> vyhodnotenia <b>posledného</b> z nasledujúcich výrazov? <pre>&gt;&gt;&gt;premenna = 15 &gt;&gt;&gt;premenna = premenna - 10 &gt;&gt;&gt;premenna + 15</pre> 20
3.	Dnes majú v kaviarni 10 % zľavu na všetko. Ak sme si pomenovali cenu kávy a cenu hotdogu, ako vypočítame cenu nákupu jedného hotdogu a jednej kávy?: a) (kava + hotdog) * 10 b) kava + hotdog * 0.9 c) (kava + hotdog) * 0.9 d) (kava + hotdog) - (kava + hotdog) * 0.1

Odporúčame, aby učiteľ uviedol správne odpovede a na záver zhrnul nové poznatky v zmysle:

- programovanie je užitočný nástroj pri riešení problémov,
- konzolu jazyka Python môžeme využívať ako pokročilú kalkulačku, ako priestor na skúmanie (napr. nových príkazov, navrhnutých postupov a pod.),
- pri práci s konzolou (a nie len s ňou) môžeme robiť chyby (syntaktické – počítač nerozumie, behové – počítač nevie vykonať, logické – po vykonaní programu dostaneme nesprávny výsledok), na niektoré z nich Python reaguje, na iné nie,
- v konzole si vieme priebežne pamätať (uchovávať) nejaké hodnoty (pomocou premenných),
- k uloženým hodnotám vieme pristupovať cez ich pomenovanie (meno premennej),
- uchovávanú hodnotu vieme zmeniť, presnejšie povedané novú hodnotu vieme pomenovať starým menom premennej,
- namiesto dlhých výrazov (príkazov) je výhodnejšie si jeden veľký výpočet rozdeliť na niekoľko menších, prehľadnejších a ľahšie kontrolovateľných.

## 02 KORYTNAČIA GRAFIKA

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>v konzole vybraného vývojového prostredia jazyka Python (napr. JetBrains PyCharm Edu, IDLE) zadávať príkazy jazyka Python.</li> </ul>	
<b>Ciele</b>	
Žiakom osvojované vedomosti a zručnosti	Žiakom rozvíjané spôsobilosti
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>identifikovať problém, ktorý sa bude riešiť algoritmicky,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia,</li> <li>uvažovať o vlastnostiach vykonávateľa (napr. korytnačka, grafické pero, robot).</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>doplniť, dokončiť, modifikovať rozpracované riešenie,</li> <li>uvažovať o rôznych riešeniach, navrhovať vylepšenie.</li> </ul> <b>Hľadanie a opravovanie chýb:</b> <ul style="list-style-type: none"> <li>hľadať chybu vo vlastnom, nesprávne pracujúcom programe a opraviť ju.</li> </ul> Vytvoriť Python projekt (priečinkov pre Python súbory). Vytvoriť a spustiť jednoduchý Python program a uložiť ho do súboru s príponou <b>py</b> . Vytvoriť Python program vykresľujúci obrázok tvorený čiarami a kruhmi rôznej farby a rozmerov	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení <b>predpokladať správanie</b> sa jednoduchých programov,</li> <li>(LOG3) využitím logických zdôvodnení <b>detegovať</b> a <b>opravovať chyby</b> v programoch a algoritmoch.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) <b>vytvárať vlastné algoritmy</b> riešiace <b>problém/časti problému</b>.</li> </ul>



EURÓPSKA ÚNIA  
Európsky sociálny fond  
Európsky fond regionálneho rozvoja



OPERAČNÝ PROGRAM  
ĽUDSKÉ ZDROJE



MINISTERSTVO  
ŠKOLSTVA, VEDY,  
VÝSKUMU A ŠPORTU  
SLOVENSKEJ REPUBLIKY



it akadémia

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu  
v rámci Operačného programu Ľudské zdroje

[www.minedu.sk](http://www.minedu.sk) [www.employment.gov.sk/sk/esf/](http://www.employment.gov.sk/sk/esf/) [www.itakademia.sk](http://www.itakademia.sk)

použitím základných príkazov korytnačej grafiky (napr. <code>forward()</code> , <code>left()</code> , <code>penup()</code> , <code>pensize()</code> , <code>dot()</code> , <code>pencolor()</code> , <code>bgcolor()</code> ).	
<b>Riešený didaktický problém</b>	
Tvorba sekvenčných programov patrí k úvodným témam výučby programovania. Častou praxou v našich školách je, že sa sprístupňuje programovaním vykresľovania jednoduchých obrázkov prostredníctvom príkazov korytnačej grafiky. Za metodicky nesprávne pokladáme nasledovné praktiky výučby danej témy: <ul style="list-style-type: none"> <li>• predstavenie veľkého množstva grafických príkazov bezprostredne nevyužitelných pri riešení úloh namiesto použitia malého počtu príkazov a ich postupného uvádzania pri riešení problémov, ktoré si ich vyžadujú,</li> <li>• nekonzistentné používanie oboch prístupov (korytnačej a karteziánskej grafiky) pri riešení grafických úloh, kde postačuje samotná korytnačia grafika,</li> <li>• žiadne alebo len ojedinelé používanie komentárov,</li> <li>• namiesto zmysluplných pôvodných názvov príkazov používanie skratiek príkazov kvôli ušetreniu času.</li> </ul>	
<b>Dominantné vyučovacie metódy a formy</b>	<b>Príprava učiteľa a pomôcky</b>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• individuálna a skupinová forma práce žiakov.</li> </ul>	Pre učiteľa: <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulochy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/02/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> Pre žiaka: <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/02/</b> pracovné súbory pre žiaka a návod na vytvorenie, uloženie a spustenie programu.</li> </ul> Použitie digitálnych nástrojov: NUTNÉ
<b>Diagnostika splnenia vzdelávacích cieľov</b>	
Sebahodnotiaci test v pracovnom zošite.	

## Úvod

Na tejto vyučovacej hodine prejdeme od programovania výpočtových problémov v konzole k programovaniu v editore kódu vybraného vývojového prostredia (napr. JetBrains PyCharm Edu, IDLE). Hodina je zameraná na programovanie kreslenia obrázkov pomocou príkazov korytnačej grafiky. Vytvorené programy sú sekvenčné, nevyužívajú žiadne vlastné funkcie, cykly, vetvenie či premenné. Vo výučbe na tejto hodine sa snažíme rozvíjať informatické myslenie žiakov (hlavne logiku a tvorbu algoritmov) pri programovaní grafických problémov s využitím čo najmenej grafických príkazov, ktoré sprístupňujeme postupne s ohľadom na ich potrebu použitia. V tejto a ďalších metodikách sme uprednostnili prístup `import turtle` pred prístupom `from turtle import *`, ktorý síce umožňuje stručnejší kód, ale zastierajú sa niektoré detaily, napr. ktorému objektu prislúchajú uvedené príkazy (napr. `mainloop()`, `clear()`). Žiakov vedieme k tomu, aby vytvárali projekty a programy v jazyku Python so zmysluplnými názvami (napr. projekt **kreslenie**, programy **vločka**, **šípka**, **hodiny**, **trikolóra**, **strom**).

V priebehu tejto vyučovacej hodiny žiaci vytvoria projekt/priečinok **kreslenie** umiestnený, napr. na lokálnom disku s cestou `c:\Users\<MenoŽiaka>\PycharmProjects\kreslenie\`, do ktorého budú ukladať Python programy, ktoré vytvoria na tejto a nasledovných hodinách. Na konci každej hodiny by si mali žiaci archivovať svoje vytvorené Python programy na školský či privátny cloud.

Po absolvovaní tejto hodiny by mali byť žiaci pripravení na nasledovnú vyučovaciu hodinu, ktorá bude zameraná na rozvíjanie spôsobilosti dekomponovať problémy a hľadať vzory, na tvorbu programov vykresľujúcich zložené obrázky obsahujúce vzor(y) s použitím vlastnej funkcie.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie** (4 minút) – diskusia k významu tvorby obrázkov pomocou programovania.
- **Skúmanie** (15 minút) – skúmanie príkazov korytnačej grafiky a vývojového prostredia (úlohy 1 až 3 z pracovného listu).
- **Vysvetlenie** (5 minút) – diskusia a zhrnutie významu grafických príkazov a základných možností vývojového prostredia.
- **Rozpracovanie** (11 minút) – programovanie grafických problémov (úlohy 4 až 6 z pracovného listu).
- **Vyhodnotenie** (5 minút) – vyriešenie sebahodnotiaceho testu s našim vyhodnotením.

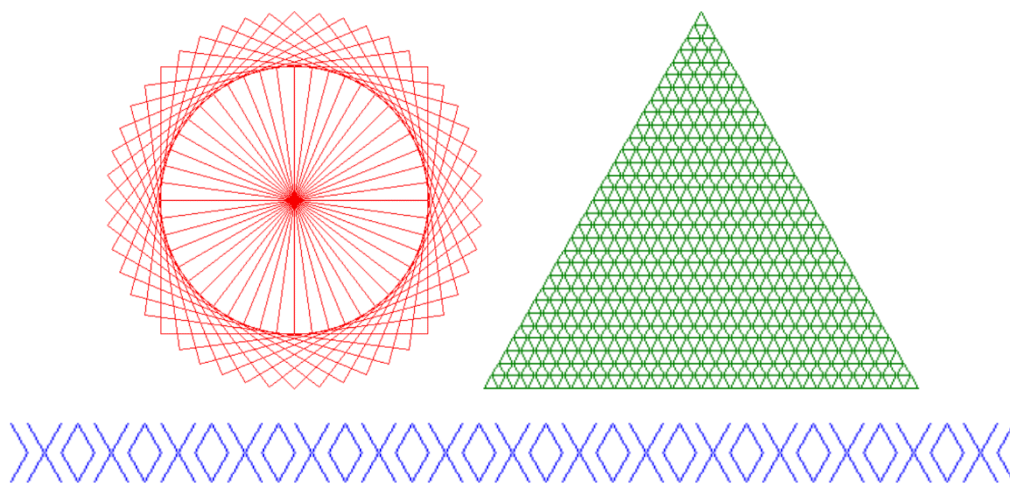


## ZAPOJENIE (CCA 4 MIN)

Začneme diskusiou s celou triedou, ktorou smerujeme žiakov k poznaniu, že obrázky sa dajú kresliť nielen rukou, či v grafických editoroch, ale aj programovať. Ide hlavne o tie obrázky, ktoré vykazujú určitú mieru pravidelnosti a zložitosti štruktúry (veľakrát sa opakujúce vzory, či vzory vnorené do iných vzorov). Týmto chceme dosiahnuť motiváciu žiakov, aby sa naučili používať programovací jazyk aj na kreslenie obrázkov.

Možné otázky v diskusii:

1. Pomocou ktorých nástrojov viete kresliť obrázky?  
(Možné odpovede: ceruzka, pero, štetec, prst, grafický editor, programovací jazyk)
2. Pomocou akého nástroja by ste vykreslili uvedené tri obrázky?  
(Možné odpovede: vektorový editor, programovací jazyk)



3. Závisí výber kresliaceho nástroja od podoby obrázka?  
(Možné odpovede: áno – ak je zložitý a pravidelný obrázok použijem počítač a pri nepravidelnom kreslím voľnou rukou)
4. Ktoré obrázky je lepšie naprogramovať ako práčne kresliť v grafickom editore?  
A naopak, ktoré obrázky je lepšie nakresliť v grafickom editore ako naprogramovať?

## SKÚMANIE (CCA 15 MIN)

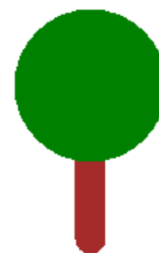
V tejto časti žiaci samostatne (v dvojiciach) prostredníctvom troch úloh postupne preskúmajú správanie vybraných príkazov korytnačej grafiky (úloha 1), oboznámia sa s vývojovým prostredím, v ktorom otvoria hotový program, ktorý preštudujú, spustia ho a upravia, aby vykreslil požadovaný obrázok (úloha 2) a napokon samostatne vytvoria vlastný projekt/priečinok a vlastný program na vykreslenie jednoduchého obrázku (úloha 3). Dôležité je, aby v tejto časti hodiny mali žiaci čo najviac príležitosti na priame a aktívne poznávanie s minimálnou a skôr našou individuálnou pomocou (bez konkrétnych odborných rád, skôr povzbudenie či všeobecne usmernenie), aby mali radosť z vlastného skúmania a objavovania.

**Úloha 1** Preskúmajte, čo sa stane, ak do konzoly postupne zadáte príkazy jazyka Python uvedené v tabuľke. Vedľa príkazu napíšte svoju predpoveď aj zistenú skutočnosť. (Zadávať každý príkaz do konzoly samostatne (aj keď sú v jednej bunke tabuľky 2 riadky) a na potvrdenie príkazov stlačte kláves ENTER).

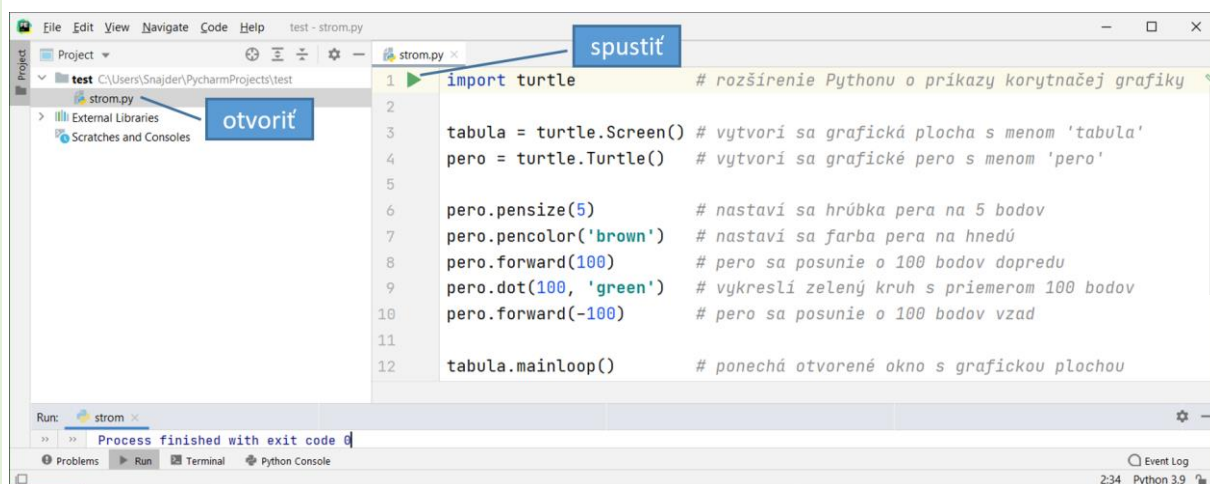
#	Príkaz(y)	Aký výsledok predpovedáte?	Čo ste zistili po spustení príkazov?
1.	<code>forward(100)</code>		Neurobilo nič, len vypísalo chybovú hlášku: <code>NameError: name 'forward' is not defined</code>
2.	<code>import turtle</code> <code>pero = turtle.Turtle()</code>		Spustilo sa okno "Python Turtle Graphics" s grafickým perom natočeným na východ
3.	<code>pero.forward(100)</code>		Grafické pero vykreslilo úsečku dĺžky 100 bodov
4.	<code>pero.left(60)</code>		Grafické pero sa otočilo vľavo o 60 stupňov
5.	<code>pero.penup()</code> <code>pero.forward(100)</code>		Grafické pero sa zdvihlo (vyplo) a posunulo bez kreslenia o 100 bodov dopredu
6.	<code>pero.pendown()</code> <code>pero.backward(100)</code>		Grafické pero sa priložilo na plochu (zaplo) a posunulo (s kreslením) o 100 bodov vzad
7.	<code>pero.clear()</code>		Z kresliaceho plátna sa zmazalo všetko čo nakreslilo grafické pero

Žiaci riešia **úlohu 1**, v ktorej najprv predpovedajú výsledok spustenia uvedených príkazov. Následne experimentovaním – spustením uvedených príkazov v konzole overujú svoje predpovede. Príkaz `forward()` uvedený v 1. riadku tabuľky nie je štandardnou súčasťou jazyka Python, preto interpretér zahlási chybovú správu "NameError: name 'forward' is not defined". Príkazy v 2. riadku umožnia využívať grafické príkazy uložené v module **turtle**. Po úspešnom odoslaní príkazu `pero = turtle.Turtle()` si žiaci štandardne nevšimnú spustenie nového okna s grafikou, na čo ich treba upozorniť. V nasledujúcich riadkoch tabuľky sú uvedené ďalšie základné príkazy korytnačej grafiky, ktorých význam žiaci preskúmajú: `forward()`, `backward()`, `left()`, `penup()` a `clear()`.

**Úloha 2** Janko chce v editore kódu naprogramovať strom uvedený na obrázku. Jeho program **strom.py** však nevykresľuje strom podľa predlohy. Pomôžte mu upraviť jeho program, aby správne vykreslil uvedený strom.



Odporúčame, aby ste najprv otvorili program **strom.py** a spustili jeho kód kliknutím na uvedené oblasti v dolnom obrázku. Program vieme spustiť aj stlačením klávesov **SHIFT + F10**. Následne by ste mali preskúmať vysvetľujúce jednoriadkové komentáre vpravo od príkazov a upraviť program tak, aby vykreslil strom podľa predlohy na obrázku vpravo.



**Riešenie:**

Pôvodné riešenie upravíme na správne, ak pred riadok 6 doplníme príkaz `pero.left(90)`, v riadku 6 zmeníme parameter na 20: `pero.pensize(20)` a pred riadok 9 doplníme príkaz `pero.penup()`. Výsledný kód bude vyzeráť nasledovne s vyznačením dvomi znakmi # v komentároch tých riadkov, v ktorých boli urobené uvedené úpravy:

```
import turtle # rozšírenie Pythonu o príkazy korytnačej grafiky

tabula = turtle.Screen() # vytvorí sa grafická plocha s menom 'tabula'
pero = turtle.Turtle() # vytvorí sa grafické pero s menom 'pero'

pero.left(90) ## natočenie pera vľavo o 90 stupňov
pero.pensize(20) ## nastaví sa hrúbka pera na 20 bodov
pero.pencolor('brown') # nastaví sa farba pera na hnedú
pero.forward(100) # pero sa posunie o 100 bodov dopredu
pero.penup() ## zdvihnutie grafického pera
pero.dot(100, 'green') # vykreslí zelený kruh s priemerom 100 bodov
pero.forward(-100) # pero sa posunie o 100 bodov vzad

tabula.mainloop() # ponechá otvorené okno s grafickou plochou
```

**Úloha 2** je zameraná, jednak na samostatné získanie prvých skúseností s otvorením a spustením hotového programového kódu (uloženého v pracovnom súbore **strom.py** v projekte/priečinku **test**), a jednak na preskúmanie programového kódu a jeho úpravu, aby sa vykreslil strom podľa predlohy. Pri riešení úlohy žiaci objavujú význam nových grafických príkazov `pensize()`, `pencolor()`, `dot()`. Mali by tiež pochopiť rovnocennosť príkazov `forward(-100)` a `backward(100)`. Oboznámia sa so syntaxou jednoriadkových komentárov, ktoré umožňujú autorom zdokumentovať význam častí zapísaného programového kódu. Získajú prvotnú

predstavu o štruktúre grafického programu, jeho úvodnej inicializačnej časti, strednej výkonnej časti a záverečnej časti.

**Úloha 3** Podľa návodu v súbore **I\_SS\_02\_Python\_Navod\_PyCharm.pdf** (resp. **I\_SS\_02\_Python\_Navod\_IDLE.pdf**) vytvorte vo vývojovom prostredí JetBrains PyCharm Edu projekt (resp. pre IDLE priečinok) **kreslenie** a program na vykreslenie veľkého tlačeneho písmena L. Vytvorený program uložte v Python súbore s menom **elko.py**.

- Najprv pouvažujte a uveďte, ktoré grafické príkazy použijete na vykreslenie písmena L:
- Po vytvorení programu, uveďte adresu, kde ste uložili súbor **elko.py** na vašom lokálnom disku:

Riešenie:

- `forward()`, prípadne `backward()` a `left()`
- `c:\Users\Ľubomír\PycharmProjects\kreslenie\elko.py`

```
import turtle                # rozšírenie Pythonu o príkazy korytnačej grafiky

tabula = turtle.Screen()    # vytvorí sa grafická plocha s menom 'tabula'
pero = turtle.Turtle()      # vytvorí sa grafické pero s menom 'pero'

pero.forward(50)             # vykreslí krátke vodorovné rameno písmena L
pero.forward(-50)            # 
pero.left(90)                # natočí sa pero na sever
pero.forward(100)            # vykreslí dlhé zvislé rameno písmena L
pero.forward(-100)           # 

tabula.mainloop()           # ponechá otvorené okno s grafickou plochou
```

Žiakom poskytneme návod v súbore **I\_SS\_02\_Python\_Navod\_PyCharm.pdf**, resp. **I\_SS\_02\_Python\_Navod\_IDLE.pdf** a necháme ich vyriešiť úlohu 3 zameranú na vytvorenie projektu **kreslenie** a programu **elko.py** vykresľujúceho veľké písmeno L. Pri riešení úlohy musia žiaci zostaviť postupnosť príkazov na vykreslenie písmena L a na konci spustiť vytvorený program. Rovnako majú vedieť určiť adresu vytvoreného projektu a programu na vlastnom počítači (napr. `C:\Users\Ľubomír\PycharmProjects\kreslenie\elko.py`).

### VYSVETLENIE (CCA 5 MIN)

V ďalšej časti hodiny necháme najprv žiakov, aby nahlas vysvetlili tri nimi skúmané záležitosti:

- význam jednotlivých príkazov korytnačej grafiky (uvedených v tabuľke úlohy 1),
- štruktúru programu na vykreslenie obrázkov pomocou korytnačej grafiky (uvedenú v úlohe 2),
- postup pri programovaní vykreslenia obrázkov (stručné body návodu v úlohe 3).

Najprv žiakom uvedieme, že vykresľovanie obrázkov postupným zapisovaním príkazov v konzole je veľmi nepraktické, obzvlášť keď sa pomýlime a musíme odznova zapisovať grafické príkazy. Vhodnejšie je použiť editor kódu a postupnosť príkazov uložiť do súboru (s príponou `py`).

Následne zhrnieme, že príkazy korytnačej grafiky nie sú základnou súčasťou jazyka Python, ale sú dostupné importovaním z knižnice príkazov (modulu) **turtle**. Pri kreslení používame príkazy pre grafické pero (`forward()`, `left()`, ...), ktorým kreslíme na grafickú plochu. Ak chceme nechať

otvorené okno s kresbou, použijeme príkaz `mainloop()` pre grafickú plochu (Poznámka: výnimočne v prostredí IDLE sa nemusí použiť príkaz `mainloop()`).

Pred samotným zapisovaním programu do editora kódu je dôležité, aby sme najprv načrtli a analyzovali obrázok s vyznačením uhlov a dĺžok čiar. Pri kreslení útvarov napr. n-uholníkov nepoužívame vnútorné, ale vedľajšie (susedné) uhly, ktoré sú ich doplnkom do 180 stupňov.

Pri práci vo vývojovom prostredí je dôležité spustiť a zastaviť beh programu, poznať miesto uloženia programu, otvárať a zatvárať jednotlivé podokná vývojového prostredia.

Pri vytváraní programu treba pomenovať program výstižným názvom, aby sme mali prehľad o vytvorených programoch v danom projekte/priečinku.

## ROZPRACOVANIE (CCA 11 MIN)

V tejto časti hodiny si žiaci prostredníctvom analytickej úlohy (**úloha 5**) a dvoch úloh na tvorbu programového kódu (**úlohy 4 a 6**) z pracovného listu, precvičia použitie základných príkazov korytnačej grafiky a niektorých stratégií riešenia problémov (napr. nakresli si obrázok, vyskúšaj a over, rozdeľ problém do podproblémov).

**Úloha 4** Vytvorte program **vlocka.py** na vykreslenie vložky uvedenej na obrázku:



Riešenie 1	Riešenie 2	Riešenie 3
<pre># rameno 50 pero.forward(50) pero.forward(-50) pero.left(60)  # rameno 50 pero.forward(50) pero.forward(-50) pero.left(60)  # rameno 50 pero.forward(50) pero.forward(-50) pero.left(60)  # rameno 50 pero.forward(50) pero.forward(-50) pero.left(60)  # rameno 50 pero.forward(50) pero.forward(-50) pero.left(60)  # rameno 50 pero.forward(50) pero.forward(-50) pero.left(60)</pre>	<pre># rameno 100 pero.forward(50) pero.forward(-100) pero.forward(50) pero.left(60)  # rameno 100 pero.forward(50) pero.forward(-100) pero.forward(50) pero.left(60)  # rameno 100 pero.forward(50) pero.forward(-100) pero.forward(50) pero.left(60)</pre>	<pre># rameno 100 pero.forward(100) pero.penup() pero.left(120) pero.forward(50) pero.left(120) pero.pendown()  # rameno 100 pero.forward(100) pero.penup() pero.left(120) pero.forward(50) pero.left(120) pero.pendown()  # rameno 100 pero.forward(100) pero.penup() pero.left(120) pero.forward(50) pero.left(120) pero.pendown()</pre>

Aj keď je precvičovacia **úloha 4** pomerne ľahká, dá sa riešiť rôznymi spôsobmi. Je zaujímavé, aj keď nie nevyhnutné, priamo vo výučbe analyzovať k akým rôznym riešeniam prišli žiaci. V tabuľke sú uvedené ukážky troch riešení: kreslením zo stredu 12, resp. 9 čiar bez zdvihnutia pera (riešenie 1, resp. 2), či kreslením len 3 čiar so zdvíhaním a pokladaním pera na grafickú plochu (riešenie 3).

**Úloha 5** Najprv slovne alebo graficky uveďte aký obrázok vykreslí dole uvedený program:

Potom otvorte program **neznamy.py** a overte, či ste uviedli správny obrázok.

```
import turtle                                # rozšírenie Pythonu o príkazy korytnačej grafiky

tabula = turtle.Screen() # vytvorí sa grafická plocha s menom 'tabula'
pero = turtle.Turtle()   # vytvorí sa grafické pero s menom 'pero'
tabula.bgcolor('lightyellow')

pero.forward(100)
pero.left(45)
pero.forward(-20)
pero.forward(20)
pero.left(-90)
pero.forward(-20)
pero.forward(20)
pero.left(45)
pero.forward(-100)

tabula.mainloop() # ponechá otvorené okno s grafickou plochou
```

Riešenie: Program vykreslí šípku smerujúcu vpravo na žltom pozadí.

Pri **úlohe 5** poskytneme žiakom pracovný súbor **neznamy.py**. Úloha je zameraná na analyzovanie hotového neznámeho programového kódu (vykreslenie šípky smerujúcej doprava na žltej grafickej ploche so žltým pozadím). Žiaci by mali samostatne pochopiť význam nového príkazu `bgcolor()` na nastavenie farby pozadia grafickej plochy.

**Úloha 6** Vytvorte program **hodiny.py** na vykreslenie hodín ukazujúcich čas 10:00 a program **trikolora.py** na vykreslenie trojfarebnej trikolóry podľa predlohy na uvedených obrázkoch:

Riešte  
podľa  
pokynov  
učiteľa



Uveďte, ako by ste postupovali v prípade vykreslenia slovenskej trikolóry s bielou farbou na okraji. (Riešenie: Nastavíme pozadie plátna na inú ako bielu farbu alebo ešte pred vykreslením bieleho kruhu vykreslíme čierny kruh s priemerom o niečo väčším ako priemer bieleho kruhu.)

Riešenie 6A:

```
import turtle                                # rozšírenie Pythonu o príkazy korytnačej grafiky

tabula = turtle.Screen() # vytvorí sa grafická plocha s menom 'tabula'
pero = turtle.Turtle()   # vytvorí sa grafické pero s menom 'pero'
```

```

pero.dot(100, 'yellow') # ciferník

pero.left(90)           # veľká (minútová) ručička
pero.forward(50)
pero.forward(-50)

pero.left(60)           # malá tučná (hodinová) ručička
pero.pensize(3)
pero.forward(30)
pero.forward(-30)

pero.hideturtle()      # skrytie grafického pera
tabula.mainloop()      # ponechá otvorené okno s grafickou plochou

```

**Riešenie 6B:**

```

import turtle           # rozšírenie Pythonu o príkazy korytnačej grafiky

tabula = turtle.Screen() # vytvorí sa grafická plocha s menom 'tabula'
pero = turtle.Turtle()   # vytvorí sa grafické pero s menom 'pero'

pero.dot(100, 'black')   # čierny kruh s priemerom 100
pero.dot(66, 'red')      # červený kruh s priemerom 66
pero.dot(33, 'yellow')   # žltý kruh s priemerom 33

pero.hideturtle()       # skrytie grafického pera
tabula.mainloop()       # ponechá otvorené okno s grafickou plochou

```

Podľa situácie necháme žiakom vyriešiť na hodine časť alebo celú **úlohu 6**. Je zaujímavé zistiť ako vyriešili žiaci vykreslenie trikolóry s bielou farbou na okraji, ktorá má byť rozpoznateľná o pozadia (napr. zmenou farby pozadia na inú ako bielu farbu či vykreslením čierneho kruhu s priemerom o bod väčším ako následného bieleho kruhu).

Príkaz `hideturtle()` na schovanie grafického pera nepredkladáme žiakom v hotovej podobe, ale predstavíme ho podľa potreby – až keď nastane situácia, že žiakom vo vykreslenom obrázku prekáža grafické pero a chcú vidieť obrázok bez neho.

Pri vytváraní a spúšťaní viacerých programov vo vývojovom prostredí JetBrains Pycharm Edu sa v Run paneli v dolnom podokne vytvárajú záložky pre každý spustený program. Pri opätovnom spustení či zastavení patričného programu je potrebné sa prepnúť na záložku s jeho menom.

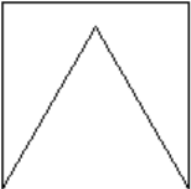
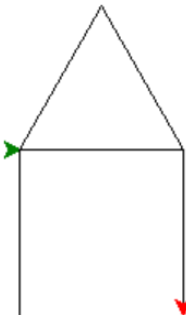

Týmto je zavŕšená precvičovacia časť hodiny. Pre ďalšie prípadné precvičenie programovania kreslenia obrázkov pomocou korytnačej grafiky či na domácu úlohu poslúžia **úlohy 7 až 11** uvedené na konci pracovného listu.



## VYHODNOTENIE (CCA 6 MIN)

V záverečnej časti hodiny necháme žiakom vyriešiť sebahodnotiaci test.

### Sebahodnotiaci test

1.	<p>Zakrúžkujte, ktorý z obrázkov A, B alebo C sa vykreslí pomocou uvedenej postupností príkazov.</p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>pero.forward(100) pero.forward(-100) pero.right(90) pero.forward(100) pero.forward(-100) pero.left(150) pero.forward(100) pero.right(120) pero.forward(100) pero.right(30) pero.forward(100)</pre> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <p><b>Obrázok A</b></p>  </div> <div style="text-align: center;"> <p><b>Obrázok B</b></p>  </div> <div style="text-align: center;"> <p><b>Obrázok C</b></p>  </div> </div> <p>Vo vybranom obrázku vyznačte štartovaciu a cieľovú pozíciu a natočenie grafického pera.</p> <p>Uvedte, koľkokrát (.....) bol použitý príkaz <code>forward()</code> a koľko (.....) úsečiek je vykreslených na obrázku.</p> <p>Uvedte o aký celkový uhol sa natočilo grafické pero v cieľovej pozícii oproti štartovacej pozícii: .....</p> <p>Riešenie:</p> <p>Zakrúžkujeme obrázok B, v ktorom sme vyznačili štartovaciu pozíciu zelenou farbou (pero je natočené na východ) a cieľovú pozíciu červenou farbou (pero je natočené na juh).</p> <p>Príkaz <code>forward()</code> bol použitý 7-krát.</p> <p>Na obrázku je vykreslených 5 úsečiek.</p> <p>Grafické pero sa zo štartovacej pozície do cieľovej pozície natočilo o 90 stupňov vpravo.</p>
----	---

Sebahodnotiaci test poskytne žiakovi spätnú väzbu o tom ako vie analyzovať hotový programový kód a urobiť patričné závery (napr. o počte určitých príkazov a nakreslených úsečiek, natočenie pera). Po vyplnení sebahodnotiaceho testu poskytneme žiakom správne odpovede (obrázok B, štartovacia pozícia: ľavý vrchol rovnostranného trojuholníka smer východ, koncová pozícia: pravý dolný bod smer juh, 7-krát použitý príkaz `forward()`, vykreslených 5 úsečiek, celkové natočenie pera je vpravo 90 stupňov).



## 03 VLASTNÉ FUNKCIE BEZ PARAMETROV A BEZ NÁVRATOVEJ HODNOTY

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>interpretácia zápisu riešenia,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvoriť a spustiť program (tvorený sekvenciou príkazov korytnačej grafiky) a uložiť ho do súboru s príponou <b>py</b>,</li> <li>vysvetliť význam základných príkazov korytnačej grafiky (napr. <code>forward()</code>, <code>left()</code>, <code>penup()</code>, <code>pensize()</code>, <code>dot()</code>, <code>pencolor()</code>, <code>bgcolor()</code>) a aplikovať ich pri tvorbe programov vykresľujúcich obrázky pozostávajúce z úsečiek a kruhov rôznych veľkostí a farieb.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>identifikovať problém, ktorý sa bude riešiť algoritmicky,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia,</li> <li>uvažovať o vlastnostiach vykonávateľa (napr. korytnačka, grafické pero, robot).</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>rozpoznávať opakujúce sa vzory.</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>krokovat riešenie, simulovať činnosť vykonávateľa s postupnosťou príkazov, s výrazmi a premennými, s vetvením a s cyklami,</li> <li>vyjadriť ideu daného návodu (objavovať</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení <b>predpokladať správanie</b> sa jednoduchých programov.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) <b>vytvárať vlastné algoritmy</b> riešiace <b>problém</b>/časti problému,</li> <li>(ALG5) <b>využívať existujúce</b> (vlastné/cudzie) <b>algoritmy</b> v návrhu vlastných algoritmov (z algoritmov riešiacich podproblémy zostaviť algoritmus riešiaci problém),</li> <li>(ALG6) <b>dotvárať nekompletné algoritmy</b> (doplň kód, dokonči program).</li> </ul> Hľadanie vzorov: <ul style="list-style-type: none"> <li>(VZO1) <b>rozpoznať časti</b> objektu/problému/procesu, ktoré majú rovnaké/podobné vlastnosti/pravidlá správania sa.</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) <b>lineárna dekompozícia</b> – lineárne rozdeliť objekty/problémy/procesy (napr. obrázky) na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa.</li> </ul>



<p>a vlastnými slovami popisovať ideu zapísaného riešenia – ako program funguje, čo zápis realizuje pre rôzne vstupy),</p> <ul style="list-style-type: none"> <li>• uvažovať o rôznych riešeniach, navrhovať vylepšenie.</li> </ul> <p><b>Hľadanie a opravovanie chýb:</b></p> <ul style="list-style-type: none"> <li>• hľadať chybu vo vlastnom, nesprávne pracujúcom programe a opraviť ju.</li> </ul> <p>Otvoriť program a modifikovať ho podľa zadania. Vysvetliť rozdiel medzi definovaním vlastnej funkcie a jej volaním v programe. Vytvoriť program využitím vlastných funkcií a základných príkazov korytnačej grafiky.</p>	
<p><b>Riešený didaktický problém</b></p>	
<p>Tvorbu programov využívajúcich vlastné funkcie pokladáme za veľmi dôležitú tému výučby programovania, lebo umožňuje žiakom riešiť problémy hľadaním vzorov a dekompozíciou zadaných problémov na podproblémy. Preto výučbu programovania vlastných funkcií zaraďujeme ako jednu z prvých tém po tvorbe jednoduchých sekvenčných programov.</p> <p>Za metodicky nesprávne pokladáme nasledovné praktiky výučby:</p> <ul style="list-style-type: none"> <li>• posunutie výučby vlastných funkcií až po výučbe cyklov, vetvenia, či zložených údajových typov, čo spôsobuje, že žiaci majú menej príležitostí precvičiť:             <ul style="list-style-type: none"> <li>○ stratégie riešenia problémov – dekompozíciu a hľadanie vzorov,</li> <li>○ použitie vlastnej funkcie bez parametrov,</li> </ul> </li> <li>• absenciu úloh nevyžadujúcich si písanie programového kódu, ktoré sú zamerané na analýzu problémov (obrázkov) a hľadanie vzorov tvoriacich podproblémy (vyskytujúcich sa v obrázkoch).</li> </ul> <p>Funkciu predstavíme metaforicky ako pečiatku so vzorom, pomocou ktorej odtlačíme tento vzor na určité miesta, čím vytvoríme obrázok.</p>	
<p><b>Dominantné vyučovacie metódy a formy</b></p>	<p><b>Príprava učiteľa a pomôcky</b></p>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• individuálna a skupinová forma práce žiakov.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/03/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/03/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<p><b>Diagnostika splnenia vzdelávacích cieľov</b></p>	
<p>Sebahodnotiaci test v pracovnom zošite.</p>	

## Úvod

Na predchádzajúcej hodine žiaci získali prvé skúsenosti s tvorbou Python programov zameraných na korytnačiu grafiku. Na tejto hodine rozšírime programovanie jednoduchých obrázkov na programovanie obrázkov pozostávajúcich z opakujúcich sa vzorov. Najprv necháme žiakov analyzovať obrázky (resp. program) a hľadať v nich opakujúce sa **vzory** (resp. opakujúce sa skupiny príkazov), pre vykreslenie ktorých môžeme použiť metaforu **pečiatky**. Potom po preskúmaní hotového programu by mali žiaci začať chápať rozdiel medzi **definovaním funkcie** (v žiackej reči vytváraním zápisu funkcie) a **volaním funkcie** (v žiackej reči používaním funkcie). V ďalšej časti hodiny analyzovaním, modifikovaním a vytváraním programov si žiaci precvičia poznatky o použití vlastných funkcií pri programovaní kresieb s opakujúcim sa vzorom. Na základe vlastných programátorských skúseností a diskusie na tejto hodine, prípadne na ďalších hodinách by mali žiaci vedieť uviesť a zdôvodniť výhody použitia vlastných funkcií (t. j. prehľadnejší programový kód, znovupoužiteľnosť už raz napísaného programového kódu, lepšia lokalizácia chyby, lepšia deľba tímovej práce). Pri výučbe funkcií uvedieme žiakom aj význam komentárov v procese návrhu funkcie a tiež na zdokumentovanie funkcionality vytvorenej funkcie. V závere hodiny žiaci vyriešia sebahodnotiaci test.

Skúsenosti a poznatky získané z tejto hodiny žiaci využijú pri programovaní obrázkov s opakujúcim sa vzorom na pravidelných pozíciách, na čo bude zameraná nasledovná hodina.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie** (6 minút) – analýza obrázkov a diskusia k vykresľovaniu obrázkov pozostávajúcich z opakujúcich sa vzorov (úlohy 1 a 2 z pracovného listu)
- **Skúmanie** (7 minút) – skúmanie obrázkov/programov obsahujúcich opakujúce sa vzory/skupiny príkazov, ktoré smeruje k potrebe zavedenia nového príkazu (funkcie) pre skupinu príkazov, ktorá je v programe uvedená viackrát (úlohy 3 a 4 z pracovného listu)
- **Vysvetlenie** (7 minút) – diskusia k spôsobu ako definovať a volať vlastný príkaz (funkciu) a tiež k významu jeho použitia pri riešení problémov
- **Rozpracovanie** (14 minút) – programovanie grafických problémov s náročnejším vzorom či viacerými vzormi (úlohy 5 až 7 z pracovného listu)
- **Vyhodnotenie** (6 minút) – vyriešenie sebahodnotiaceho testu s našim vyhodnotením

**ZAPOJENIE (CCA 6 MIN)**

Žiaci riešia úlohy 1 a 2, v ktorých majú analyzovať obrázky a rukou vykresliť (vyznačiť) vzory, z ktorých sú vytvorené uvedené obrázky.

**Úloha 1** Po viacnásobnom otočení maliarskeho valčeka sa na stenu odtlačil uvedený obrázok s ľudovým motívom:



Vyznačte v tomto obrázku vzor, ktorý je nanesený na valčeku. Prípadne tento vzor nakreslite vedľa obrázku.

Uveďte koľkokrát sa otočil valček so vzorom, ktorý na stenu postupne odtlačil uvedený obrázok: .....-krát

Riešenie:

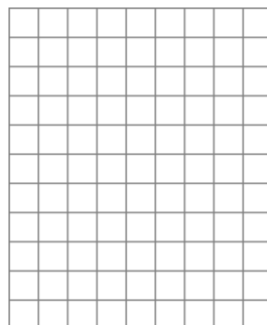
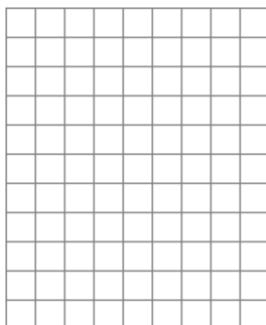
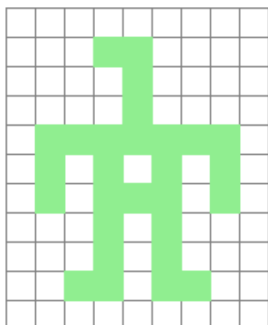


3-krát

V **úlohe 1** je obrázok s ľudovým motívom tvorený jedným vzorom, ktorého pozície sú pravidelne lineárne rozmiestnené. Táto úloha je primárne zameraná na hľadanie opakujúceho sa vzoru v obrázku. Rovnako je aj propedeutickou úlohou pre programovanie cyklov, ale tiež pre programovanie viacerých vnorených funkcií. Úloha je aj inšpiráciou pre neskoršie kreatívne vykresľovanie známych ľudových či vlastných ozdobných motívov (STEAM úloha). Ak žiaci nerozumejú postupu ako sa opakovane odtláča vzor na valčeku, učiteľovi odporúčame, aby si na hodinu priniesol malý valček (napr. valcovú zátku z fľaše) a názorne im ukázal spôsob odtlačania vzoru valčeka natreného farbou na plochu. Inou alternatívou pomoci žiakom je nahradiť metaforu valčeka metaforou pečiatky, ktorú postupne odtláčame na najbližšiu pozíciu vpravo od doteraz odtlačeného obrázka. Alebo im to pečiatkovanie vieme priblížiť na príklade odtlačok mokrých topánok na dlážke.

**Úloha 2** Navrhnite vzor pre pečiatku (pečiatky), pomocou ktorej (ktorých) opečiatkujete celý tvar panáka na obrázku.

Riešte  
podľa  
pokynov  
učiteľa



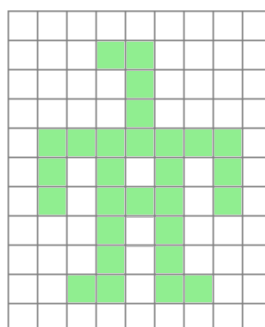
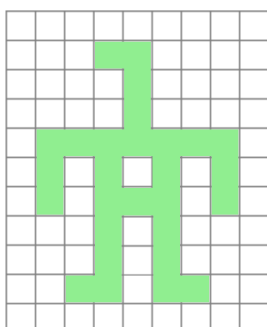
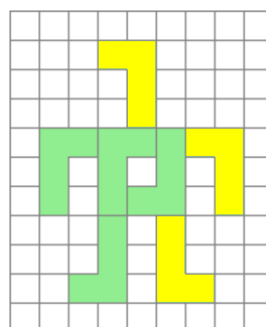
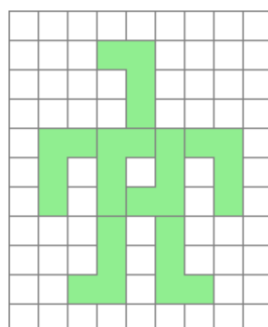
Uvedte, či stačí na opečiatkovanie celého tvaru panáka pečiatka s jedným vzorom:      áno – nie

Ak ste prišli aj na iné riešenia tejto úlohy, zakreslite ich do prázdnych štvorcových mriežok vedľa tej s panákom.

Prediskutujte so spolužiakmi svoje riešenia z pohľadu veľkosti a zložitosti pečiatky a počtu jej opečiatkovaní pri tvorbe panáka.

Riešenie:

Úloha môže mať viacero riešení, ktoré sú uvedené nižšie:



Na opečiatkovanie panáčka stačí pečiatka s jedným vzorom (riešenia na obrázkoch 1, 3 a 4). Riešenie na obrázku 1 predpokladá, že pečiatku v tvare písmena L (s plochou 4 základných štvorcov) vieme otočiť aj na opačnú stranu. Riešenie na obrázku 2 vyžaduje až dva vzory. Ďalšie riešenie úlohy dostaneme, ak vzor v tvare písmena L (s plochou 4 základných štvorcov) rozdelíme na dva rovnaké vzory – obdĺžniky s veľkosťou  $1 \times 2$  základné štvorce.

V **úlohe 2** sa od žiakov očakáva nájdenie vzorov, z ktorých pozostáva zelená postavička panáka vykreslená v štvorcovej sieti. Úloha je formulovaná ako divergentná s viacerými možnými riešeniami, pričom sú žiakom ako pomôcka k dispozícii ďalšie tri štvorcové siete. Túto divergentnú úlohu uzavrieme diskusiou k možným riešeniam a tiež výhodám (obmedzeniam) vykresľovania obrázkov obsahujúcich vzory.

Možné otázky v diskusii:

1. Ako by vyzeralo riešenie s čo najmenším jedným vzorom?  
(Možná odpoveď: vzorom je základný štvorec)
2. Ako by vyzeralo riešenie s čo najmenším počtom odtlačkov jedného vzoru?  
(Možná odpoveď: vzorom je celá postavička panáka)

3. Ako by vyzeralo riešenie s čo najväčším vzorom (vzormi) opakujúci sa aspoň dvakrát v obrázku?  
(Možná odpoveď: jedným vzorom je písmeno L zložené zo 4 základných štvorcov a druhým vzorom je nepriamo zhodný osovo symetrický obraz písmena L)
4. Aké výhody a obmedzenia má vykresľovanie obrázkov pečiatkovaním vzorov oproti vykresľovaniu bez pečiatkovania?  
(Možná odpoveď: výhodou je znovupoužitie pečiatky a možno aj kratší zápis riešenia, obmedzením je, že s danými pečiatkami vzorov nevieme vykresliť ľubovoľný obrázok)

Ak máme dostatok času a patrične motivovaných žiakov, môžeme zaradiť aj ďalšiu otázku:

5. Doplňte obrázok postavičky panáčka o jeden základný štvorec tak, aby bol celý obrázok osovo symetrický podľa zvislej osi. Aké budú iné možné riešenia úlohy s použitím jedného vzoru okrem riešenia so vzorom jedného základného štvorca a riešenia so vzorom celej postavičky panáčka? (Možná odpoveď: vzhľadom na prvočíselný počet základných štvorcov tvoriacich postavičku panáčka, neexistuje žiadne ďalšie riešenie s jedným vzorom okrem spomínaných dvoch riešení)

Ak naopak chceme ušetriť čas, môžeme od každej dvojice žiakov vyžadovať len jedno riešenie úlohy a skrátiť diskusiu ukazujúcu na viaceré možné riešenia úlohy a na výhody kreslenia pomocou pečiatok.

### SKÚMANIE (CCA 7 MIN)

V tejto časti hodiny necháme žiakom riešiť úlohy 3 a 4 z pracovného listu, v ktorých majú preskúmať obrázky a tiež programový kód, ktoré obsahujú opakujúce sa vzory.

**Úloha 3** Na uvedenom obrázku nájdite a (orámčekovaním) vyznačte vzor, z ktorého vieme zostaviť daný obrázok.



V programe na vykreslenie daného obrázka (orámčekovaním) vyznačte časti, ktoré vykresľujú nájdený vzor.

```
import turtle

pero = turtle.Turtle()
tabula = turtle.Screen()
pero.penup()

pero.dot(50, 'black')
pero.forward(50)
pero.dot(50, 'lightgray')
pero.forward(50)
pero.dot(50, 'black')
pero.forward(50)
pero.dot(50, 'lightgray')
pero.forward(50)
pero.right(90)
pero.dot(50, 'black')
pero.forward(50)
pero.dot(50, 'lightgray')
pero.forward(50)

tabula.mainloop()
```

V **úlohe 3** majú žiaci preskúmať obrázok aj odpovedajúci programový kód a v oboch nájsť opakujúci sa vzor. Prvá časť úlohy je (podobne ako úlohy 1 a 2) zameraná na nájdenie opakujúceho sa vzoru. V druhej časti úlohy majú žiaci v programovom kóde vyznačiť časti (skupiny príkazov), ktoré vykresľujú nájdený vzor. Žiakom treba pripomenúť, že kruhy sa dajú vykresľovať aj so zdvihnutým perom. Ak by bolo pero položené dolu, tak by sa pri posunoch do nových pozícií vykresľovali farebné úsečky. Je zaujímavé pozorovať, koľkým žiakom bude prekážať, že sa 3-krát opakujú rovnaké skupiny príkazov. Možno si z predchádzajúcich kurzov programovania niektorí spomenú, že takúto rovnakú skupinu príkazov by mohli označiť ako vlastný príkaz a potom ho v programe uviesť na patričných miestach. Ak nie, nič sa nedeje, lebo sa k tomu vrátíme v nasledujúcej časti Vysvetlenie. Aj keď zadanie úlohy je veľmi netradičné (vyžadujúce si kreslenie rámcikov), časovo nie je veľmi náročné.

**Úloha 4** Preskúmajte uvedený program **corobim.py**. (Poznámka: Na konci riadku 04 s novým príkazom `vzor()` je uvedená dvojbodka. Riadky 05 až 09 sú odsadené dohodnutým počtom medzier, napr. 4)

```

01 import turtle
02
03
04 def vzor():
05     # zápis postupu vykreslenia vzoru = definovanie funkcie vzor()
06     pero.dot(50, 'black')
07     pero.forward(50)
08     pero.dot(50, 'lightgray')
09     pero.forward(50)
10
11
12 tabula = turtle.Screen()
13 pero = turtle.Turtle()
14 pero.penup()
15
16 vzor()          # vykonanie postupu = volanie funkcie vzor()
17 pero.right(90)  # otočenie sa o 90 stupňov vpravo
18 vzor()          # vykonanie postupu = volanie funkcie vzor()
19
20 tabula.mainloop()

```

a) Popíšte alebo nakreslite obrázok, ktorý vykreslí tento program:

Riešenie: Vykreslí obrázok z úlohy 3 bez 2 kruhov

b) Vysvetlite, aký je rozdiel medzi príkazom `def vzor()` : na riadkoch 04 až 09 a príkazom `vzor()` na riadkoch 16 a 18:

Riešenie: Pomocou `def vzor()` definujeme ako sa má vykresliť vzor a pomocou vlastnej funkcie `vzor()` tento vzor vykreslíme.

c) Uveďte, akú zmenu treba urobiť v uvedenom programe, aby vykreslil rovnaký obrázok ako v úlohe 3:

Riešenie: Doplniť pred riadok 16 ešte jedno volanie funkcie `vzor()`

V nadväznosti na úlohu 3, z ktorej možno vzišla potreba použitia nového príkazu (funkcie), necháme žiakom v **úlohe 4** preskúmať hotový program **corobim.py**. V ňom je uvedený kód pre definovanie a volanie vlastného príkazu (funkcie), s ktorým sa prvýkrát stretnú pri programovaní v jazyku Python.

## VYSVETLENIE (CCA 7 MIN)

Po skúsenostiach žiakov zo skúmania riešení úloh 3 a 4 rozviníme diskusiu, pomocou ktorej odhalíme prvotné žiacke predstavy o vytvorení a použití vlastnej funkcie a nasmerujeme žiakov k ich správne pochopeniu:

1. V čom je upravené riešenie úlohy 4 využívajúce vlastné funkcie lepšie, či horšie ako riešenie úlohy 3 bez vlastných funkcií?  
(Možná odpoveď: programový kód je kratší a prehľadnejší)
2. V ktorej časti programu vytvárame vlastnú funkciu a v ktorej ju používame? Mohli by sme najprv použiť funkciu a až potom ju vytvoriť?  
(Možná odpoveď: funkciu vytvárame (=definujeme) na začiatku programu a používame (=voláme) ju až po jej vytvorení (=definovaní))



3. Koľkokrát vytvárame vlastnú funkciu a koľkokrát ju môžeme použiť?  
(Možná odpoveď: funkciu definujeme raz, ale volať ju môžeme koľkokrát chceme/potrebuje)
4. Z akých častí pozostáva zápis vytvorenia vlastnej funkcie?  
(Možná odpoveď: z hlavičky s názvom vlastnej funkcie a z tela odsadeného medzerami)
5. Kde končí zápis vytvorenia vlastnej funkcie?  
(Možné odpovede: (nesprávna) ak za príkazmi tela nasleduje prázdny riadok; ak za príkazmi tela nasledujú príkazy, ktoré už nie sú odsadené)
6. Ako sa zmení vykonávanie programu, ak v ňom neuvedieme komentáre?  
(Možná odpoveď: nijako, komentáre nemajú vplyv na beh programu)
7. Aký význam pre programátora majú komentáre? Nie je to zbytočná strata času pri ich uvádzaní?  
(Možné odpovede: (nesprávna) áno je to zbytočnosť; nie, komentáre nie sú zbytočné, lebo slúžia na zdokumentovanie časti programu, aby sme rýchlejšie (hlavne s odstupom času) pochopili čo robia jednotlivé časti programu)

Po diskusii so žiakmi zhrnieme nové učivo – uvedieme **dekompozíciu** ako jednu zo stratégií riešenia problémov, čo ilustrujeme pomocou schematických zápisov dvoch príkladov funkcií – jednej na vykreslenie obrázku tela zvieráťa pozostávajúceho z hlavy, trupu a končatín; druhej na vypísanie povinností v pracovnom týždni uvedením povinností v jednotlivých pracovných dňoch.

Program na vykreslenie tela zvieráťa	Program na výpis povinností pracovného týždňa
<pre>def hlava():     # príkazy na vykreslenie hlavy  def trup():     # príkazy na vykreslenie trupu  def končatiny():     # príkazy na vykreslenie končatín  hlava() trup() končatiny()</pre>	<pre>def pondelok():     # výpis zoznamu povinnosti v pondelok  def utorok():     # výpis zoznamu povinnosti v utorok  def streda():     # výpis zoznamu povinnosti v stredu  def štvrtok():     # výpis zoznamu povinnosti vo štvrtok  def piatok():     # výpis zoznamu povinnosti v piatok  pondelok() utorok() streda() štvrtok() piatok()</pre>

Vysvetlíme, že pomocou kľúčového slova `def` definujeme **vlastné funkcie** (`hlava ...`, `pondelok ...`), ktorých vykonávanie v programe vyvoláme uvedením ich názvu doplneným o okrúhle zátvorky (podobne ako ich majú uvedené aj ďalšie štandardné príkazy, napr. `forward()` či `penup()`).

Ďalej uvedieme **hľadaj vzor** ako ďalšiu stratégiu riešenia problémov, pomocou ktorej vieme rozpoznať opakujúce sa podproblémy (vzory), čo ilustrujeme pomocou schematických zápisov dvoch príkladov funkcií – na vykreslenie obrázku s jedným, resp. dvoma vzormi (pečiatkami).

Program na vykreslenie obrázku s 1 vzorom	Program na vykreslenie obrázku s 2 vzormi
<pre>def pečiatka():     '''     príkazy na vykreslenie vzoru,     z ktorého pozostáva obrázok     '''  def presun1():     # príkazy na 1. presun graf. pera  def presun2():     # príkazy na 2. presun graf. pera  pečiatka() presun1() pečiatka() presun2() pečiatka()</pre>	<pre>def pečiatka1():     # príkazy na vykreslenie vzoru 1  def pečiatka2():     # príkazy na vykreslenie vzoru 2  def presun1():     # príkazy na 1. presun graf. pera  def presun2():     # príkazy na 2. presun graf. pera  pečiatka1() presun1() pečiatka2() presun2() pečiatka1()</pre>

Zdôrazníme **význam použitia komentárov** pre sprehľadnenie zápisov funkcií programu:

- v procese návrhu funkcie (ako poznámky, čo treba urobiť – „to do“),
- na zdokumentovanie funkcionality funkcie (ako komentáru, čo robí daná funkcia),
- na znefunkčnenie časti kódu (ale to ukážeme až vtedy, ak nastane takáto situácia).

Okrem jednoriadkových komentárov ukážeme žiakom príklad viacriadkového komentára vymedzeného počiatkovou a koncovou trojicou apostrofov (prípadne úvodzoviek). Dokumentačné reťazce uvedieme až pri funkciách s parametrami.

Napokon prediskutujeme so žiakmi a následne zhrnieme **výhody používania vlastných funkcií**:

- písanie prehľadnejšieho kódu odpovedajúceho riešenému problému (riešenie každého podproblému bude reprezentovať jedna funkcia),
- lepšia lokalizácia a opravovanie chýb v programe,
- skrátenie kódu viacnásobnou znovupoužiteľnosťou už raz napísaného programového kódu,
- delba práce medzi viacerých programátorov (každý z nich rieši len vybrané podproblémy).

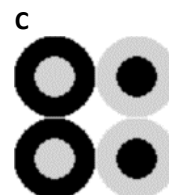
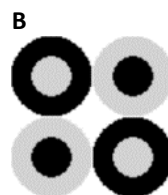
## ROZPRACOVANIE (CCA 14 MIN)

V tejto časti hodiny si žiaci analýzou, modifikáciou a tvorbou programov uvedených v úlohách 5 až 7 z pracovného listu precvičia programovanie vlastných funkcií a osvojené príkazy korytnačej grafiky, ale tiež precvičia použitie rôznych stratégií riešenia problémov – „dekompozíciu“, „hľadaj vzor“ a „načrtni si obrázok“. Obzvlášť pri riešení grafických úloh je náčrt obrázku s uvedenými rozmermi (v štvorcovej či inej mriežke) výbornou pomôckou vo fáze rozboru úlohy.

Pri programovaní funkcií dbáme na používanie priliehavých názvov vlastných funkcií zodpovedajúcich ich správaniu, používanie komentárov k popisu správania funkcie, písanie programov podľa štýlu PEP-8 (funkcie uvádzame spolu na začiatku programu hneď za klauzulou import oddelené navzájom dvomi prázdnymi riadkami).

**Úloha 5** Vytvorte program **stvorgulka2.py** na vykreslenie obrázkov A až C s použitím funkcie na vykreslenie vzoru.

Riešte  
podľa  
pokynov  
učiteľa



Riešenie:

```
import turtle

def vzor():
    # vykreslenie vzoru s dvoma dvojicami sústredných kruhov
    pero.dot(50, 'black')
    pero.dot(25, 'lightgray')
    pero.forward(50)
    pero.dot(50, 'lightgray')
    pero.dot(25, 'black')
    pero.forward(-50)

def obrazok_A():
    # vykreslenie obrázku A
    vzor()
    pero.forward(2 * 50)
    vzor()

def obrazok_B():
    # vykreslenie obrázku B
    vzor()
    pero.forward(50)
    pero.right(90)
    pero.forward(50)
    pero.right(90)
    vzor()

def obrazok_C():
    # vykreslenie obrázku C
    vzor()
    pero.right(90)
    pero.forward(50)
    pero.right(-90)
    vzor()

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.penup()

obrazok_A()
#obrazok_B()
#obrazok_C()

tabula.mainloop()
```

**Úloha 5** je analogická s úlohami 3 a 4, je zameraná na precvičenie tvorby vlastnej funkcie pri kreslení troch (prípadne len dvoch) obrázkov, v ktorých sa opakujú rovnaké vzory. Očakávame, že v žiackom riešení bude súčasťou funkcie vykresľujúcej vzor aj posun grafického pera na nasledujúcu pozíciu v smere kreslenia vzoru. Žiakom tu môžeme ukázať aj alternatívne riešenie, v ktorom funkcia vykreslí vzor a posunie pero na počiatočnú pozíciu kreslenia vzoru. Toto alternatívne riešenie je vhodné pre vykreslenie obrázku C a tiež pre podobné prípady, keď nie je zreteľné postupné nadväzujúce vykresľovanie vzorov za sebou v určitom smere. Veľmi dôležité je, aby si žiaci uvedomili, že volanie vlastnej funkcie (napr. `vzor()`) môžeme použiť v definícii inej vlastnej funkcie (napr. `obrazok_A`).

Príkaz `pero.hideturtle()` predstavíme žiakom až vtedy, kedy si ho sami vyžiadajú – v situácii, keď im prekáža grafické pero pri kompletnom zobrazení vykresleného obrázka.

**Úloha 6** Preskúmajte, čo robí uvedený program `krabicka.py`.

```

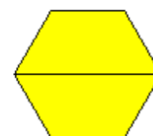
01 import turtle
02
03
04 def obrazok():
05     pero.fillcolor('green')
06     pero.begin_fill()
07     pero.forward(100)
08     pero.left(120)
09     pero.forward(50)
10     pero.left(60)
11     pero.forward(50)
12     pero.left(60)
13     pero.forward(50)
14     pero.left(120)
15     pero.end_fill()
16
17
18 tabula = turtle.Screen()
19 pero = turtle.Turtle()
20
21 obrazok()
22
23 tabula.mainloop()

```

- a) Vysvetlite význam príkazov `fillcolor()`, `begin_fill()`, `end_fill()`:

Riešenie: Príkaz `fillcolor()` nastaví farbu výplne, s ktorou sa vyplní kresba vytvorená skupinou príkazov (na riadkoch 07 až 14) medzi príkazmi `begin_fill()` a `end_fill()`

- b) Uvedený program upravte tak, aby vykreslil nasledovný obrázok:



Riešenie:

```
import turtle
```

```

def obrazok():
    # vykreslenie lichobežníka so žltou výplňou
    pero.fillcolor('yellow')
    pero.begin_fill()
    pero.forward(100)
    pero.left(120)
    pero.forward(50)
    pero.left(60)
    pero.forward(50)
    pero.left(60)
    pero.forward(50)
    pero.left(120)
    pero.end_fill()

def posun():
    # posunutie sa do druhého krajného bodu základne lichobežníka a
    otočenie sa o 180 stupňov
    pero.forward(100)
    pero.left(180)

tabula = turtle.Screen()
pero = turtle.Turtle()

obrazok()
posun()
obrazok()
posun()

pero.hideturtle()
tabula.mainloop()

```

**Úloha 6** je zameraná na analýzu programového kódu obsahujúceho vlastnú funkciu (uloženého v pracovnom súbore **krabicka.py**), na samostatné pochopenie doposiaľ nevysvetlených príkazov na vypĺňanie uzatvoreného útvaru tvoreného postupnosťou čiar (`fillcolor()`, `begin_fill()`, `end_fill()`) a na modifikáciu programu – doplnenie ďalšieho volania funkcie `obrazok()` a doplnenie definície a volania vlastnej funkcie `posun()`.

**Úloha 7** Vytvorte program **stromy.py** na vykreslenie uvedeného obrázku.

Riešte  
podľa  
pokynov  
učiteľa



Riešenie

```

import turtle

def listnaty():
    # vykreslenie listnatého stromu
    pero.pendown()

```

```
pero.pencolor('brown')
pero.forward(100)
pero.penup()
pero.dot(100, 'green')
pero.forward(-100)

def ihlicnaty():
    # vykreslenie ihličnatého stromu
    pero.pendown()
    pero.pencolor('brown')
    pero.forward(130)
    pero.pencolor('green')
    pero.fillcolor('green')
    pero.begin_fill()
    pero.left(150)
    pero.forward(80)
    pero.left(120)
    pero.forward(80)
    pero.left(120)
    pero.forward(80)
    pero.right(30)
    pero.end_fill()
    pero.penup()
    pero.forward(-130)

def posun():
    # posunutie sa na nasledovnú pozíciu vpravo
    pero.right(90)
    pero.forward(100)
    pero.left(90)

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.pensize(20)
pero.left(90)

ihlicnaty()
posun()
posun()
listnaty()
posun()
ihlicnaty()
posun()
listnaty()

pero.hideturtle()
tabula.mainloop()
```


V **úlohe 7** si žiaci precvičia použitie vlastnej funkcie vykresľovaním obrázku tvoreného dvoma vzormi (pre ihličnatý a listnatý strom) a príkazov na vyplňanie uzatvoreného útvaru tvoreného postupnosťou čiar. Očakávame, že v žiackom riešení sa pri vykresľovaní vzorov grafické pero presunie na počiatok vykresľovania vzorov (do dolnej časti kmeňa ihličnatého, resp. listnatého stromu). Túto úlohu skôr odporúčame zaradiť na domácu úlohu.

Týmto je zavŕšená precvičovacia časť hodiny. Na ďalšie prípadné precvičenie tvorby a použitia vlastných funkcií či domácu úlohu poslúžia **úlohy 8 až 12** uvedené na konci pracovného listu.

## VYHODNOTENIE (CCA 6 MIN)

V závere hodiny necháme žiakom vyriešiť sebahodnotiaci test.

### Sebahodnotiaci test

1.	Zakrúžkovaním vyberte skupinu (skupiny) príkazov, pomocou ktorej (ktorých) sa vykreslí uvedený obrázok. <div style="text-align: center;">  </div>								
	<table border="1"> <thead> <tr> <th>Skupina príkazov A</th> </tr> </thead> <tbody> <tr> <td> <pre>def vzor():     pero.dot(40, 'black')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)     pero.dot(40, 'blue')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  vzor() vzor() vzor()</pre> </td> </tr> </tbody> </table>	Skupina príkazov A	<pre>def vzor():     pero.dot(40, 'black')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)     pero.dot(40, 'blue')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  vzor() vzor() vzor()</pre>	<table border="1"> <thead> <tr> <th>Skupina príkazov B</th> </tr> </thead> <tbody> <tr> <td> <pre>def belgicko():     pero.dot(40, 'black')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  def rumunsko():     pero.dot(40, 'blue')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)</pre> </td> </tr> </tbody> </table>	Skupina príkazov B	<pre>def belgicko():     pero.dot(40, 'black')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  def rumunsko():     pero.dot(40, 'blue')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)</pre>	<table border="1"> <thead> <tr> <th>Skupina príkazov C</th> </tr> </thead> <tbody> <tr> <td> <pre>def dvojgulka():     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  def sestgulka():     pero.dot(40, 'black')     pero.forward(40)     dvojgulka()     pero.dot(40, 'blue')     pero.forward(40)     dvojgulka()</pre> </td> </tr> </tbody> </table>	Skupina príkazov C	<pre>def dvojgulka():     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  def sestgulka():     pero.dot(40, 'black')     pero.forward(40)     dvojgulka()     pero.dot(40, 'blue')     pero.forward(40)     dvojgulka()</pre>
Skupina príkazov A									
<pre>def vzor():     pero.dot(40, 'black')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)     pero.dot(40, 'blue')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  vzor() vzor() vzor()</pre>									
Skupina príkazov B									
<pre>def belgicko():     pero.dot(40, 'black')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  def rumunsko():     pero.dot(40, 'blue')     pero.forward(40)     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)</pre>									
Skupina príkazov C									
<pre>def dvojgulka():     pero.dot(40, 'yellow')     pero.forward(40)     pero.dot(40, 'red')     pero.forward(40)  def sestgulka():     pero.dot(40, 'black')     pero.forward(40)     dvojgulka()     pero.dot(40, 'blue')     pero.forward(40)     dvojgulka()</pre>									
	Skupinu (skupiny) príkazov s chybným riešením opravte. Stručne okomentujte uvedené riešenia úlohy:  Riešenie:  Skupina príkazov A obsahuje navyše jedno volanie funkcie <code>vzor()</code> . Ostatné skupiny príkazov B a C sú správnymi riešeniami úlohy. Skupina príkazov B predstavuje riešenie využívajúce dva rôzne vzory vykresľujúce trojice kruhov. Skupina príkazov C predstavuje riešenie využívajúce vzor so šesticou kruhov, ktorý v sebe obsahuje vnorený vzor s dvojicou kruhov.								

V uvedenej úlohe majú žiaci vybrať skupinu (skupiny) príkazov, ktoré vykreslia obrázok s radom farebných kruhov. Po vyriešení úlohy žiakom zdôrazníme rozdiel medzi definovaním a volaním vlastnej funkcie a poskytneme správne riešenie úlohy.

## 04 CYKLUS S PEVNÝM POČTOM OPAKOVANÍ

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>pomocou cyklov,</li> <li>interpretácia zápisu riešenia,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>Vytvoriť Python program využitím vlastných funkcií a základných príkazov korytnačej grafiky.</li> <li>V obrázku s opakujúcim sa vzorom určiť pre každú inštanciu vzoru počiatočnú pozíciu a natočenie grafického pera.</li> </ul>	
<b>Ciele</b>	
Žiakom osvojované vedomosti a zručnosti	Žiakom rozvíjané spôsobilosti
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>identifikovať problém, ktorý sa bude riešiť algoritmicky,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia,</li> <li>uvažovať o vlastnostiach vykonávateľa (napr. korytnačka, grafické pero, robot).</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>používať matematické výrazy pri vyjadrovaní vzťahov a podmienok,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>rozpoznávať opakujúce sa vzory,</li> <li>rozpoznávať aká časť algoritmu sa má vykonať pred, počas aj po skončení cyklu.</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>krokovat riešenie, simulujú činnosť vykonávateľa s postupnosťou príkazov, s výrazmi a premennými, s vetvením a s cyklami</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení <b>predpokladať správanie</b> sa jednoduchých <b>programov</b>,</li> <li>(LOG5) logicky <b>zdôvodniť rozdelenie</b> algoritmu/programu/problému/objektu na menšie časti.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) <b>vytvárať vlastné algoritmy</b> riešiace <b>problém</b>/časti problému.</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK2) hierarchická dekompozícia – <b>hierarchicky rozdeliť</b> objekty/problémy/procesy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa.</li> </ul> Hľadanie vzorov: <ul style="list-style-type: none"> <li>(VZO4) <b>zovšeobecniť</b> riešenie podobných problémov na celú triedu, zovšeobecniť na základe konkrétnych prípadov (namiesto viacerých podprogramov použijeme jeden s viacerými parametrami),</li> <li>(VZO5) <b>preniesť/použiť vzory/myšlienky/riešenia</b> z jedného problému na druhý problém.</li> </ul>





<ul style="list-style-type: none"> <li>vyjadriť ideu daného návodu.</li> </ul> <p><b>Hľadanie a opravovanie chýb:</b></p> <ul style="list-style-type: none"> <li>rozpoznávať, kedy program pracuje nesprávne</li> <li>hľadať chybu vo vlastnom, nesprávne pracujúcom programe a opraviť ju.</li> </ul> <p>Upraviť program s opakujúcimi sa príkazmi bez cyklu na program s cyklom.</p> <p>Vysvetliť význam príkazu <code>for i in range(n)</code> ako príkazu s pevným počtom opakovaní.</p> <p>Charakterizovať (odlíšiť) hlavičku cyklu (zatiaľ len počet opakovaní) a telo cyklu (ako skupinu opakujúcich sa príkazov, ktorá je medzerami odsadená vzhľadom k zápisu hlavičky cyklu).</p> <p>Vytvoriť program využívajúci cyklus <code>for</code>, vlastné funkcie a základné príkazy korytnačej grafiky.</p>	
<p><b>Riešený didaktický problém</b></p>	
<p>Cykly spolu s funkciami (podprogramami) a vetvením patria k základným riadiacim konštrukciám programovacích jazykov. Veľmi často je v školskej praxi zaužívaný prístup, v ktorom sa cyklus <code>for</code> predstaví ešte pred vlastnými funkciami a tiež sa na prvej hodine jeho výučby uvedú a precvičia rôzne varianty použitia riadiacej premennej cyklu <code>for</code>. Za metodicky nesprávne považujeme, ak učiteľ predkladá žiakom úlohy zamerané len na tvorbu rutinných programov a nie úlohy na analýzu a hľadanie chýb v hotových riešeniach, modifikáciu či dopĺňanie nekompletných riešení, a tiež úlohy na tvorbu programov riešiacich zaujímavý problém. Niektorí učitelia nevhodne zaraďujú vnorené cykly pri riešení problémov, kde by mal byť prirodzene použitý jednoduchý cyklus s volaním vlastnej funkcie.</p> <p>V tejto metodike nadväzujeme na výučbu vlastných funkcií a zameriavame sa na využitie stratégií riešenia problémov, hlavne hľadanie vzoru a dekompozície. Cyklus <code>for</code> predstavíme len v jeho najjednoduchšej podobe (ako príkaz „opakuj n-krát niečo“) bez uvádzania detailov riadiacej premennej a funkcie <code>range()</code> a s telom cyklu neobsahujúcim riadiacu, ani iné premenné.</p>	
<p><b>Dominantné vyučovacie metódy a formy</b></p>	<p><b>Príprava učiteľa a pomôcky</b></p>
<ul style="list-style-type: none"> <li>Bádateľská metóda (model 5E),</li> <li>individuálna a skupinová forma práce žiakov.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li><b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li><b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li><b>ucitel/pracovne_subory_riesenia/04/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li><b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li><b>ziak/pracovne_subory/04/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<p><b>Diagnostika splnenia vzdelávacích cieľov</b></p>	
<p>Sebahodnotiaci test v pracovnom zošite.</p>	

## Úvod

Na predchádzajúcej hodine žiaci riešili problémy pomocou stratégií hľadania vzoru a dekompozície, pričom pri programovaní využili vlastné funkcie a príkazy korytnačej grafiky. Na tejto hodine budú žiaci naďalej riešiť problémy využitím uvedených stratégií. Pri programovaní kreslenia opakujúcich sa vzorov na pravidelne rozmiestnených pozíciách použijú príkaz cyklu `for` s pevným počtom opakovaní bez použitia premenných v tele cyklu. Pomocou analyticky formulovaných úloh necháme žiakom preskúmať príkaz cyklu `for` a smerujeme ich k ponímaniu cyklu ako programovej konštrukcie na skrátenie a sprehľadnenie zápisu programového kódu. Zdôrazňujeme pritom význam nájdenia **invariantnej pozície pera**, ktorá sa má zachovať v rámci tela cyklu. Pre lepšie pochopenie cyklu `for` používame metaforu **maliarskeho valčeka**, pomocou ktorého vieme na pravidelné pozície steny zadaný počet krát odtláčať vzor nanesený na valčeku. Následne riešením ďalších úloh precvičujeme a prehľbujeme poznatky z programovania cyklu a vlastných funkcií a tiež uvedených stratégií riešenia problémov. V závere hodiny žiaci vyriešia sebahodnotiaci test.

Na nadväzujúcej vyučovacej hodine zopakujeme a zosumarizujeme učivo prebraté na prvých štyroch vyučovacích hodinách vyučovania programovania v jazyku Python a následne sa budeme venovať riadiacej premennej v cykle `for`, funkcii `range()` a použitiu premenných v tele cyklu.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie** (5 minút) – analýza obrázkov s opakujúcim (aj neopakujúcim) sa vzorom na pravidelných pozíciách a diskusia k spoločným črtám opisov obrázkov (úloha 1 z pracovného listu)
- **Skúmanie** (7 minút) – skúmanie hotových a upravovanie nekompletných príkazov obsahujúcich príkaz cyklu `for`, získanie prvej skúsenosti a predstava o príkaze cyklu `for` (úloha 2 z pracovného listu)
- **Vysvetlenie** (6 minút) – diskusia k ponímaniu príkazu cyklu `for` a naše zhrnutie učiva
- **Rozpracovanie** (14 minút) – programovanie obrázkov s opakujúcimi sa vzormi na pravidelne rozmiestnených pozíciách (úlohy 3 až 5 z pracovného listu)
- **Vyhodnotenie** (8 minút) – vyriešenie sebahodnotiaceho testu s našim vyhodnotením

## ZAPOJENIE (CCA 5 MIN)

Na začiatku hodiny dáme žiakom vyriešiť **úlohu 1**, v ktorej majú analyzovať štyri obrázky, pričom len prvé tri z nich majú opakujúce sa vzory na pravidelných pozíciách.

**Úloha 1** Opíšte **stručne** a čo **najpresnejšie** (pre kamaráta na telefóne), čo vidíte na uvedených obrázkoch **A, B, C, D**:

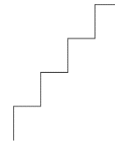
Obrázok A



Obrázok B



Obrázok C



Obrázok D



*Riešenie:*

*A: 7 štvorcov v jednom riadku nalepených tesne vedľa seba*

*B: 10 vzájomne dotýkajúcich sa kruhov zoradených do kruhu*

*C: 4 schody s rovnakou výškou a hĺbkou*

*D: lomená čiara pozostávajúca zo 7 rovnako dlhých na seba nadväzujúcich úsečiek (hore, vpravo, hore, ...)*

*Uvedte, ktoré z obrázkov sa vám opísovali ľahšie (Riešenie: A, B, C) a ktoré ťažšie (Riešenie: D)*

*Uvedte, čo majú spoločné vaše popisy ľahšie opísateľných obrázkov: (Riešenie: Obrázky A, B, C obsahujú vzor, ktorý sa opakuje na pravidelných pozíciách)*

Diskusiou smerujeme žiakov k potrebe zavedenia cyklu s pevným počtom opakovaní, ktorý umožní programovo implementovať skrátený zápis s číslou pri opise obrázkov s opakujúcim sa vzorom na pravidelných pozíciách. Veľmi dôležité je upozorniť žiakov, aby ten popis obrázkov bol nielen stručný, ale aj čo najviac presný, aby sa podľa neho dali (nejakou inou osobou či robotom) nakresliť popisované obrázky.

Možné otázky v diskusii:

1. Uvedte, ktoré z obrázkov sa vám opísovali ľahšie a zdôvodnite prečo opisy niektorých obrázkov boli pre vás ľahšie ako pri iných obrázkoch.  
(Možná odpoveď: Opisy prvých troch obrázkov sú kratšie a prehľadnejšie, lebo obsahujú opakujúce sa vzory na pravidelných pozíciách)
2. Pomocou akých príkazov viete skrátiť a aj sprehľadniť zápis programu?  
(Možná odpoveď: V jazyku Python sme na skrátenie zápisu programu použili vlastné funkcie, ktoré pomenovali opakujúcu sa postupnosť príkazov. Predtým sme v jazyku Logo použili príkaz cyklu opakuj na skrátenie a aj sprehľadnenie zápisu programu s opakujúcimi sa príkazmi)

## SKÚMANIE (CCA 7 MIN)

V tejto časti hodiny budú žiaci skúmať hotové programy obsahujúce príkaz cyklu `for`, čím autenticky získajú prvé skúsenosti a prvotnú predstavu o jeho fungovaní.

**Úloha 2** Preskúmajte obidva uvedené programy **A** a **B**. Najprv len na základe prečítania uvedeného programového kódu, potom po spustení ich kódov uložených v súboroch **corobim1.py** a **corobim2.py**.

**Program A**

```
import turtle

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.penup()

pero.dot(40, 'orange')
pero.forward(40)
pero.dot(40, 'orange')
pero.forward(40)
pero.dot(40, 'orange')
pero.forward(40)
pero.dot(40, 'orange')
pero.forward(40)
pero.dot(40, 'orange')
pero.forward(40)

tabula.mainloop()
```

**Program B**

```
import turtle

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.penup()

for i in range(5):
    pero.dot(40, 'orange')
    pero.forward(40)

tabula.mainloop()
```

3. Opíšte obrázky, ktoré vykreslia uvedené programy **A** a **B**: (Riešenie: Obidva vykreslia 5 kruhov v rade)
4. Ktorý zo zápisov pokladáte za lepší a prečo: (Riešenie: Druhý zápis je kratší a prehľadnejší)
5. Ako by sa zmenil výsledok programu **B**, ak by sme v riadku 7 namiesto `range(5)` uviedli `range(10)`: (Riešenie: Vykreslil by 10 oranžových kruhov v rade)

V **úlohe 2** majú žiaci preskúmať dva programy, najprv len po prečítaní, potom aj po ich spustení. Riešením prvej podúlohy zistia, že obidva programy vykreslia rovnaký obrázok – rad piatich dotýkajúcich sa oranžových kruhov. V riešení druhej podúlohy by sa mali vyjadriť, že druhý zápis programu je lepší, lebo je kratší a prehľadnejší. Hlbavejším žiakom môžeme položiť otázku, či platí vo všeobecnosti, že zápis programu s cyklom je vždy kratší ako bez cyklu. V ďalšej podúlohe žiaci majú zmeniť parameter vo funkcii `range()` z 5 na 10 a získať tak predstavu o tom, že tento parameter určuje počet opakovaní cyklu `for`.

## VYSVETLENIE (CCA 6 MIN)

Po prvých skúsenostiach s príkazom cyklu `for` (po riešení úlohy 2) necháme žiakov prezentovať a prediskutovať svoju predstavu o príkaze cyklu `for`.

Možné otázky v diskusii:

1. Na čo je dobrý príkaz `for`?  
(Možná odpoveď: Na skrátenie zápisu programu s opakujúcimi sa príkazmi.)
2. Z akých častí sa skladá zápis príkazu `for`?  
(Možná odpoveď: Riadok začínajúci slovom `for` (hlavička cyklu) a riadky s odsadenými príkazmi (telo cyklu).)
3. Ako funguje príkaz `for`, aký význam majú jeho hlavička a telo?  
(Možná odpoveď: V prvom riadku v parametri funkcie `range()` je uvedené koľkokrát sa budú opakovať príkazy, ktoré sú odsadené v nasledovných riadkoch.)
4. Musíme použiť v príkaze `for` vlastné funkcie?  
(Možná odpoveď: V príkaze `for` sa môžu použiť rôzne príkazy, aj vlastné funkcie.)

Následne zhrnieme, že príkaz `for` sa využíva pri riešení problémov, ktoré pozostávajú z viacerých rovnakých (alebo podobných) podproblémov. Dá sa prirovnať k maliarskemu valčeku, ktorý pri otáčaní opakovane odtlačí farbu so zadaným vzorom na stenu. Učiteľ môže použiť aj názornú pomôcku, napr. valcovú korkovú zátku s nafarbeným vzorom. Príkaz `for` sa tiež označuje ako **príkaz s pevným (so známym) počtom opakovaní** alebo aj skráteno **cyklus** `for`.

Poradie	Zápis kódu bez príkazu cyklu FOR	Zápis kódu s príkazom cyklu FOR
1 2 3 4	<code>príkazy()</code> <code>príkazy()</code> <code>príkazy()</code> <code>príkazy()</code>	<code>for i in range(4):</code> <code>príkazy()</code>
1 2 ... počet	<code>príkazy()</code> <code>príkazy()</code> ... <code>príkazy()</code>	<code>for i in range(počet):</code> <code>príkazy()</code>
1 2 3 4 5 6 7	<code>a()</code> <code>b()</code> <code>a()</code> <code>b()</code> <code>a()</code> <code>b()</code> <code>c()</code>	<code>for i in range(3):</code> <code>a()</code> <code>b()</code> <code>c()</code>
1 2 3 4 5	<code>a()</code> <code>a()</code> <code>a()</code> <code>b()</code> <code>c()</code>	<code>for i in range(3):</code> <code>a()</code> <code>b()</code> <code>c()</code>

Zobrazíme tabuľku s príkladmi niekoľkých zápisov syntaxe príkazu `for`. Žiakom zatiaľ stačí uviesť, že cyklus `for` pozostáva z **hlavičky cyklu** (prvého riadku začínajúcim slovom `for` a končiacim dvojbodkou, na ktorú často začiatovníci zabúdajú) a **tela cyklu** (riadkami s odsadenými príkazmi). Príkazy v tele cyklu sa vykonajú toľkokrát, ako je to uvedené v parametri funkcie `range()` v hlavičke

cyklu. Pomocou posledných dvoch riadkov tabuľky uvedieme dva príklady, ktoré by mali žiakom ozrejmiť **vymedzenie tela cyklu**. Zdôraznime, že telo cyklu sa neukončí zaradením voľného riadku, ale zrušením odsadenia príkazov v tele cyklu vzhľadom k hlavičke cyklu. Prípadne môžeme nechať žiakov, aby k uvedenému záveru prišli na základe vlastného skúmania.

Pri kreslení obrázkov pomocou príkazu cyklu `for` je dôležité, aby v tele cyklu boli uvedené príkazy, po vykonaní ktorých sa dostane grafické pero **do určitej významnej pozície**, napr. počiatočný bod celého kreslenia, alebo počiatočný bod kreslenia nasledovnej časti obrázka.

Cykly `for` aj vlastná funkcia umožňujú **skrátiť a sprehľadniť** programový kód. Pomocou vlastných funkcií sme riešenie problému mohli rozdeliť na riešenie rôznych podproblémov, napr. kreslenie obrázkov s rôznymi vzormi alebo s rovnakými vzormi aj na nepravidelných pozíciách. Pomocou cyklu `for` rozdelíme problém do rovnakých alebo podľa určitého pravidla podobných podproblémov, napr. kreslenie pravidelných obrázkov.

### ROZPRACOVANIE (CCA 14 MIN)

V tejto časti hodiny si žiaci programovaním **úloh 3 až 5** z pracovného listu precvičia použitie príkazu cyklu `for` a stratégií riešenia problémov – hľadanie vzorov a dekompozíciu.

**Úloha 3** V uvedenom programe ***schody.py*** vyznačte opakujúce sa časti a upravte ho tak, aby ste pomocou príkazu `for` skrátili jeho zápis.

#### Pôvodný program

```
import turtle

def schod():
    pero.forward(20)
    pero.left(-90)
    pero.forward(50)
    pero.left(90)

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.left(90)

schod()
schod()
schod()
schod()
schod()

tabula.mainloop()
```

#### Upravená časť programu

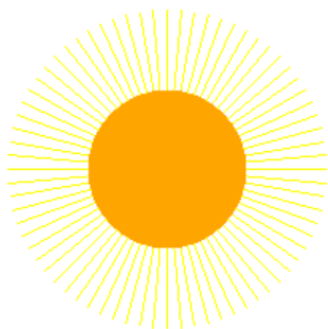
```
for i in range(5):
    schod()
```

V **úlohe 3** použitím vlastnej funkcie `schod` nadväzujeme na predchádzajúce učivo a ukazujeme žiakom príklad prehľadného kódu, ktorý sa vďaka opakujúcim sa príkazom dá skrátiť a sprehľadniť pomocou cyklu `for`. Tu by mali žiaci prísť na to, že opakujúci sa príkaz bude tvoriť telo cyklu.

**Úloha 4** Vytvorte programy *slnko.py* a *stupne\_vitazov.py* vykresľujúce uvedené obrázky A a B tak, aby bolo grafické pero na konci vykreslenia v počiatočnej pozícii a počiatočnom natočení.

Riešte  
podľa  
pokynov  
učiteľa

Obrázok A



Riešenie A:

```
import turtle

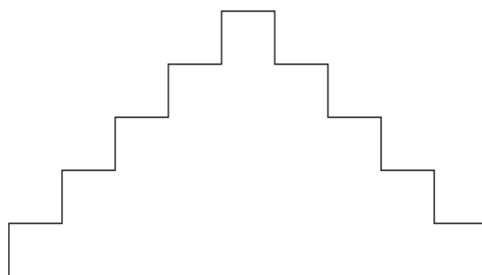
tabula = turtle.Screen()
pero = turtle.Turtle()

# vykreslenie 72 žltých lúčov s dĺžkou 100 bodov
pero.pencolor('yellow')
for i in range(72):
    pero.forward(100)
    pero.forward(-100)
    pero.left(360 / 72)

# vykreslenie oranžového disku s priemerom 100 bodov
pero.dot(100, 'orange')

tabula.mainloop()
```

Obrázok B



Riešenie B:

```
import turtle

def schod():
    # vykreslenie jedného schodu
    pero.forward(40)
    pero.right(90)
    pero.forward(40)
    pero.left(90)

def schody():
    # vykreslenie piatich schodov
    for i in range(5):
        schod()

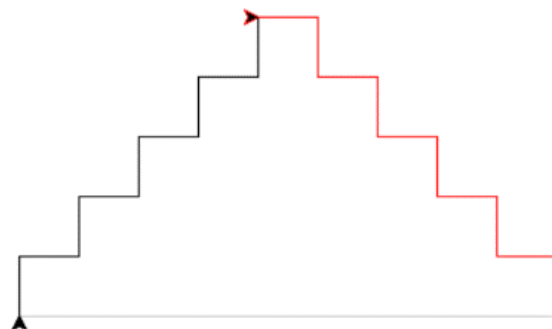
tabula = turtle.Screen()
pero = turtle.Turtle()

pero.left(90)
schody()
pero.right(90)
pero.forward(-40)
schody()
pero.forward(9 * (-40))

tabula.mainloop()
```

**Úloha 4A** na vykreslenie slnka s viacerými lúčmi a kotúčom je pomerne jednoduchou úlohou. Pri nej je dôležité, aby si žiaci uvedomili vzťah medzi počtom opakovaní  $n$  a natočením sa jednotlivých lúčov  $360 / n$  a tiež vykreslenie slnečného kotúča až po vykreslení lúčov. Pri tejto úlohe je vhodné usmerniť žiakov, aby si urobili náčrt riešenia úlohy a premysleli posuny a natočenia korytnačky pri vykreslení každého lúča.

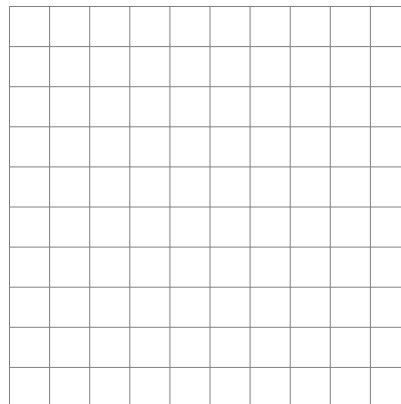
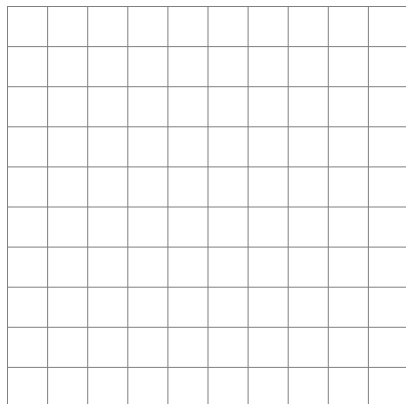
V **úlohe 4B** by mali žiaci dekomponovať problém do 3 podproblémov – schody nahor, schody nadol, spojovacia vodorovná čiara. Pri vykresľovaní schodov nahor a nadol sa dá nájsť rovnaký (aj keď prekývajúci sa) vzor, čo je uvedené na obrázku vpravo. V riešeníach oboch úloh majú žiaci zabezpečiť po vykreslení návrat grafického pera do počiatočnej pozície a počiatočného natočenia. Túto úlohu odporúčame zaradiť len pre šikovnejších žiakov, prípadne za domácu úlohu.





**Úloha 5** Potrebujeme vytvoriť hrací plán pre hru Piškvorky v tvare štvorcovej mriežky  $10 \times 10$ . Do uvedených obrázkov načrtnite jeden, prípadne dva spôsoby vykreslenia tohto **hracieho plánu**. Vo svojom návrhu vyznačte **počiatočný** a **koncový bod** vykresľovania **s natočením** grafického pera a **vzor**, ktorý sa pravidelne opakuje v obrázku.

Riešte  
podľa  
pokynov  
učiteľa



Napokon vytvorte program **mriezka.py** na vykreslenie štvorcovej mriežky pomocou niektorého z navrhnutých spôsobov riešenia.

Riešenie 1:

```
import turtle

def ciarky():
    # vykreslenie 11 vodorovných čiar dĺžky
    100 bodov
    for i in range(11):
        pero.forward(100)
        pero.forward(-100)
        pero.penup()
        pero.left(90)
        pero.forward(100 / 10)
        pero.right(90)
        pero.pendown()

tabula = turtle.Screen()
pero = turtle.Turtle()

# vykreslenie mriežky 11 x 11 čiar dĺžky
100 bodov
ciarky()
pero.right(90)
pero.penup()
pero.forward(100 / 10)
pero.pendown()
ciarky()
pero.penup()
pero.forward(100)
pero.right(90)
pero.forward(100 + 100 / 10)
pero.left(180)

tabula.mainloop()
```

Riešenie 2:

```
import turtle

def stvorec():
    # vykreslenie štvorca
    for i in range(4):
        pero.forward(10)
        pero.left(90)

def rad_stvorcov():
    # vykreslenie radu 10 štvorcov
    for i in range(10):
        stvorec()
        pero.forward(10)
    pero.forward(-10 * 10)

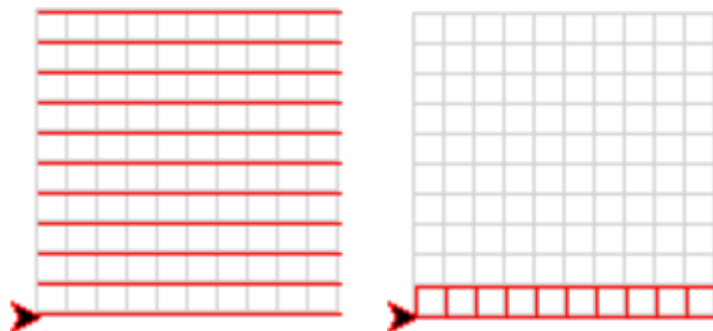
tabula = turtle.Screen()
pero = turtle.Turtle()

# vykreslenie 10 radov 10 štvorcov
for i in range(10):
    rad_stvorcov()
    pero.left(90)
    pero.forward(10)
    pero.right(90)
pero.left(90)
pero.forward(-10 * 10)
pero.right(90)

tabula.mainloop()
```

**Úloha 5** na vykreslenie štvorcovej mriežky  $10 \times 10$  je zameraná na využitie stratégie dekompozície a hľadania vzoru, na precvičenie cyklu a vlastných funkcií. Od žiakov vyžadujeme, aby najprv obrázok analyzovali a do neho zakreslili vzor, ktorý sa v ňom opakuje a tiež počiatočný

a koncovú pozíciu a natočenie grafického pera. Ukážky dvoch príkladov opakujúcich sa vzorov (11 vodorovných čiar, ktorý zopakujeme 2-krát, resp. vodorovný rad 10 štvorcov, ktorý zopakujeme 10-krát) sú uvedené na obrázku:



Pri vykresľovaní obrázkov s väčším počtom čiar žiaci prídu s požiadavkou ako urýchliť vykresľovanie obrázkov. Jedným riešením je nastavenie rýchlosti vykresľovania pera `pero.speed(0)`, ďalším riešením je nastavenie spomalenia vykresľovania plátna `tabula.delay(0)`. Túto úlohu pokladáme za náročnejšiu. V bežnej triede ju môžeme vynechať, resp. dať žiakom riešiť len jej prvú analytickú časť bez tvorby programu.

Týmto je završená precvičovacia časť hodiny. Pre ďalšie prípadné precvičenie tvorby a použitia vlastných funkcií či domácu úlohu poslúžia **úlohy 6 až 10** uvedené na konci pracovného listu.

## VYHODNOTENIE (CCA 8 MIN)

V záverečnej časti hodiny necháme žiakom vyriešiť sebahodnotiaci test. Tento poskytne žiakom spätnú väzbu o tom ako rozumejú príkazu cyklu `for`, t. j. kde končí telo cyklu a ktoré príkazy sú už mimo tela cyklu (**prvá testová úloha**). Rovnako nám tento test umožní zistiť, či žiaci vedia upraviť program tak, aby po vykreslení obrázku s opakujúcim sa vzorom bola grafické pero v pôvodnej pozícii a natočení (**druhá testová úloha**, ktorú odporúčame riešiť podľa pokynov učiteľa).

### Sebahodnotiaci test

1.	Uvedte koľkokrát sa vykoná vlastná funkcia <code>prikaz()</code> v programoch <b>A</b> , <b>B</b> a <b>C</b> .		
	<b>Program A</b> <pre>for i in range(3):     prikaz()     prikaz() prikaz()</pre>	<b>Program B</b> <pre>for i in range(3):     prikaz()      prikaz()</pre>	<b>Program C</b> <pre>for i in range(3):     prikaz() prikaz() prikaz()</pre>
	Riešenie: 7-krát	Riešenie: 6-krát	Riešenie: 5-krát

2. Upravte uvedený program **test.py**, aby vykreslil obrázok časti notovej osnovy tak, aby na konci vykreslenia bolo grafické pero v **počiatočnej pozícii** a v **počiatočnom natočení**. **Dĺžka čiar** notovej osnovy je 200 bodov a **vzdialenosť prvej čiary od piatej** je 100 bodov.

```
import turtle

tabula = turtle.Screen()
pero = turtle.Turtle()

for i in range(5):
    pero.pendown()
    pero.forward(200)
    pero.forward(-200)
    pero.penup()
    pero.left(90)
    pero.forward(100 / 5)
    pero.left(-90)

tabula.mainloop()
```

Riešenie:

```
import turtle

tabula = turtle.Screen()
pero = turtle.Turtle()

# vykreslenie 5 rovnobežných čiar
for i in range(5):
    pero.pendown()
    pero.forward(200)
    pero.forward(-200)
    pero.penup()
    pero.left(90)
    pero.forward(100 / 4)
    pero.right(90)

# presunutie sa na štartovaciu pozíciu
pero.left(90)
pero.penup()
pero.forward(-5 * 100 / 4)
pero.right(90)

tabula.mainloop()
```

Po vyplnení sebahodnotiaceho testu poskytneme žiakom správne odpovede v **prvej testovacej úlohe 6**: postupne 7, 6, resp. 5 vykonaní vlastnej funkcie `prikaz()` pre programové kódy **A**, **B** a **C**. A v **druhej testovacej úlohe**: upravenie výrazu na riadku 12 na `pero.forward(100 / 4)`, lebo pri posunutí o vzdialenosť 100 / 5 bodov by bola prvá čiara od piatej vzdialená 80 bodov, nie požadovaných 100 bodov. Na nastavenie grafického pera na počiatočnú pozíciu a natočenie, je potrebné doplniť za riadok 13 programový kód:

```
pero.left(90)
pero.penup()
pero.forward(-5 * 100 / 4)
pero.left(-90)
```

## 05 OPAKOVANIE I. + DIDAKTICKÝ TEST

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 2 vyučovacie hodiny
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>používať premennú vo výrazoch,</li> <li>vytvárať vlastné funkcie v <b>Pythone</b>,</li> <li>používať cyklus s pevným počtom opakovaní <b>for i in range()</b>.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti.</li> </ul> <b>Hľadanie a opravovanie chýb:</b> <ul style="list-style-type: none"> <li>rozlišovať chybu pri realizácii od chyby v zápise.</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>rozpoznávať opakujúce sa vzory.</li> </ul> <b>Základy jazyka Python:</b> <ul style="list-style-type: none"> <li>riešiť úlohy využitím korytnačej grafiky,</li> <li>vytvárať vlastné grafické funkcie,</li> <li>riešiť náročnejšie úlohy využitím dekompozície, vzorov a abstrakcie.</li> </ul>	Koncepty informatického myslenia  <b>Algoritmy:</b> <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (vytváranie grafických objektov),</li> <li>(ALG6) dotvárať nekompletné algoritmy (doplniť chýbajúci kód v programe).</li> </ul> <b>Hľadanie vzorov:</b> <ul style="list-style-type: none"> <li>(VZO1) rozpoznať časti objektu, ktoré majú rovnaké vlastnosti (hľadanie opakujúcich sa grafických prvkov).</li> </ul> <b>Dekompozícia:</b> <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia – lineárne rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (rozdelenie objektu na menšie časti pre vykreslenie).</li> </ul> <b>Abstrakcia:</b> <ul style="list-style-type: none"> <li>(ABS3) využiť podstatné prvky problémov (riešiť slovne zadané problémy).</li> </ul>
<b>Riešený didaktický problém</b>	
<p>V úlohách v školskej informatike sa na hodinách venuje pri výučbe základov programovacieho jazyka väčšia pozornosť riešeniu veľkého počtu úloh, pri ktorých zadanie nie je komplexne formulované, pričom hlavným cieľom je spravidla zvládnutie a precvičenie syntaxe programovacieho jazyka. V reálnom svete sa však žiaci budú stretávať s problémami, ktoré vyžadujú dôslednú analýzu a určenie, ktoré prvky problému sú potrebné, príp. ktoré je možné v danom štádiu riešenia problému zanedbať. Taktiež je pre žiakov náročnou úlohou vyhľadávanie opakujúcich sa vzorov, ktorých identifikácia napomáha pri dekompozícii problému na menšie, jednoduchšie riešiteľné problémy.</p>	



<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• problémové vyučovanie</li> <li>• frontálna a individuálna forma</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/05/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného,</li> <li>• <b>ucitel/testy/test1/</b> didaktický test, javová analýza testu, tabuľka pre vyhodnotenie testu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/05/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu, výsledky testu I.	

## Úvod

Toto je piata metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Metodika slúži na zopakovanie a systematizáciu poznatkov z predošlých 4 metodík. Zameriava sa na identifikáciu opakujúcich sa vzorov v úlohách a dekompozíciu úloh na čiastkové podproblémy, ktoré žiaci riešia samostatnými funkciami bez parametrov. Metodika využíva problémové vyučovanie, pri ktorom je žiakom najprv predstretý problém vychádzajúci z reálneho života, analýzou ktorého sa žiaci učia využívať dôležité prvky informatického myslenia – hľadanie vzorov, dekompozíciu, ako aj abstrakciu. K samotnému programátorskému riešeniu sa žiaci dostanú až po úvodnej analytickej časti. Analytická časť by sa pri riešení problémov nemala podceňovať – v prípade problému z tejto metodiky ide o pochopenie procesu dekompozície na vytváranie menších funkčných blokov (vlastných funkcií), ktoré budú žiaci neskôr využívať pri modifikáciách riešenia úvodnej úlohy, preto je dôležité ich usmerniť, aby funkcie, ktoré budú (aj neskôr pri iných programátorských úlohách) vytvárať boli naozaj opakovane využiteľné, čím dôjde k zefektívneniu vytvárania zdrojového kódu.

Po absolvovaní tejto metodiky by mal učiteľ zaradiť hodinu určenú na preverenie žiackych vedomostí formou testu, ktorého príklad (spolu s riešením a javovou analýzou) je prílohou tejto metodiky.

## PRIEBEH VÝUČBY

Osnova prvej (opakovacej) vyučovacej hodiny:

- **Úvod (10 minút)** – rozhovor so žiakmi – návrh riešenia modelovej úlohy.
- **Precvičovanie – samostatná práca (27 minút)** – riešenie modelovej úlohy a ďalších úloh z pracovného listu.
- **Zhrnutie (3 minúty)** – diskusia.

Druhá vyučovacia hodina je určená na riešenie testu žiakmi a diskusiu o žiackych riešeniach.

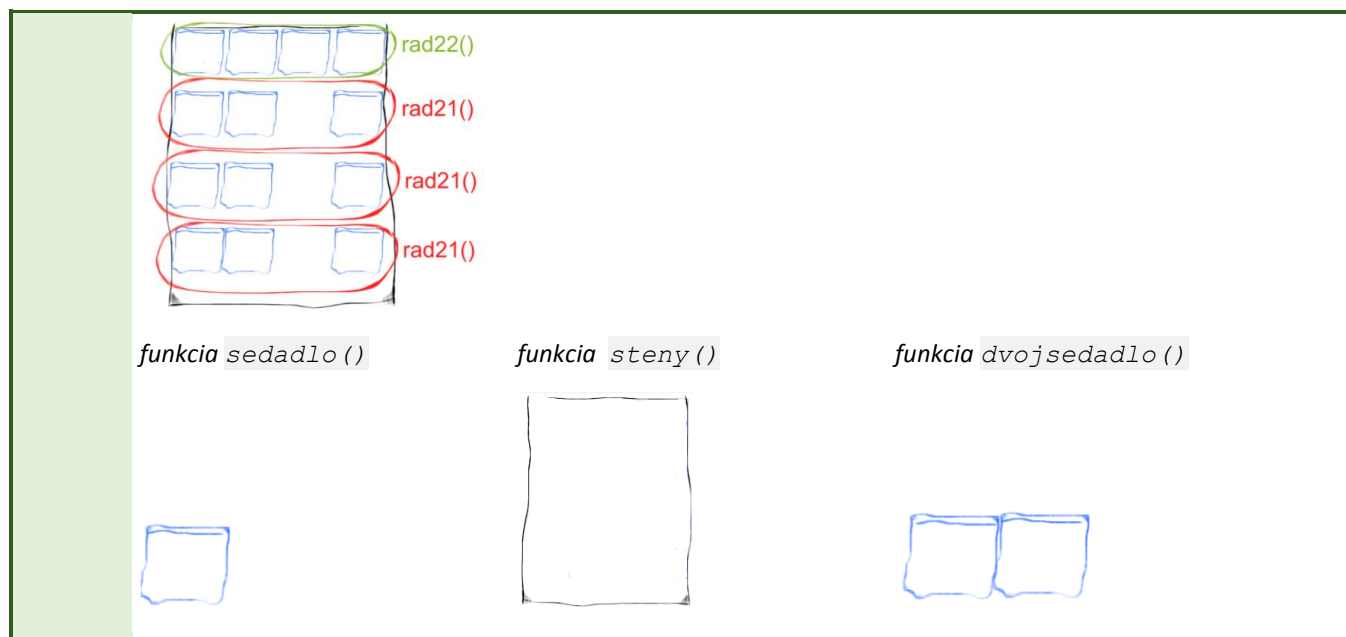
## Úvod (cca 10 min)

Hodinu začneme frontálnym motivačným rozhovorom – žiakom pomocou dataprojektora premietneme prvú snímku prezentácie so zadáním problémovej úlohy. Pri riešení tejto úlohy podporujeme žiakov, aby sa ich čo najviac zapojilo do návrhov riešenia.

**Úloha 1** Prepravná spoločnosť nás požiadala o návrh objednávkového systému, preto je potrebné vytvoriť program, ktorý by vykresľoval rozloženie miest v ich autobusoch. Načrtnite, ako by mohol vyzeráť grafický výstup programu a navrhните preň potrebné funkcie.

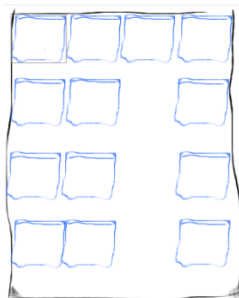
Riešenie:



**Poznámka:**

Úloha zámerne neposkytuje úzko špecifikované zadanie a požiadavky na podobu výstupu/riešenia. Cieľom je vytvoriť pracovnú diskusiu, pri ktorej žiaci budú nútení klásť čo najviac otázok, ktorými sami môžu poukázať na niektoré prvky, ktoré by mohli byť pri riešení dôležité (koľko radov sedadiel je vlastne v autobuse? je tam viacero dverí, ku ktorým vedú bočné uličky? potrebujeme uvažovať aj sedadlo vodiča? a iné...). Všetky tieto úlohy nám umožňujú sledovať myslenie žiakov a spoločne s nimi vytvárať model, ktorý budeme riešiť. Napr. pri otázke o sedadle vodiča sa môžeme zamerať na zadanie úlohy, v ktorom ide o návrh objednávkového systému pre cestujúcich, teda či je do tohto systému vhodné zahrnúť aj sedadlo vodiča môže byť predmetom diskusie – kvôli orientácii miest vo vozidle by bolo určite dobré ho tam dať, no bude to aj tak miesto, ktoré si nikto nebude môcť rezervovať, preto pri riešení našej úlohy – v zjednodušenej podobe – ho môžeme vynechať. Majme na pamäti, že sa snažíme viesť žiakov k vlastnej formulácii úlohy tak, aby ju boli schopní sami (v zadanom čase) aj vyriešiť.

Náčrt riešenia robíme na tabuli (vyzve niektorého zo žiakov) a môže vyzeráť napríklad takto:



Podľa náčrtu by žiaci mali navrhnúť, že budeme potrebovať vedieť vykresliť sedadlo a steny autobusu:

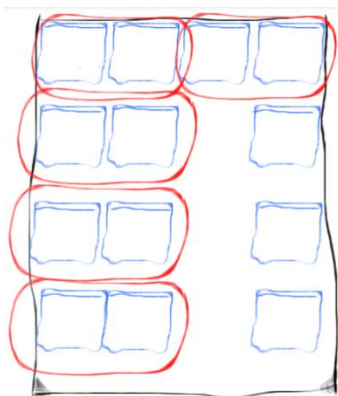
`funkcia sedadlo()`

`funkcia steny()`



Funkcie zatiaľ neprogramujeme, zameriavame sa na analytickú činnosť. Položíme žiakom otázku, ako by mala vyzeráť funkcia `sedadlo()` a funkcia `steny()` – žiaci slovne formulujú postupnosť grafických krokov. Nie je potrebné riešiť zatiaľ jednotlivé inštrukcie, ani ich parametre, ani iné technické detaily (napr. návrat do počiatočnej pozície, či zdvihnutia/položenia pera a pod.)

Aby sme urobili našu prácu efektívnejšou, zamerajme sa na vyhľadanie takých vzorov, pre ktoré by sme vedeli navrhnúť ďalšie funkcie, využívajúce buď funkciu `sedadlo()` alebo funkciu `steny()`. Žiaci by mohli navrhnúť vytvorenie funkcie `dvojsedadlo()` na vykreslenie dvojíc sedadiel, ktoré sú v každom rade aspoň raz (v poslednom rade dokonca dvakrát):

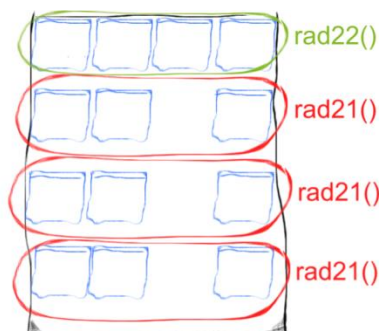


funkcia `dvojsedadlo()`



Môžeme žiakom poukázať na fakt, že táto funkcia môže byť pre nás užitočná aj v prípade, ak by sme v budúcnosti chceli navrhovať iný typ autobusov – napríklad vo väčšine diaľkových autobusov sú v radoch dvojsedadlá aj napravo, aj naľavo.

Využitím funkcie `dvojsedadlo()` a funkcie `sedadlo()` môžeme ďalej vytvoriť aj funkcie `rad21()` a `rad22()` na vykreslenie radu sedadiel s dvojsedadlom a jedným sedadlom, resp. dvoma dvojsedadlami:





Na záver vyzveme žiakov, aby vymenovali (príp. spísali na tabuľu) všetky príkazy jazyka Python, ktoré budú pri programovaní navrhnutých funkcií potrebovať – získame tak sumarizáciu a prehľad všetkých potrebných funkcií.

## PRECVIČOVANIE – SAMOSTATNÁ PRÁCA (CCA 27 MIN)

V tejto fáze žiaci pracujú s pracovným listom. Učiteľ v tejto fáze do priebehu hodiny nezasahuje, len individuálne monitoruje progres žiakov, a kontroluje ich riešenia. V prípade problémov alebo nejasností, objasní alebo preformuluje problém konkrétnemu žiakovi. Žiaci najprv riešia **úlohu 2** (programové riešenie úlohy 1) a **úlohu 3** (rozšírenie riešenia úlohy 2), ktoré by mali stihnúť na hodine všetci žiaci. Zdatnejší programátori môžu riešiť úlohu 4 podľa zadania učiteľa aj na hodine a úlohu 5 vyriešia formou domácej úlohy, nakoľko vyžaduje viac času na premýšľanie a analýzu vzorov. Žiaci si riešenia jednotlivých úloh ukladajú do samostatných súborov, ktoré na konci odovzdajú učiteľovi.

**Úloha 2** Naprogramujte navrhnuté funkcie (z úlohy 1). Vytvorte program **autobus1.py**, ktorý pomocou naprogramovaných funkcií vykreslí rozloženie miest v autobuse podľa úlohy 1.

**Riešenie:** Programové riešenia sa u jednotlivých žiakov môžu líšiť. Príklad riešenia je v súbore **autobus1.py**. V riešení naprogramujeme a využijeme funkcie, ktoré sme navrhli v predchádzajúcej úlohe.

```
import turtle

def sedadlo():
    pero.pendown()
    for i in range(4):
        pero.forward(30)
        pero.left(90)
    pero.penup()

def dvojsedadlo():
    for i in range(2):
        sedadlo()
        pero.forward(40)
    pero.backward(80)

def rad21():
    dvojsedadlo()
    pero.forward(120)
    sedadlo()
    pero.backward(120)

def rad22():
    for i in range(2):
        dvojsedadlo()
        pero.forward(80)
    pero.backward(160)

def sedadla():
```

```
# vykreslenie sedadiel v autobuse
for i in range(3):
    rad21()
    pero.left(90)
    pero.forward(40)
    pero.right(90)
rad22()
pero.left(90)
pero.backward(120)
pero.right(90)

def steny():
    pero.pendown()
    for i in range(2):
        pero.forward(150)
        pero.left(90)
        pero.forward(150)
        pero.left(90)

def autobus1():
    sedadla()
    steny()

tabula = turtle.Screen()
pero = turtle.Turtle()

autobus1()

tabula.mainloop()
```

**Poznámka:**

V tejto úlohe smerujeme žiakov k dobrej dekompozícii problému na podproblémy. Uvedené riešenie je len jedno z možných riešení. Žiaci môžu úlohu dekomponovať aj inak. V riešení sa nezaobráame farbami alebo výplňou. V tomto momente to nie je podstatné. Ak by sme začali riešiť aj tieto prvky, samotný kód sa môže neúmerne predĺžiť na úkor jeho prehľadnosti. Odporúčame žiakov viesť k riešeniam, pri ktorých sa grafické pero po vykreslení jednotlivých častí (napr. sedadlo, dvojsedadlo, rad12, rad22) vráti do štartovacej pozície. Celkové zostavenie programu tak bude jednoduchšie. lebo presne vieme, že vykreslenie jednotlivých častí nám pozíciu pera nezmení. Odporúčame, aby si žiaci jednotlivé vzdialenosti a dĺžky dokreslili do vytvoreného plánu.

**Úloha 3** Upravte program z predošlej úlohy tak, aby vykresľoval rozloženie miest v inom type autobusov (podľa obrázku vpravo) a uložte ho ako **autobus2.py**.



**Riešenie:** Programové riešenia sa u jednotlivých žiakov môžu líšiť, ale mali by využívať v maximálnej miere navrhnuté funkcie z predošlej úlohy, príp. ich drobné modifikácie. Príklad riešenia je v súbore **autobus2.py**.

Rozloženie sedadiel v takomto autobuse je mierne odlišné. Využiť však môžeme niektoré z funkcií z predchádzajúcej úlohy.

```
import turtle

def sedadlo():
    pero.pendown()
    for i in range(4):
        pero.forward(30)
        pero.left(90)
    pero.penup()

def dvojsedadlo():
    for i in range(2):
        sedadlo()
        pero.forward(40)
    pero.backward(80)

def rad22():
    for i in range(2):
        dvojsedadlo()
        pero.forward(120)
    pero.backward(240)

def rad5():
    for i in range(5):
        sedadlo()
        pero.forward(40)
    pero.backward(200)

def sedadla():
    for i in range(7):
        rad22()
        pero.left(90)
        pero.forward(40)
        pero.right(90)
    rad5()
    pero.left(90)
    pero.backward(280)
    pero.right(90)

def steny():
    pero.pendown()
    for i in range(2):
        pero.forward(190)
        pero.left(90)
        pero.forward(310)
        pero.left(90)

def autobus2():
    sedadla()
    steny()
```

```
tabula = turtle.Screen()
pero = turtle.Turtle()

autobus2()

tabula.mainloop()
```

### Úloha 4

Riešte  
podľa  
pokynov  
učiteľa

Doplňte k pripravenej funkcii `sestuholnik()` v súbore **geometria.py** ďalšie funkcie, ktoré funkciu `sestuholnik()` volajú, pomocou ktorých je možné vykresliť nasledovný ornament určený na potlač obrusov:



**Tvorivé rozšírenie úlohy:** Navrhňte pomocou Vašich funkcií nový, zaujímavý a netradičný vzor potlače na obris.

**Riešenie:**

*Programové riešenia sa u jednotlivých žiakov môžu líšiť. Odporúčame, aby žiaci previedli vhodnú dekompozíciu vzoru, napríklad:*

vzor	kvet	červený šesťuholník	šesťuholník
	retiazka	zelený šesťuholník	šesťuholník

Riešenie úlohy je uložené v súbore **geometria.py**.

```
import turtle

def sestuholnik():
    pero.pendown()
    for i in range(6):
        pero.forward(20)
        pero.left(60)
    pero.penup()

def cerveny_sestuholnik():
    pero.color('red')
    pero.begin_fill()
    sestuholnik()
    pero.end_fill()

def zeleny_sestuholnik():
    pero.color('green')
    pero.begin_fill()
    sestuholnik()
    pero.end_fill()

def kvet():
    for i in range(6):
```



```

    cerveny_sestuholnik()
    pero.forward(20)
    pero.right(60)

def retiazka():
    for i in range(3):
        zeleny_sestuholnik()
        pero.forward(40)

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.width(2)
pero.penup()
# pero.setpos(-500,0)

for i in range(5):
    kvet()
    pero.forward(70)
    retiazka()
    pero.forward(35)

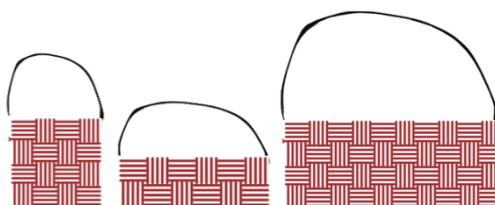
tabula.mainloop()

```

**Úloha 5**

Riešte  
podľa  
pokynov  
učiteľa

Navrhните vhodné funkcie pre vykresľovanie jednoduchého výpletového vzoru pre košíky. Funkcie (resp. ich kombinácie) by mali umožňovať vykresľovanie vzorov pre všetky tri výrobné typy košíkov (zobrazené na obrázkoch nižšie). Riešenie uložte do súboru **kosik.py**.



Riešenie:

Pri riešení je potrebné sa zamerať len na samotný vzor, rúčku košíka riešiť netreba. Odporúčame, aby žiaci previedli vhodnú dekompozíciu, napr.: rad – veľký vzor – malý vzor.

Programové riešenia sa u jednotlivých žiakov môžu líšiť. Správne riešenie by malo byť dostatočne flexibilné – zmenu vykresľovaného vzoru bude možné dosiahnuť len zmenou parametrov cyklov, žiadne iné zmeny v kóde by sa robiť nemali. Príklad riešenia je v súbore **kosik.py**.

```

import turtle

def maly_vzor():
    for i in range(5):
        pero.pendown()
        pero.forward(50)
        pero.penup()
        pero.forward(-50)
        pero.left(90)
        pero.forward(10)

```

```
        pero.right(90)

def velky_vzor():
    for i in range(4):
        maly_vzor()
        pero.forward(55)
        pero.right(90)
        pero.forward(5)

def rad():
    for i in range(5):          # počet stĺpcov vzorov v košíku
        velky_vzor()
        pero.forward(100)

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.color('brown')
pero.width(5)
pero.penup()
# pero.setpos(-300, 300)

for i in range(2):            # počet radov vzorov v košíku
    rad()
    pero.right(90)
    pero.forward(110)
    pero.right(90)

tabula.mainloop()
```

### ZHRNUTIE (CCA 3 MIN)

Vyzveme žiakov, aby sa vyjadrili, či mali pri samostatnej práci nejaké problémy s úlohami, čo bolo pre nich ťažké, resp. ktoré z ponúknutých programátorských problémov zvládli bez väčších komplikácií. Pokiaľ sa počas predošlej fázy vyskytli nejaké zaujímavé alebo netradičné nápady na vyriešenie niektorej z úloh, môžeme ich pomocou dataprojektora frontálne ukázať (napr. riešenie tvorivého rozšírenia úlohy 4).

## 06 VLASTNÉ FUNKCIE S PARAMETRAMI – KRESLIACE ÚLOHY

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>• pomocou postupnosti príkazov,</li> <li>• pomocou premenných.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>• vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>• používať premennú vo výrazoch,</li> <li>• používať príkazy korytnačej grafiky,</li> <li>• vytvárať a používať vlastné funkcie bez parametra,</li> <li>• používať cyklus s pevným počtom opakovaní (<code>for i in range(&lt;pocet&gt;)</code>).</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>• formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia,</li> <li>• identifikovať vstupné informácie zo zadania úlohy,</li> <li>• popisovať očakávané výstupy, výsledky, akcie.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>• aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou premenných:</b> <ul style="list-style-type: none"> <li>• zovšeobecniť riešenie tak, aby fungovalo nielen s konštantami.</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>• rozpoznávať opakujúce sa vzory.</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>• vyjadriť ideu daného návodu (objavovať a vlastnými slovami popisovať ideu zapísaného riešenia – ako program funguje, čo zápis realizuje pre rôzne vstupy).</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>• zovšeobecniť riešenie pomocou funkcií s parametrami,</li> <li>• popísať a dokumentovať navrhnuté riešenia.</li> </ul>	Koncepty informatického myslenia  <b>Algoritmy:</b> <ul style="list-style-type: none"> <li>• (ALG8) zapísať algoritmy v konkrétnom formálnom jazyku (zápis v programovacom jazyku).</li> </ul> <b>Hľadanie vzorov:</b> <ul style="list-style-type: none"> <li>• (VZO2) určiť rovnaké/podobné vlastnosti/pravidlá správania sa častí systému (vytváranie funkcií na základe identifikácie opakujúcich sa objektov).</li> </ul> <b>Abstrakcia:</b> <ul style="list-style-type: none"> <li>• (ABS2) z konkrétnych prípadov (inštancií) objektov abstrahovať pojmy/postupy (návrhy vlastných funkcií a parametrov).</li> </ul>
<b>Riešený didaktický problém</b>	
<p>Žiaci pri programovaní spočiatku využívajú postupy špecificky riešiace daný problém s vopred definovanými konštantnými hodnotami. Programový kód tak obsahuje podobné časti (líšiace sa len v konkrétnych hodnotách), ktoré však nie sú všeobecne využiteľné. Vďaka abstrakcii a porovnávaniu vlastností objektov je možné navrhovať funkcie s vhodnými parametrami a s menším počtom inštrukcií tak dosiahnuť väčšiu variabilitu programov.</p> <p>Ďalšou chybou je, že žiaci parameter použijú v hlavičke pri definovaní vlastnej funkcie, ale v tele funkcie</p>	

ho už nepoužívajú. To znamená, že v tele funkcie zadávajú pevné, konštantné hodnoty. Vytvorila tak niekoľko veľmi podobných funkcií namiesto jednej, kde by postačovalo pri jej volaní meniť iba parameter.

Žiaci často definíciu funkcie nejako vytvoria, ale nie sú schopní popísať čo funkcia presne robí a aké očakáva vstupy. Preto v tejto metodike predstavujeme aj dokumentačné reťazce funkcií a žiaci by sa ich mali naučiť postupne vytvárať.

<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• frontálna, individuálna a skupinová forma (dvojice žiakov).</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/06/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/06/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.	



## Úvod

Toto je 6. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Tematicky nadväzuje na metodiku 3, v ktorej sa žiaci naučili vytvárať a používať vlastné funkcie (bez parametra). V tejto metodike sa žiaci cez abstrakciu dostávajú k funkciám s parametrami (jedným aj viacerými) a v rámci nasledujúcich hodín budú koncept precvičovať a rozširovať na náročnejších funkciách pre kreslenie aj výpočet. V prípade potreby je možné časovú dotáciu pre metodiky zvýšiť na dve vyučovacie hodiny, čo umožní žiakom lepšie osvojenie aj precvičenie nových poznatkov a zručností. Je dôležité žiakov upozorniť, že v čom spočíva práca dobrého programátora – nielen v písaní samotného zdrojového kódu, ale taktiež v systematickom, efektívnom, univerzálnom, abstraktnom uvažovaní, preto aj týmto analytickým aktivitám je potrebné venovať pri výučbe zodpovedajúci priestor – v tejto metodike (a aj v nasledujúcich) sa postupne žiaci učia abstraktne myslieť, prezentovať svoje myšlienky pred ostatnými žiakmi a taktiež spoločne hľadať riešenia aj bez toho, aby tvorili zdrojový kód.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou výučby je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – rozhovor so žiakmi – vyvodenie myšlienky parametrizácie funkcií a úlohy na spoločné riešenie.
- **Skúmanie (12 minút)** – práca vo dvojiciach s pracovným listom (úlohy 1 až 4).
- **Vysvetlenie (5 minút)** – žiacke vysvetlenie nových poznatkov.
- **Rozpracovanie (13 minút)** – samostatné programovanie náročnejších úloh (úlohy 5 až 8 z pracovného listu).
- **Hodnotenie (5 minút)** – kontrola žiackych prác, vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 5 MIN)

Hodinu začneme frontálnym motivačným rozhovorom – žiakom pomocou dataprojektora premietneme sprievodnú prezentáciu, kde im pomocou snímok 2 až 6 predstavíme myšlienku parametrizácie funkcie. Začneme s konkrétnym príkladom objektu (auto, vlak alebo hamburger), pre ktorý by sme vedeli navrhnúť vhodnú funkciu (`auto()`, `vlak()` alebo `hamburger()`). Avšak, pokiaľ by sme chceli niečo na našom objekte zmeniť (farbu, počet vagónov, veľkosť), vyžadovalo by si to zmenu v kóde – naša funkcia by nebola dostatočne univerzálna. Preto by bolo vhodnejšie modifikovať

naše funkcie tak, aby sme vedeli vykresľovať rôzne podobné objekty len vďaka tomu, že požadovanú vlastnosť (parameter) uvedieme v zátvorke, napr. pri volaní `auto('červené')` by sme vykreslili červené auto, pri volaní `vlak(4)` by sme vykreslili vlak so 4 vagónmi a pri volaní `hamburger('XXL')` by sme vykreslili najväčší hamburger. Preto by sme potrebovali definovať funkcie aj s parametrom `auto(farba)`, `vlak(dĺžka)` alebo `hamburger(veľkosť)`.



Ďalej vyzveme žiakov k spoločnému zodpovedaniu otázok z prezentácie (snímky 5 a 6):

1. Navrhňte ďalšie príklady skupín objektov, ktoré sa líšia jedným parametrom! (napr. dieťa (výška), kytica (počet\_kvetov), kvet (farba) a pod.)
2. Navrhňte niektoré tovary z obchodu, kde by sa mohli zadať parametre, ktoré by ich rozlišovali! (napr. tričko (veľkosť, farba), žuvačky (značka, príchuť), vajíčka (veľkosť, počet, typ\_chovu) a pod.)
3. Ktoré z príkazov Pythonu používajú parameter? (`for i in range(10)`, `dot(40, 'orange')`, `forward(40)`, `left(60)`, `width(3)`)

Pri druhej aj tretej otázke môžeme žiakom zároveň ukázať, že existujú aj funkcie, ktoré majú viacero parametrov. Na základe uvedenej aktivity predstavíme žiakom cieľ hodiny:

- Vytvárať programy riešiace konkrétne problémy využitím vlastných funkcií s parametrami.

## SKÚMANIE (CCA 12 MIN)

V tejto fáze žiaci pracujú vo dvojiciach s pracovným listom (úlohy 1 až 5) a pracovným súborom **sestuholnik.py**. Keďže vždy je uvedený názov súboru, s ktorým žiaci pracujú, nemusia žiaci všetky vytvorené kódy prepisovať do pracovného listu (resp. tak môžu spraviť v rámci domácej prípravy). V pripravených programových kódoch uvádzame aj dokumentačné reťazce funkcií a odporúčame, aby ich vytvárali aj žiaci (minimálne od úlohy 5).

Cieľom úloh a žiackeho bádania je sledovať podobnosti v kódach funkcií na vykreslenie základných geometrických útvarov a navrhnúť ich zovšeobecnenie do funkcie `uholnik(n)` na vykreslenie požadovaného n-uholníka. Učiteľ do práce nezasahuje, monitoruje aktivitu žiakov a v prípade potreby len usmerní ich činnosť.

### Poznámka

Vývojové prostredie PyCharm dokáže časť dokumentačného reťazca funkcie vygenerovať automaticky. Stačí ak pod hlavičku funkcie napíšete reťazec začínajúci tromi apostrofmi a stlačíte kláves Enter. Momentálne nepotrebné časti dokumentačného reťazca (napr. `return`) môžeme vymazať.

Dokumentáciu k použitej funkcii zobrazíte, ak je kurzor nastavený na jej mene klávesovou skratkou Ctrl+Q (View – Quick Documentation).

**Úloha 1** V súbore **sestuholnik.py** je program na kreslenie pravidelných šesťuholníkov. Vyznačte v jeho kóde v pracovnom liste **farebne** IBA tie parametre, ktoré spôsobia zmenu veľkosti šesťuholníka:

```
import turtle

def sestuholnik():
    pero.pendown()
    for i in range(6):
        pero.forward(20)
        pero.left(360 / 6)
    pero.penup()

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.width(2)

sestuholnik()

tabula.mainloop()
```

Riešenie:

```
import turtle

def sestuholnik():
    pero.pendown()
    for i in range(6):
        pero.forward(20)
        pero.left(360 / 6)
    pero.penup()

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.width(2)

sestuholnik()

tabula.mainloop()
```

**Úloha 2** Doplňte do súboru **sestuholnik.py** podobným spôsobom programy na kreslenie rovnostranných trojuholníkov a štvorcov (t. j. pravidelných štvoruholníkov). Zapište svoje kódy:

```
def trojuholnik():
```

```
def stvoruholnik():
```

Riešenie:

```
def trojuholnik():
    pero.pendown()
    for i in range(3):
        pero.forward(20)
        pero.left(360 / 3)
    pero.penup()
```

```
def stvoruholnik():
    pero.pendown()
    for i in range(4):
        pero.forward(20)
        pero.left(360 / 4)
    pero.penup()
```

**Úloha 3** Porovnajte kódy funkcií pre kreslenie trojuholníka, štvoruholníka a šesťuholníka a doplňte podľa nich tabuľku:

Objekt (pravidelný z hľadiska veľkosti uhlov)	Počet uhlov	Uhol otočenia korytnačky (= veľkosť vonkajších uhlov objektu)
trojuholník		
štvoruholník		
päťuholník		
šesťuholník		
sedemuholník		
n-uholník		

Riešenie:

Objekt (pravidelný z hľadiska veľkosti uhlov)	Počet uhlov	Uhol otočenia korytnačky (= veľkosť vonkajších uhlov objektu)
trojuholník	<b>3</b>	<b>120</b> ( $3 \times 120^\circ = 360^\circ$ )
štvoruholník	<b>4</b>	<b>90</b> ( $4 \times 90^\circ = 360^\circ$ )
päťuholník	<b>5</b>	<b>72</b> ( $5 \times 72^\circ = 360^\circ$ )
šesťuholník	<b>6</b>	<b>60</b> ( $6 \times 60^\circ = 360^\circ$ )
sedemuholník	<b>7</b>	<b>360 / 7</b>
n-uholník	<b>n</b>	<b>360 / n</b>

**Úloha 4** Porovnajtie kódy funkcií pre kreslenie trojuholníka, štvoruholníka a šesťuholníka. V ktorých častiach sa líšia?

Navrhните, ako by mala vyzerat' funkcia `uholnik(n)`, kde `n` je parameter vyjadrujúci počet uhlov (napr. `uholnik(3)` nakreslí trojuholník, `uholnik(6)` nakreslí šesťuholník a pod.):

```
def uholnik(n):
```

Overte správnosť Vášho programu vykreslením trojuholníka a šesťuholníka. Funguje Váš program správne?

**ÁNO – NIE**

Riešenie:

```
def uholnik(n):  
    pero.pendown()  
    for i in range(n):  
        pero.forward(20)  
        pero.left(360 / n)  
    pero.penup()
```

## VYSVETLENIE (CCA 5 MIN)

Na základe kontroly riešení úloh z pracovných listov žiaci vysvetlia, ako sa im podarilo navrhnúť funkciu `uholnik(n)` z poslednej úlohy. Pokiaľ sa to niektorým žiakom nepodarilo, požiadame ostatných, aby vysvetlili svoj postup riešenia. Zdôrazníme, že `n` je premenná (bola už v predošlých hodinách spomínaná) a jej obsah sa mení podľa toho, akú hodnotu zadáme pri volaní funkcie. V zdrojových kódach sme zámerne ponechali v príkaze otáčania kresliaceho pera výraz `360 / n`. Ak by sme hodnotu vo funkciách vyčíslovali, závislosť veľkosti uhla otáčania od počtu strán by bola menej viditeľná.

Diskutujme so žiakmi, či funkcie môžu mať aj viac parametrov a či by vedeli identifikovať miesto vo vytvorených funkciách, úpravou ktorého by funkcia kreslila uholníky líšiace sa nielen počtom strán, ale aj ich veľkosťou.

### Poznámka

Uhol otáčania sme v tejto fáze ponechali v tvare opakovaného výpočtu v cykle. Dôvodom je, aby vzor výpočtu veľkosti uhla bol priamo viditeľný a žiaci ho mohli priamo použiť. Neskôr, keď žiaci nadobudnú viac skúseností, je možné tento

výpočet presunúť pred cyklus, hodnotu uhla vhodne pomenovať premennou a premennú použiť v cykle pri otáčaní. Rovnaký prístup neskôr aplikujeme aj pre iné, opakovane vypočítavané hodnoty.

## ROZPRACOVANIE (CCA 13 MIN)

Požiadajte žiakov, aby navrhli úpravu pre svoju funkciu `stvoruholnik()` tak, aby vedela vykresľovať rôzne veľké štvorce.

```
def stvoruholnik(strana):  
    pero.pendown()  
    for i in range(4):  
        pero.forward(strana)  
        pero.left(360 / 4)  
    pero.penup()
```

Je potrebné žiakov upozorniť, ak si to neuvedomili a volanie funkcie `stvoruholnik()` neupravili, že zmenu je potrebné vykonať aj pri volaní funkcie a doplniť volanie o parameter dĺžky strany:

```
stvoruholnik(50)
```

Premenná `strana` vo funkcii `stvoruholnik()` nadobudne hodnotu 50. Všade, kde sa vo funkcii vyskytne, sa použije táto hodnota.

Požiadajte žiakov, aby podobne upravili aj funkciu `trojuholnik()` a funkciu `sestuholnik()`:

```
def trojuholnik(strana):  
    pero.pendown()  
    for i in range(3):  
        pero.forward(strana)  
        pero.left(360 / 3)  
    pero.penup()  
  
def sestuholnik(strana):  
    pero.pendown()  
    for i in range(6):  
        pero.forward(strana)  
        pero.left(360 / 6)  
    pero.penup()
```

Žiaci pokračujú v zovšeobecňovaní riešenia vykresľovania n-uholníkov. V predchádzajúcej časti vytvorili samostatné funkcie pre kreslenie uholníkov s rôznym počtom strán, pričom funkcie na základe hodnoty parametra (`strana`) kreslili požadované uholníky s rôznou veľkosťou. Nasledujúcou úlohou je vytvoriť jednu všeobecnú funkciu, ktorá bude kresliť rôzne veľké uholníky s rôznym počtom strán.

**Úloha 5** Upravte funkciu `uholnik(n)` tak, aby sme pomocou nej dokázali kresliť rôzne veľké uholníky s rôznym počtom strán.

Pre funkciu vytvorte dokumentačný reťazec.

Riešenie:

```
def uholnik(n, strana):
    pero.pendown()
    for i in range(n):
        pero.forward(strana)
        pero.left(360 / n)
    pero.penup()
```

V nasledujúcej časti hodiny žiaci riešia samostatne úlohy 6 až 8 z pracovného listu. Žiakov upozorníme, že v úlohe 7 nie je potrebné programovať riešenia, úlohu stačí riešiť len v pracovnom liste.

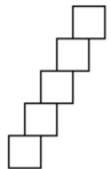
**Úloha 6** Vytvorte nový súbor **stvorec.py**.

Časť b) riešte podľa pokynov učiteľa

a) Vytvorte v ňom funkciu `veza(pocet, strana)` na vykreslenie veže z požadovaného počtu štvorcov zadanej veľkosti. TIP: Využite funkciu `stvoruholnik(velkost)`, ktorú ste v predošlých úlohách navrhli. Pre funkcie vytvorte dokumentačné reťazce.



b) Dopĺňte do programu aj funkciu `schody(pocet, strana)` na vykreslenie schodiska z požadovaného počtu štvorcov zadanej veľkosti: Pre funkcie vytvorte dokumentačné reťazce.



Riešenie:

```
import turtle

def stvorec(strana):
    pero.pendown()
    for i in range(4):
        pero.forward(strana)
        pero.right(360 / 4)
    pero.penup()

def veza(pocet, strana):
    for i in range(pocet):
        stvorec(strana)
        pero.forward(strana)
    pero.backward(pocet * strana)

def schody(pocet, strana):
    for i in range(pocet):
        stvorec(strana)
        pero.forward(strana)
        pero.right(90)
        pero.forward(strana / 2)
        pero.left(90)
    pero.backward(pocet * strana)
    pero.left(90)
```

```

pero.forward(pocet * strana / 2)
pero.right(90)

tabula = turtle.Screen()
pero = turtle.Turtle()
pero.left(90)

veza(5, 20)
# schody(5, 20)

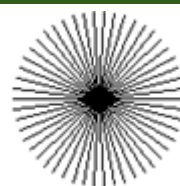
tabula.mainloop()

```

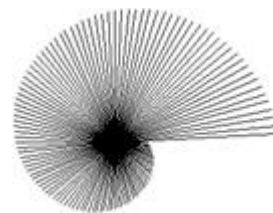
**Úloha 7**

Časť b)  
riešte  
podľa  
pokynov  
učiteľa

a) Vytvorte program **luce.py** a v ňom funkciu `kresli_luce(dlзка, pocet)` tak, aby vedela kresliť rôzne slnká s určeným počtom lúčov a určenou dĺžkou, teda napr. :



b) Pomocou neznámej funkcie `divne_luce()` sme nakreslili nasledujúci obrázok:



V čom sa naša neznáma funkcia líši od Vašej funkcie `luce()`?

Riešenie:

a)

```

def luce(dlзка, pocet):
    for i in range(pocet):
        pero.forward(dlзка)
        pero.backward(dlзка)
        pero.right(360 / pocet)

```

b) Funkcia sa líši tým, že jednotlivé lúče sa postupne predlžujú (resp. sa postupne skracujú).

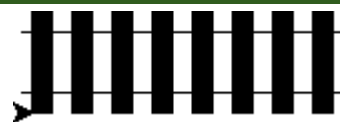
**Úloha 8**

Riešte  
podľa  
pokynov  
učiteľa

Vytvorte program **plot.py** na vykreslenie plotu. Plot je zložený zo zadaného počtu tyčiek zo zadanou šírkou a výškou.

Spojky spájajúce tyčky plotu sú vo výške 1/5 a 4/5 výšky tyčky.

Pre funkcie vytvorte dokumentačné reťazce.



Riešenie:

```

import turtle

```



```
def tycka(sirka, vyska):
    pero.pendown()
    pero.begin_fill()
    for i in range(2):
        pero.forward(sirka)
        pero.left(90)
        pero.forward(vyska)
        pero.left(90)
    pero.end_fill()
    pero.penup()

def spojka(pocet, sirka):
    pero.pendown()
    pero.backward(sirka / 2)
    pero.forward(2 * pocet * sirka)
    pero.backward(2 * pocet * sirka - sirka / 2)
    pero.penup()

def plot(pocet, sirka, vyska):
    for i in range(pocet):
        tycka(sirka, vyska)
        pero.forward(2 * sirka)

    pero.backward(2 * pocet * sirka)
    pero.left(90)
    pero.forward(vyska * 0.2)
    pero.right(90)
    spojka(pocet, sirka)
    pero.left(90)
    pero.forward(vyska * 0.6)
    pero.right(90)
    spojka(pocet, sirka)
    pero.left(90)
    pero.backward(vyska * 0.8)
    pero.right(90)
    pero.pendown()

tabula = turtle.Screen()
pero = turtle.Turtle()

plot(8, 10, 50) # vykreslenie plotu s 8 tyčkami so šírkou 10 a výškou 50

tabula.mainloop()
```

## VYHODNOTENIE (CCA 5 MIN)

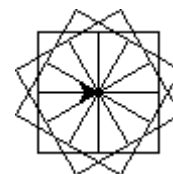
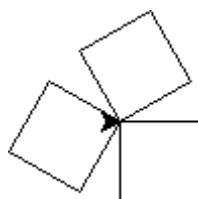
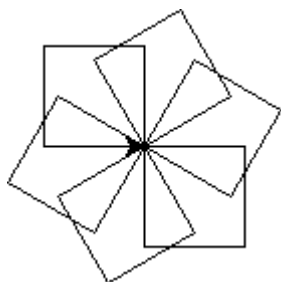
Vyzveme žiakov, aby sa vyjadrili, či mali pri samostatnej práci nejaké problémy s úlohami, čo bolo pre nich ťažké, resp. ktoré z ponúknutých programátorských problémov zvládli bez väčších komplikácií.

Následne požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal a nie klasifikácia známku. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než

umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí (môžeme na vykresľovanie použiť riešenie zo súboru **otocene\_stvorc.py**). Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

### Sebahodnotiaci test

Nájdite chyby v nasledujúcom programe, aby sme pomocou neho potom mohli vykresľovať zadaný počet otočených štvorcov zadanej veľkosti, napr.:



```
def stvorec(strana):
    for i in range(4):
        pero.forward(50)
        pero.right(90)

def vzor(pocet):
    for i in range(pocet):
        stvorec()
        pero.right(90)

vzor()
```

### Riešenie:

```
def stvorec(strana):
    for i in range(4):
        pero.forward(strana)
        pero.right(90)

def vzor(pocet, strana):
    for i in range(pocet):
        stvorec(strana)
        pero.right(360 / pocet)

vzor(5, 50) # vykreslenie vzoru s 5 štvorcami dĺžky 50
```

- parameter `strana` je potrebné vo funkcii `stvorec()` použiť,
- v definícii funkcie `vzor()` je potrebné uviesť aj druhý parameter `strana`,
- pri volaní funkcie `stvorec()` je potrebné zadať aj dĺžku strany,

- `90` → `360/pocet`  
pevné otočenie o  $90^\circ$  umožní vykresľovať len štvorce otočené o pevný uhol  $90^\circ$ ,
- `vzor()` → `vzor(5, 50)`  
chýba parameter, napríklad pre vykreslenie 5 štvorcov s dĺžkou strany 50.

## 07 VLASTNÉ FUNKCIE S PARAMETRAMI A VÝSTUPOM – VÝPOČTOVÉ ÚLOHY

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou premenných.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>používať premennú vo výrazoch,</li> <li>vytvárať a používať vlastné funkcie s parametrami.</li> </ul>	
<b>Ciele</b>	
<i>Žiakom osvojované vedomosti a zručnosti</i>	<i>Žiakom rozvíjané spôsobilosti</i>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>identifikovať problém, ktorý sa bude riešiť algoritmicky,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>používať matematické výrazy pri vyjadrovaní vzťahov a podmienok.</li> </ul> <b>Pomocou premenných:</b> <ul style="list-style-type: none"> <li>identifikovať zo zadania úlohy, ktoré údaje musia byť zapamätané, resp. sa menia (a teda vyžadujú použitie premenných),</li> <li>riešiť problémy, v ktorých si treba zapamätať a neskôr použiť zapamätané hodnoty vo výrazoch,</li> <li>zovšeobecniť riešenie tak, aby fungovalo nielen s konštantami.</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>vytvoriť program využitím vlastných funkcií pre výpočet s viacerými parametrami s návratovou hodnotou,</li> <li>vytvárať a používať dokumentačné reťazce.</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG1) využitím logických zdôvodnení predpokladať správanie sa algoritmov (realizácia matematických výpočtov),</li> <li>(LOG5) logicky zdôvodniť rozdelenie problému na menšie časti (návrhy vlastných čiastkových funkcií a parametrov).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG8) zapísať algoritmy v konkrétnom formálnom jazyku (zápis v programovacom jazyku).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia - lineárne rozdeliť problémy/procesy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (návrhy vlastných čiastkových funkcií a parametrov).</li> </ul> Abstrakcia: <ul style="list-style-type: none"> <li>(ABS2) z konkrétnych prípadov (inštancií) problémov abstrahovať vzťahy, (opakovaný výpočet objemu kocky/guli pre rôzne dĺžky strán/polomerov).</li> </ul>
<b>Riešený didaktický problém</b>	
Využitie funkcií pre výpočet umožňuje dekomponovať numerické riešenie problému na menšie, opakovane využiteľné funkcie. Pri dekompozícii a následnom definovaní zodpovedajúcich funkcií tieto často	

pracujú s globálnymi premennými a výsledok výpočtu vypisujú. V metodikách vedíme žiakov k tomu, aby všetky potrebné hodnoty, s ktorými funkcia pracuje, dostala v parametroch a výsledok svojho výpočtu vrátila ako návratovú hodnotu. Funkcie sa takto stávajú univerzálnejšími. Nie sú závislé na kontexte, v ktorom sa použijú (nespoliehajú sa na globálne premenné) a dajú sa využiť ako súčasť komplexnejšieho riešenia (ich návratová hodnota sa využije v nasledujúcich výrazoch).

<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• frontálna, individuálna a skupinová forma (dvojice žiakov)</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/07/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/07/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.	

---

## Úvod

Toto je 7. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Tematicky nadväzuje na metodiku 6, v ktorej sa žiaci naučili vytvárať a používať vlastné funkcie s parametrom. V tejto metodike sa žiaci dostávajú k funkciám s parametrami pre výpočet a v rámci nasledujúcich hodín budú tento koncept rozširovať. Prechádzame od „kresliacich“ funkcií k funkciám pre výpočet. Keďže výsledkom funkcií už nie je grafický výstup, ale hodnota, zavádzame funkcie s výstupom – návratovou hodnotou.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou výučby je sebahodnotiaci test.

---

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – rozhovor so žiakmi - vyvodenie myšlienky funkcií pre výpočet.
- **Skúmanie (5 minút)** – práca vo dvojiciach s pracovným listom.
- **Vysvetlenie (7 minút)** – žiacke vysvetlenie toho, čo zistili v predchádzajúcej časti.
- **Rozpracovanie (18 minút)** – samostatné programovanie náročnejších úloh (úlohy 4 až 8 z pracovného listu).
- **Vyhodnotenie (5 minúty)** – kontrola žiackych prác, vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 5 MIN)

Hodinu začneme frontálnym motivačným rozhovorom. Žiakom pripomenieme, že dosiaľ sme sa venovali kresleniu a vytváraníu vlastných funkcií s parametrami. Pomocou zmeny hodnôt parametrov funkcie kreslili obrázky, ktoré sa líšili veľkosťou alebo počtom nejakých častí. Oveľa častejšie, a to aj v reálnom živote, sa však stretávame s funkciami, ktoré neposkytujú grafický výstup, ale realizujú rôzne výpočty.

Vyzveme žiakov, aby v nasledujúcej úlohe uviedli aj ďalšie príklady funkcií a diskutovali o nich.

**Úloha 1** V reálnom živote využívame množstvo funkcií, ktoré realizujú rôzne výpočty a výsledky ktorých využívame pre plánovanie našich ďalších činností. Napr.:

vstupné parametre		výstupná hodnota
hodinová mzda počet odpracovaných hodín za mesiac daň	→	výška výplaty
veľkosť nádrže auta priemerná spotreba	→	dojazd auta
plocha podlahy výdatnosť plechovky farby cena plechovky farby	→	náklady na vymaľovanie
výška vkladu úroková miera	→	výška budúceho zárobku
dĺžky strán trojuholníka	→	obvod trojuholníka

Diskutujte o funkciách, ktoré sa vyskytujú v našom živote.

Aké sú vstupné hodnoty týchto funkcií?

Čo je výsledkom týchto funkcií?

Riešenie:

Vstupné hodnoty sú nejaké čísla – hodnoty nejakých veličín, napr. cena, množstvo peňazí, objem.

Výsledkom je nejaké vypočítané číslo – hodnota novej veličiny, napr. dojazd, náklady na renováciu, obvod útvaru.

V nadväznosti na riešenie predchádzajúcej úlohy predstavíme žiakom cieľ hodiny: implementovať vlastné funkcie s parametrami a výstupom pre výpočet pri riešení problémov.

## SKÚMANIE (CCA 5 MIN)

Žiaci pracujú vo dvojiciach s pracovným listom. Učiteľ do práce nezasahuje, monitoruje aktivitu žiakov a v prípade potreby len usmerní ich činnosť.

**Úloha 2** Otvorte súbor **kocka.py**, otestujte ho a preskúmajte jeho zdrojový kód.

Čo je výsledkom behu programu <b>kocka.py</b> ?	
V ktorej premennej v hlavnom programe je uložená dĺžka hrany kocky?	
Aká je dĺžka hrany kocky? Zmeňte ju na 50 a spustite program.	
Vo funkcii <code>objem_kocky()</code> nájdite neznámy príkaz: Na čo slúži?	
Nájdite príkaz, ktorým sa vypíše informácia o vypočítanom objeme:	
Vytvorte na vyznačenom mieste v programe funkciu <code>povrch_kocky()</code> a prepíšte si jej definíciu:	<pre>def povrch_kocky( ): :</pre>
Doplňte v programe na vhodné miesto aj výpočet a výpis vypočítaného povrchu kocky. Program otestujte a vyskúšajte zmeniť aj veľkosť hrany kocky.	

*Riešenie:*

Čo je výsledkom behu programu <b>kocka.py</b> ?	objem kocky, 1000
V ktorej premennej v hlavnom programe je uložená dĺžka hrany kocky?	hrana
Aká je dĺžka hrany kocky? Zmeňte ju na 50 a spustite program.	10
Vo funkcii <code>objem_kocky()</code> nájdite neznámy príkaz: Na čo slúži?	<code>return</code> vracia vypočítanú hodnotu
Nájdite príkaz, ktorým sa vypíše informácia o vypočítanom objeme:	<pre>print('Objem kocky je', objem)</pre>
Vytvorte na vyznačenom mieste v programe funkciu <code>povrch_kocky()</code> a prepíšte si jej definíciu:	<pre>def povrch_kocky(a):     s = 6 * a ** 2     return s</pre>
Doplňte v programe na vhodné miesto aj výpočet a výpis vypočítaného povrchu kocky. Program otestujte a vyskúšajte zmeniť aj veľkosť hrany kocky.	<pre>povrch = povrch_kocky(hrana) print('Povrch kocky je', povrch)</pre>

## VYSVETLENIE (CCA 7 MIN)

V rámci vzájomnej diskusie požiadame žiakov, aby vysvetlili nové prvky (príkaz `return`) a rozdielny spôsob použitia funkcií – spracovanie jej návratovej hodnoty. V tomto prípade sme si návratovou hodnotu zapamätali – pomenovali menom (`objem`, resp. `povrch`) a následne ju vypísali.



Na záver tejto časti zhrnieme nové poznatky a zdôrazníme, že na rozdiel od funkcií pre kreslenie (kde výstupom funkcie bol vykreslený obrázok) pri funkciách na výpočet funkcie spravidla vracajú vypočítanú hodnotu pomocou príkazu `return`. Túto hodnotu je v mieste volania funkcie potrebné uložiť do vhodne pomenovanej premennej pre použitie v ďalších výpočtoch alebo častiach programu.

## ROZPRACOVANIE (CCA 18 MIN)

**Úloha 3** Sochár má v úmysle vytvoriť dielo, v ktorom použije niekoľko dutých kociek. Dutú kocku vytvorí tak, že do formy v tvare kocky vloží menšiu kocku a priestor medzi dvoma kockami zaleje betónom. Pomôžte sochárovi vypočítať, koľko  $\text{m}^3$  betónu bude potrebovať na dutú kocku a definujte funkciu `objem_dutej_kocky()` na výpočet množstva betónu v dutej kocke.

Pre definovanú funkciu vytvorte dokumentačný reťazec.

Koľko  $\text{m}^3$  bude sochár potrebovať, ak väčšia kocka má hranu dĺžky 1,125 m a menšia kocka hranu dĺžky 0,95 m?

Riešenie realizujte v súbore **kocka.py**.

Premyslite si, či a ako sa sa dajú využiť vami definované funkcie.

Riešenie:

```
def objem_dutej_kocky(a_vacsia, a_mensia):  
    v_vacsej_kocky = objem_kocky(a_vacsia)  
    v_mensej_kocky = objem_kocky(a_mensia)  
    v_dutej_kocky = v_vacsej_kocky - v_mensej_kocky  
    return v_dutej_kocky  
  
hrana1 = 1.125  
hrana2 = 0.95  
objem_betonu = objem_dutej_kocky(hrana1, hrana2)  
print('Potrebný objem betónu je', objem_betonu)
```

V tejto úlohe by žiaci mali využiť už definovanú funkciu `objem_kocky()`. Keďže výpočet objemu priestoru medzi kockami sa realizuje ako rozdiel objemov dvoch kociek, funkcia `objem_kocky()` sa použije pri každom výpočte dvakrát. Cieľom je ukázať žiakom, že funkcie definujeme len raz, ale použiť ich môžeme opakovane niekoľko krát. Tento princíp sa ešte viac zvýrazní v nasledujúcej úlohe.

**Úloha 4** Sochár si pripravil niekoľko foriem pre svoje „kockové“ súsošie, Koľko  $\text{m}^3$  betónu si musí objednať, ak chce vytvoriť duté kocky nasledovných rozmerov? Riešenie realizujte v súbore **kocka.py**.

	hrana väčšej kocky	hrana menšej kocky
kocka 1	1,125	0,95
kocka2	2,95	2,7
kocka3	0,75	0,5

Riešenie:

```
objem_kocky1 = objem_dutej_kocky(1.125, 0.95)
```

```
objem_kocky2 = objem_dutej_kocky(2.95, 2.7)
objem_kocky3 = objem_dutej_kocky(0.75, 0.5)
objem_celkom = objem_kocky1 + objem_kocky2 + objem_kocky3
print('Potrebný objem betónu na všetky kocky je', objem_celkom)
```

Žiaci budú mať možno tendenciu pomenovávať každý z rozmerov. Riešenie môžeme zjednodušiť tak, že hodnoty dĺžok zadáme ako konkrétne hodnoty pri volaní funkcie. Podobne sme to robili pri kreslení pomocou korytnačej grafiky, kde sme pri volaní funkcií kresliaceho pera zadávali konkrétne hodnoty. V tomto prípade nie je potrebné vytvárať novú funkciu typu `objem_betonu_susosia()`. Počet kociek v súsoší môže byť variabilný a žiaci zatiaľ nepoznajú spôsob, ako niekoľko hodnôt „zabaliť“ do nejakej dátovej štruktúry (napr. do zoznamu).

**Poznámka:**

Zatiaľ vytvárame pomerne jednoduché funkcie, ktorých telo tvorí len malý počet riadkov. Žiaci môžu namietajú, že pre „pár riadkov“ sa neoplatí definovať samostatné funkcie. Argumentom proti sú fakty, že výsledný kód je prehľadnejší, zrozumiteľnejší a menej náchylný na chyby (funkciu definujeme len raz, ale voláme ju opakovane).

Žiaci pokračujú samostatným riešením nasledujúcich úloh z pracovného listu.

**Úloha 5** Index telesnej hmotnosti (angl. Body Mass Index – BMI) patrí medzi najviac používané metódy merania obezity.

Riešte

Počíta sa ako hmotnosť v kilogramoch delená druhou mocninou výšky v metroch.

podľa

Vytvorte funkciu `pocitaj_bmi()` pre výpočet hodnoty indexu BMI. Riešenie uložte do súboru **zdravie.py**.

pokynov

Riešenie:

učiteľa

```
def pocitaj_bmi(m, v):
    vysledok = m / v ** 2
    return vysledok

hmotnost = 95.0
vyska = 1.90
bmi = pocitaj_bmi(hmotnost, vyska)
print('Vypočítané BMI je', bmi)
```

**Poznámka:**

Žiaci majú tendenciu používať rovnaké identifikátory pre rôzne objekty. Keďže každý identifikátor použijú v inom kontexte (mennom priestore) väčšinou to nespôsobí chybu. Napriek tomu odporúčame viesť žiakov k tomu, aby tak nerobili a volili rôzne názvy identifikátorov. Extrémnym príkladom môže byť nasledovné neprehľadné, i keď funkčné riešenie:

```
def bmi(hmotnost, vyska):
    bmi = hmotnost / vyska ** 2
    return bmi
```

```
hmotnost = 95
vyska = 1.90
```

```
bmi = bmi(hmotnost, vyska)
print('Vypočítané BMI je', bmi)
```

**Úloha 6** Akú hmotnosť má snehuliak?

**Riešte podľa pokynov učiteľa** Vytvorte funkciu `hmotnost_snehuliaka()`, ktorá pre zadané priemery troch gúl, z ktorých je snehuliak postavený, vráti jeho hmotnosť. Riešenie uložte do súboru **snehuliak.py**.

**Pomôcka 1:** Môžete predpokladať, že 1 m<sup>3</sup> stlačeného snehu v guľi má hmotnosť asi 300 kg.

**Pomôcka 2:** Objem gule  $V$  s polomerom  $r$  vypočítame podľa vzorca:

$$V = \frac{4}{3}\pi r^3$$

**Pomôcka 3:** Niektoré výpočty budete robiť opakovane. Premyslite si, pre ktoré časti výpočtu je výhodné definovať samostatné funkcie.

**Riešenie:**

```
def objem_gule(priemer):
    objem = 4 / 3 * 3.14 * (priemer / 2) ** 3
    return objem

def hmotnost_snehovej_gule(priemer):
    hustota_snehu = 300
    objem_snehu = objem_gule(priemer)
    hmotnost = hustota_snehu * objem_snehu
    return hmotnost

def hmotnost_snehuliaka(priemer1, priemer2, priemer3):
    hmotnost1 = hmotnost_snehovej_gule(priemer1)
    hmotnost2 = hmotnost_snehovej_gule(priemer2)
    hmotnost3 = hmotnost_snehovej_gule(priemer3)
    hmotnost_celkom = hmotnost1 + hmotnost2 + hmotnost3
    return hmotnost_celkom

print('Hmotnosť snehuliaka je', hmotnost_snehuliaka(1, 0.7, 0.4))
```

**Poznámka:**

Aj v tomto prípade odporúčame žiakov smerovať k vhodnej dekompozícii problému na menšie podproblémy (vyššie uvedené riešenie nie je jediným správnym) a vhodnému pomenovaniu identifikátorov (funkcií a premenných). Výsledný program tak bude prehľadný a ľahko pochopiteľný.

Hodnotu konštanty  $\pi$  môžeme v riešení nahradiť približnou hodnotou 3,14. Presnejšia hodnota je prístupná v konštante `pi` v module `math`. Ponechávame na zvážení učiteľa, ktorý prístup pre svojich žiakov uprednostní.

```
import math
print(math.pi)    #3.141592653589793
```

**VYHODNOTENIE (CCA 5 MIN)**

Následne požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal, a nie klasifikácia známkom. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

### *Sebahodnotiaci test*

V programe na prevod hodín na minúty vypadli niektoré časti zdrojového kódu. Doplňte chýbajúce časti tak, aby sme dostali nasledovný výstup: 5 hodín je 300 minút

```
def hodiny_na_minuty(hodiny):  
    minuty =   
    return   
  
cas_v_hodinach = 5  
cas_v_minutach = (cas_v_hodinach)  
print(cas_v_hodinach, 'hodín je', , 'minút')
```

Riešenie:

```
def hodiny_na_minuty(hodiny):  
    minuty =   
    return   
  
cas_v_hodinach = 5  
cas_v_minutach =   
print(cas_v_hodinach, 'hodín je', , 'minút')
```

## 08 CHYBY VO VÝPOČTOCH, ICH ROZPOZNÁVANIE A ODSTRAŇOVANIE

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>• pomocou nástrojov na interakciu,</li> <li>• interpretácia zápisu riešenia,</li> <li>• hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>• vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>• používať premennú vo výrazoch,</li> <li>• vytvárať a používať vlastné funkcie s parametrami a návratovou hodnotou,</li> <li>• používať cyklus s pevným počtom opakovaní.</li> </ul>	
<b>Ciele</b>	
<i>Žiakom osvojované vedomosti a zručnosti</i>	<i>Žiakom rozvíjané spôsobilosti</i>
<b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>• rozpoznávať situácie, kedy treba získať vstup,</li> <li>• identifikovať vlastnosti vstupnej informácie (obmedzenia, rozsah, formát).</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>• krokovat' riešenie, simulujú činnosť vykonávateľa s postupnosťou príkazov, s výrazmi a premennými, s vetvením a s cyklami,</li> <li>• dopĺňať, dokončovať, modifikovať rozpracované riešenie.</li> </ul> <b>Hľadanie a opravovanie chýb:</b> <ul style="list-style-type: none"> <li>• rozpoznávať, kedy program pracuje nesprávne,</li> <li>• hľadať chybu vo vlastnom, nesprávne pracujúcom programe a opraviť ju,</li> <li>• zisťovať, pre aké vstupy, v ktorých prípadoch, situáciách program zle pracuje,</li> <li>• uvádzať kontra príklad, kedy niečo neplatí, nefunguje,</li> <li>• posudzovať a overovať správnosť riešenia (svojho aj cudzieho).</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>• vedieť identifikovať chyby v programoch,</li> <li>• navrhovať vhodné testovacie hodnoty,</li> <li>• testovať programy pomocou testovacích hodnôt,</li> <li>• používať ladiace nástroje vývojového prostredia pre hľadanie a odstraňovanie chýb,</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>• (LOG2) využitím logických zdôvodnení predpokladať správanie sa jednoduchých programov (výsledok realizácie matematických výpočtov),</li> <li>• (LOG3) využitím logických zdôvodnení detegovať a opravovať chyby v programoch a algoritmoch.</li> </ul> Vyhodnotenie: <ul style="list-style-type: none"> <li>• (VYH2) definovať vlastné kritériá pre vyhodnotenie priebehu alebo výsledkov programu (testovacie hodnoty),</li> <li>• (VYH3) posúdiť správnosť postupu na základe definovaných kritérií (testovanie programu pomocou testovacích hodnôt, testovať program),</li> <li>• (VYH4) posúdiť kritéria pre vyhodnotenie z pohľadu relevantnosti a úplnosti (posúdiť, či navrhnuté testovacie hodnoty sú vhodné).</li> </ul>

<ul style="list-style-type: none"> <li>načítať a vhodne upraviť (pretypovať) vstup od používateľa.</li> </ul>	
<b>Riešený didaktický problém</b>	
<p>Vo vyučovaní sa chyba často chápe ako niečo negatívne, nechcené, prípadne ako zlyhanie žiaka, programátora. Chyby sú však prirodzenou súčasťou programovania (nie len toho žiackeho) a je nesprávne ich vnímať ako niečo výnimočné a nežiadúce. Je naivné snažiť sa dosiahnuť stav, kedy žiaci chyby nerobia. Často sa chyby v žiackych programoch odstraňujú v rýchllosti, za pomoci učiteľa. Žiak tak môže nadobudnúť dojem, že chyby nemôže robiť a ak ich aj spraví, nepozná mechanizmy ako ich identifikovať a odstrániť. V tejto metodike sa zameriavame na problematiku chýb (najmä logických), spôsoby ich identifikácie a následného odstránenia.</p>	
<b>Dominantné vyučovacie metódy a formy</b>	<b>Príprava učiteľa a pomôcky</b>
<ul style="list-style-type: none"> <li>riadené bádanie.</li> <li>frontálna, individuálna a skupinová forma (5-8 dvojíc žiakov).</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li><b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li><b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li><b>ucitel/pracovne_subory_riesenia/08/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li><b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li><b>ziak/pracovne_subory/08/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<b>Diagnostika splnenia vzdelávacích cieľov</b>	
Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.	

## Úvod

Toto je 8. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Metodika priamo nadväzuje na metodiku 1, v ktorej sa žiaci cielene stretli s rôznymi typmi chýb, najmä syntaktickými a behovými. V tejto metodike sa zameriame na logické chyby, ich identifikáciu a odstraňovanie.

Doterajšie programy spracovávali konštantné hodnoty, ktoré programátor priamo do programu zapísal. V tejto metodike žiakov oboznámime so spôsobom, ako získať a v prípade potreby upraviť (pretypovať) dáta od používateľa. Žiaci sa naučia vytvárať jednoduché používateľské rozhranie, ktoré umožní interaktivitu programu s používateľom.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou výučby je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (4 minúty)** – diskusia na tému chyby pri programovaní.
- **Skúmanie (8 minút)** – práca vo dvojiciach s pracovným listom (úloha 1 a 2).
- **Vysvetlenie (7 minút)** – žiacke vysvetlenie toho, čo zistili v predchádzajúcej časti a rozšírenie vysvetlenia učiteľom.
- **Rozpracovanie (15 minút)** – samostatné programovanie náročnejších úloh (úlohy 3 a 4 z pracovného listu).
- **Hodnotenie (6 minút)** – riešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 4 MIN)

Hodinu začneme krátkou diskusiou na tému chyby pri programovaní. Vyzveme žiakov, aby sa vyjadrili k nasledovným témam, resp. odpovedali na otázky:

- Robil som chyby počas predchádzajúceho programovania?
- Ako som prišiel na to, že v programe je chyba?
- Hľadal som chybu v programe aj keď to vyzeralo, že program funguje správne?
- Robia chyby aj iní programátori?
- Sú v programoch (textový procesor, webový prehliadač ...), ktoré používame, chyby? Ak áno, prečo a ako sa s tým vysporiadame?

Cieľom diskusie je, aby si žiaci uvedomili, že robiť chyby pri programovaní je prirodzené. Rovnako prirodzené je aj snažiť sa chyby hľadať a odstraňovať v programoch. Napriek tejto snahe výsledok nemusí byť vždy 100 % správny. Pozor, aby si žiaci takéto tvrdenie nevyložili nesprávne tak,

že je v poriadku, ak v ich programoch sú chyby. Tvrdenie treba chápať tak, že so zvyšujúcou sa náročnosťou riešenia vzrastá aj pravdepodobnosť výskytu nejakej neodhalenej chyby.

## SKÚMANIE (CCA 8 MIN)

V nasledujúcej časti môžu žiaci pracovať vo dvojiciach. Ich úlohou je overiť správnosť jedného žiackeho riešenia písomky z pohľadu logickej chyby (t. j. pre korektné vstupy program vráti nesprávny výsledok). Ak riešenie nie je správne, majú zdôvodniť prečo to autor, napriek tomu, že riešenia testoval, neodhalil. Vo funkcii `objem_gule()` nie je chyba. Zaradili sme ju ako úlohu proti stereotypu. Kódy nie je potrebné prepisovať do programu. Výpočty sú pomerne jednoduché, chyby je možné odhaliť len analýzou kódu.

**Úloha 1** Žiaci v teste z programovania dostali za úlohu naprogramovať tri funkcie:

- `obsah_kruhu(polomer)`,
- `objem_gule(polomer)`,
- `povrch_kvadra(a, b, c)`.

a pre nejaké konkrétne hodnoty si overiť správnosť riešenia. Karolove riešenie (uvedené v súbore **karol.py**) aj s overením správnosti je nasledovné:

```
def obsah_kruhu(polomer):  
    vysledok = 3.14 * polomer * 2  
    return vysledok  
  
def objem_gule(polomer):  
    vysledok = 4 * 3.14 * polomer * polomer ** 2 / 3  
    return vysledok  
  
def povrch_kvadra(a, b, c):  
    vysledok = 2 * (a * b + b * c + b * a)  
    return vysledok  
  
print(obsah_kruhu(2)) # má byť približne 12.56, výsledok je ok  
print(objem_gule(3)) # má byť približne 113.04, výsledok je ok  
print(povrch_kvadra(20, 10, 10)) # má byť 1000, výsledok je ok
```

Preskúmajte Karolove riešenie. Môže byť Karol spokojný a očakávať dobrú známku? Ak nie, akých chýb sa dopustil a prečo ich neodhalil, keď si overoval správnosť svojho programu? Zdôvodnite svoju odpoveď.

Riešenie:

Karol sa v svojom riešení dopustil dvoch typov chýb:

1. Vo funkciách `obsah_kruhu()` a `povrch_kvadra()` napísal nesprávny vzorec na výpočet.
2. Zvolil si nevhodné hodnoty na testovanie. Pri výpočte obsahu kruhu majú výrazy  $2 * 2$  a  $2^2$  rovnaké hodnoty. Pri výpočte povrchu kvádra zvolil rovnaké dĺžky dvoch hrán, takže výrazy  $c * a$  a  $b * a$  majú rovnaké hodnoty. Pri týchto testovacích hodnotách sa chyba neprejaví. Ak by riešenie otestoval pre viac hodnôt alebo pre lepšie navrhnuté hodnoty, odhalil by chybu vo funkciách.



Nasledujúca úloha je zameraná na skúmanie nového príkazu `input()` a pretypovanie vstupnej hodnoty, v tomto prípade príkaz `int()`. Ak žiaci vynechajú príkaz `int()`, program neskončí predčasne s chybou. Výsledkom je ale chybný výstup. Predpokladáme, že jeden zo spôsobov ako zistiť funkčnosť príkazu (`int()`) je príkaz pri jednom spustení použiť a pri druhom spustení nepoužiť. Z rozdielu výstupov je možné dedukovať jeho funkčnosť.

**Úloha 2** Čo robí nasledujúci program? Ak sú v ňom nejaké, pre vás neznáme príkazy, preskúmajte čo robia. Svoje zistenia si zaznamenajte. Program nájdete v súbore **neznamy.py**.

```
meno = input('Ako sa voláš? Napíš svoje meno: ')
hodiny = input('Zadaj, koľko hodín si už venoval programovaniu: ')
hodiny = int(hodiny)

minuty = hodiny * 60

print('Ahoj', meno)
print('Celkovo si programovaniu venoval už', минуты, 'minút.')
```

Riešenie:

Príkaz `input()` vypíše zadaný text do konzoly a čaká na vstup od používateľa. Používateľom zadanú hodnotu vráti. Vrátenú hodnotu sme v programe pomenovali (`meno` alebo `hodiny`).

Príkaz `int()` pretypoval zadanú hodnotu na celé číslo. Bez tohto príkazu program vo výstupe 60-krát zopakuje zadanú hodnotu. Je to preto, lebo zadanú hodnotu berie ako text.

## VYSVETLENIE (CCA 7 MIN)

V tejto časti by žiaci mali vysvetliť svoje zistenia.

V prvej časti by sa mali žiaci vyjadriť ku chybám v programoch a k výberu vhodných testovacích dát. Zistenia by mali smerovať k potrebe testovania programov a k premyslenému výberu testovacích dát. Testovacie dáta by sme mali voliť tak, aby sme vedeli aký výstup očakávame a vedeli ho porovnať so skutočnosťou. Vhodný výber testovacích dát by mal eliminovať možnosť prehliadnutia chyby. Ak nevieme testovacie dáta takto cielene vybrať, môžeme zrealizovať viac testov, s rôznymi hodnotami, pričom porovnávame očakávané výstupy so skutočnými.

V druhej časti žiaci uvedú svoje zistenia k príkazom `input()` a `int()`. Ich zistenia by mali smerovať k tomu, že príkaz `input()` použijeme na zobrazenie výzvy a získanie vstupu od používateľa. Používateľove vstupy sú v programe prístupné ako texty – typ reťazec. Ak v programe očakávame, že vstupom bude číslo, musíme takto získanú hodnotu pretypovať na zodpovedajúci typ.

Odporúčame, aby učiteľ uviedol aj príkaz `float()`, ktorý použijeme na pretypovanie hodnoty na desatinné číslo. Rovnako je dobré upozorniť žiakov na to, že v niektorých prípadoch, ak vynecháme pretypovanie, môže program skončiť predčasne s chybou (ak danú operáciu nie je možné vykonať s hodnotou typu reťazec).

## ROZPRACOVANIE (CCA 15 MIN)

V nasledujúcej úlohe žiaci navrhujú testovacie hodnoty pre vytvorený program. Chyby v programe už nie sú také „naivné“ a na prvý pohľad viditeľné ako v predchádzajúcich úlohách. Preto je dôležité testovacie hodnoty navrhnúť premyslene. Nie je ani tak dôležitý počet testovacích hodnôt, ale skôr ich cielený výber. Žiaci by mali vedieť zdôvodniť navrhnuté testovacie hodnoty.

**Úloha 3** Nasledujúci program by mal pomôcť taxikárom vypočítať sumu, ktorú by mal zákazník zaplatiť. Taxikár má takéto pravidlá pre výpočet výslednej sumy:

- nástupné: 1,5 EUR,
- čakanie: 30 centov / minúta,
- jazda: 70 centov / km.

Navrhnite vhodné testovacie hodnoty a overte správnosť nasledujúceho programu.

Zdôvodnite, prečo ste navrhli práve tieto testovacie hodnoty.

Ak ste v programe našli chyby, opravte ich.

Program je uložený v súbore **taxi.py**.

```
def vysledna_suma(cakanie_min, prejdene_km):
    nastupne = 1.5
    cena_cakanie = 0.3 * cakanie_min
    cena_jazda = 0.7 * prejdene_km
    suma = nastupne + cena_cakanie + cena_jazda
    return suma

vzdialenost = input('Zadaj prejdenú vzdialenosť v km: ')
vzdialenost = int(vzdialenost)

cakanie = input('Zadaj dobu čakania v min: ')
cakanie = int(cakanie)

zaplatit = vysledna_suma(vzdialenost, cakanie)
print('Celková suma na zaplatenie:', zaplatit)
```

Riešenie:

Program spracováva dva vstupy: prejdenú vzdialenosť a dobu čakania. Testovacie vstupy pre tieto hodnoty by mali byť rôzne, aby sme vylúčili ich zámenu v programe.

Obidva testovacie vstupy by mohli byť aj nie celé čísla. Program by sme mali otestovať aj pre celé aj pre desatinné čísla.

Vstupy by sme mali navrhnúť tak, aby sme ľahko vedeli vypočítať, akú odpoveď očakávame.

Testovacie vstupy môžu byť napr.	cakanie	vzdialenost	očakávaný výsledok
	1	20	15,8
	1,5	10	8,95
	20	1,5	8,55

Pomocou týchto testovacích hodnôt odhalíme nasledovné chyby:

- pri volaní funkcie je vymenené poradie argumentov,
- program predpokladá len celočíselné vstupy.

Väčšina pokročilejších vývojových prostredí obsahuje ladiace nástroje, ktoré nám umožnia sledovať, krok za krokom, priebeh realizácie výpočtu. Nasledujúca úloha je zameraná na použitie ladiaceho nástroja (angl. debugger) pri krokovaní programu. Odporúčame, aby učiteľ demonštroval žiakom použitie ladiaceho nástroja v používanom vývojovom prostredí.

**Úloha 4** V predchádzajúcich úlohách bolo pomerne jednoduché opraviť chyby, ak sme prišli na to, že programy sú chybné. Niekedy chybu, aj keď o nej vieme, nie je jednoduché lokalizovať len na základe nesprávneho výsledku programu. V týchto prípadoch by nám pomohlo vedieť, ako výpočet prebieha v jednotlivých krokoch programu.

Pre výpočet faktoriálu prirodzeného čísla sme si vytvorili program **vypocet\_faktorialu.py**. Otvorte daný program a overte jeho správnosť. Ak je program chybný, postupným krokovaním výpočtu nájdite a následne opravte chyby.

Riešenie:

Krokovaním programu odhalíme, že premenná `cislo` nadobúda hodnoty od 0 po  $n-1$ . Výsledok súčinu bude preto vždy nula. Jedno z riešení je použiť v súčine hodnotu `cislo + 1`.

#### Poznámka:

Chybu v predchádzajúcej úlohe je možné elegantnejšie vyriešiť úpravou parametrov funkcie `range`: `range(1, n + 1)`. Nechávame na zvážení učiteľa, či danú alternatívu žiakom uvedie. Použitie príkazu `range()` týmto spôsobom je uvedené aj v časti Vedomosti v kocke.

## VYHODNOTENIE (CCA 6 MIN)

V záverečnej časti hodiny požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

Program s funkciou `NSD()`, ani program v 2. úlohe nie sú žiakom prístupné. Testovacie hodnoty by mali žiaci navrhnúť len na základe požiadavky na funkcionálnu programovú úlohu. V 2. úlohe by žiaci mali analyzovať uvedený kód a na základe analýzy odpovedať na otázku.

#### Sebahodnotiaci test

1.	Vytvorili sme funkciu <code>NSD()</code> , ktorá pre dve zadané prirodzené čísla vráti ich najväčšieho spoločného deliteľa. Ktoré z uvedených <b>hodnôt sú vhodné pre testovanie</b> správnosti funkcie vzhľadom na vzájomné vzťahy medzi zadávanými hodnotami?  a) <code>NSD(12, 16)</code> b) <b><code>NSD(12, 16)</code></b> c) <code>NSD(12, 16)</code> d) <code>NSD(17, 19)</code> <code>NSD(45, 75)</code> <b><code>NSD(19, 12)</code></b> <code>NSD(45, 75)</code> <code>NSD(1, 12)</code> <code>NSD(20, 50)</code> <b><code>NSD(23, 23)</code></b> <code>NSD(17, 19)</code> <code>NSD(17, 17)</code> <code>NSD(60, 150)</code> <b><code>NSD(1, 12)</code></b> <code>NSD(13, 11)</code> <code>NSD(13, 17)</code>
2.	Aký je <b>výstup</b> nasledovného programu, ak používateľ zadá na vstupe hodnoty 5 a 12?  <pre>dlzka_dovolenky = input('Zadaj koľko dní budeš na dovolenke: ') dlzka_dovolenky = int(dlzka_dovolenky)</pre>

```
rozpocet_den = input('Zadaj rozpočet na jeden deň: ')\n\nrozpocet_dovolenka = dlzka_dovolenky * rozpocet_den\nprint('Rozpočet na dovolenku:', rozpocet_dovolenka)
```

- a) Rozpočet na dovolenku: 60,
- b) Rozpočet na dovolenku: 555555555555,
- c) Rozpočet na dovolenku: 1212121212,**
- d) Program skončí predčasne s chybou.

Pri hľadaní najväčšieho spoločného deliteľa dvoch prirodzených čísiel je vhodné uvažovať rôzne vzájomné vzťahy medzi danými číslami:

- čísla sú súdeliteľné,
- čísla sú nesúdeliteľné, prípadne jedno z čísiel je 1,
- jedno z čísiel je prvočíslo,
- čísla sú rovnaké.

Všetky uvedené vzťahy sú uvedené v možnosti b).

Druhá úloha je zameraná na pochopenie príkazu vstupu a následného pretypovania vstupných hodnôt. Keďže rozpočet na jeden deň sme nepretypovali, následne s ním pracujeme ako s reťazcom. Výsledok je denný rozpočet zopakovaný toľkokrát, koľko dní trvá dovolenka.

## 09 PODMIENENÝ PRÍKAZ

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>jazyk na zápis riešenia,</li> <li>pomocou vetvenia.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>používať premennú vo výrazoch,</li> <li>načítavať a vypisovať dáta pomocou príkazov <code>input</code> a <code>print</code>,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať matematické výrazy pri vyjadrovaní vzťahov a podmienok.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie a podmienky, kedy treba použiť vetvenie,</li> <li>rozpoznávať, aká časť algoritmu sa má vykonať pred, v rámci a po skončení vetvenia,</li> <li>riešiť problémy, ktoré vyžadujú vetvenie so zloženými podmienkami (s logickými spojkami),</li> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>využiť pri riešení úloh riadiaci príkaz <code>if-elif-else</code>.</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG5) logicky zdôvodniť rozdelenie algoritmu na menšie časti (testovanie jednotlivých prípadov).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (programy na detegovanie a kódovanie číselných intervalov),</li> <li>(ALG8) zapísať algoritmy v konkrétnom formálnom jazyku (zápis algoritmov ktoré riešia uvedené problémy).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia - lineárne rozdeliť problémy/procesy na menšie časti tak, aby sa dali využiť na dosiahnutie cieľa (rozdelenie riešenia problému na rôzne situácie v závislosti od splnenia podmienok).</li> </ul> Hľadanie vzorov: <ul style="list-style-type: none"> <li>(VZO5) preniesť riešenia z jedného problému na druhý problém (spôsob rozpoznávania príslušnosti hodnoty do intervalu).</li> </ul>
<b>Riešený didaktický problém</b>	
<p>Príkaz vetvenia je dôležitou riadiacou štruktúrou. Najčastejšie žiacke miskoncepce vychádzajú z nesprávneho zostavenia podmienok, najmä zložených podmienok. V kóde sa tak opakovane testujú rovnaké časti podmienok, prípadne niektoré situácie sa netestujú vôbec. Ďalším problémom je neschopnosť zostavovať vnorené podmienky, namiesto série za sebou idúcich podmienok. Výsledný program je tak neefektívny a pri testovaní sa nevyužívajú zistenia z predchádzajúcich testov.</p>	
<b>Dominantné vyučovacie metódy a formy</b>	<b>Príprava učiteľa a pomôcky</b>

- riadené bádanie,
- frontálna, individuálna a skupinová forma (dvojice žiakov).

Pre učiteľa:

- **ucitel/programovanie\_v\_pythone.pdf**  
metodika vyučovania,
- **ucitel/pracovny\_zosit\_riesene\_ulohy.docx**  
pracovný zošit a riešenia úloh,
- **ucitel/pracovne\_subory\_riesenia/09/**  
riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.

Pre žiaka:

- **ziak/pracovny\_zosit.docx**  
pracovný zošit,
- **ziak/pracovne\_subory/09/**  
pracovné súbory pre žiaka.

Použitie digitálnych nástrojov: NUTNÉ

### *Diagnostika splnenia vzdelávacích cieľov*

Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.



EURÓPSKA ÚNIA  
Európsky sociálny fond  
Európsky fond regionálneho rozvoja



OPERAČNÝ PROGRAM  
ĽUDSKÉ ZDROJE



MINISTERSTVO  
ŠKOLSTVA, VEDY,  
VÝSKUMU A ŠPORTU  
SLOVENSKEJ REPUBLIKY



it akadémia

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu  
v rámci Operačného programu Ľudské zdroje

[www.minedu.sk](http://www.minedu.sk) [www.employment.gov.sk/sk/esf/](http://www.employment.gov.sk/sk/esf/) [www.itakademia.sk](http://www.itakademia.sk)

## Úvod

Toto je 9. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Obsahovo využíva poznatky, postupy, ale aj príklady riešené v predošlých metodikách, ktoré rozširuje pomocou podmienok a vetvenia v programoch.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou výučby je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – frontálny motivačný rozhovor so žiakmi – vyvodenie myšlienky podmienok a ich aplikácie z bežného života.
- **Skúmanie (7 minút)** – práca vo dvojiciach s pracovným listom (úlohy 1 a 2).
- **Vysvetlenie (8 minút)** – žiacke vysvetlenie toho, čo zistili v predchádzajúcej časti.
- **Rozpracovanie (15 minút)** – samostatné programovanie náročnejších úloh (úlohy 3 až 7 z pracovného listu).
- **Vyhodnotenie (5 minút)** – kontrola žiackych prác, vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 5 MIN)

Hodinu začneme krátkou aktivitou, v ktorej budú žiaci formulovať (formalizovať) pravidlá správania sa pre zamestnancov meteorologickej stanice. Okrem jednotlivých aspektov počasia by žiaci mali uvažovať aj ich kombinácie.

**Úloha 1** Meteorológom zamestnávateľ poskytol pracovné oblečenie: nie veľmi teplý plášť do dažďa a premokavý kabát do chladného počasia. Ak zamestnanec nemá vhodné oblečenie do daného počasia, môže pracovať z domu. Ak počasie nevyžaduje ochranné oblečenie, môže zamestnanec relaxovať v prírode.

Sformulujte jednoduché pravidlá pre meteorológa Karola, aby sa vedel ľahko rozhodnúť, ako bude tráviť deň.

Riešenie:

**AK** prší a je chladno **TAK** pracuj z domu,

**AK** prší a nie je chladno **TAK** zober si plášť do dažďa a choď do práce,

**AK** neprší a je chladno **TAK** zober si kabát a choď do práce,

**AK** neprší a nie je chladno **TAK** choď na výlet do prírody.

alebo šikovnejšie riešenie, kde každú situáciu otestujeme najviac 1x:

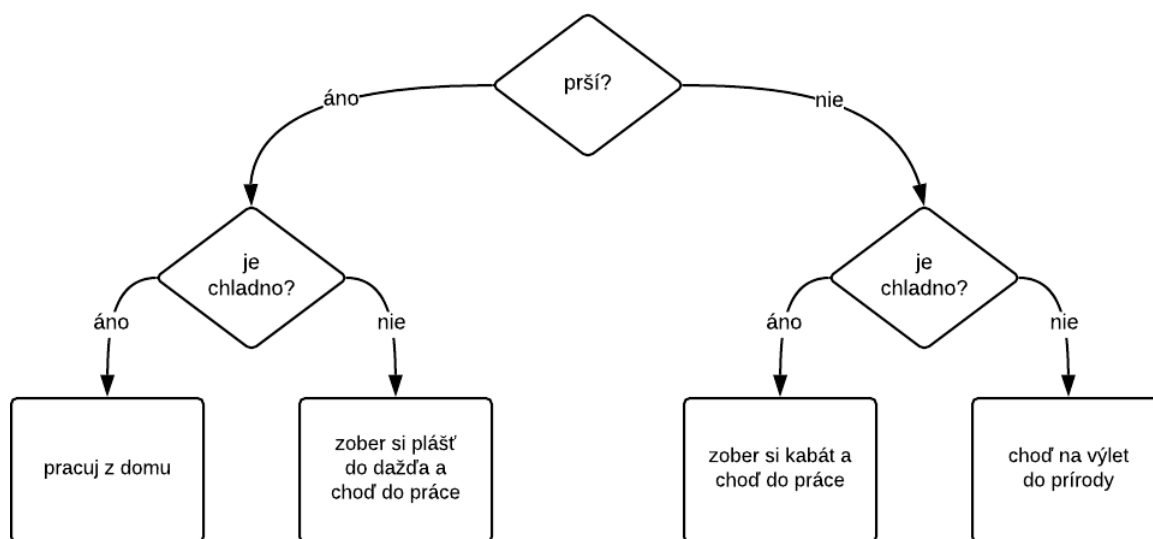
**AK**    prší    **TAK**    **AK**    je chladno    **TAK**    pracuj z domu

<b>INAK</b>	<b>AK</b>	je chladno	<b>INAK</b>	zober si plášť do dažďa a choď do práce
			<b>TAK</b>	zober si kabát a choď do práce
			<b>INAK</b>	choď na výlet do prírody

Riešenia žiakov by mali smerovať k vyššie uvedenej sérii pravidiel.

Prvá séria pravidiel obsahuje zložené podmienky, ktorých časti je potrebné vyhodnocovať opakovane. Niektorí žiaci môžu navrhnúť šikovnejšie riešenie, v ktorom sa každá časť testuje najviac 1x (druhá séria pravidiel).

Niektorí žiaci môžu mať problém pochopiť spôsob vyhodnocovania podmienok, ak sú tieto zapísané ako text. Podmienky je možné prehľadnejšie zapísať aj graficky spôsobom napr.:



Žiaci môžu diskutovať o situácii, či si možno zároveň obliecť plášť do dažďa a kabát. Výsledok tejto diskusie ponechávame na učiteľov, ale odporúčame uprednostniť odpoveď nie.

Výsledkom tejto časti je, aby si žiaci uvedomili, že podobné rozhodnutia na základe nejakých pravidiel robíme aj v bežnom živote. Môžeme prípadne uviesť niekoľko jednoduchých príkladov (rozhodnutie kúpiť nejakú vec na základe množstva dostupných peňazí, prechod cez priechod pre chodcov na základe dopravných pravidiel, výpočet koreňov kvadratickej rovnice pre rôzne hodnoty diskriminantu a pod.) Podobne, na základe rozhodnutí, môže prebiehať realizácia programu.

## SKÚMANIE (CCA 7 MIN)

Žiaci pracujú samostatne, prípadne vo dvojiciach. Ak žiaci pociťujú potrebu spolu komunikovať pri riešení nasledujúcich úloh, umožnite im to. Žiaci pracujú s pracovným listom a riešia úlohy 2 a 3. Učiteľ do práce nezasahuje, monitoruje aktivitu žiakov a v prípade potreby len usmerní ich činnosť.

V druhej úlohe sa žiaci prvýkrát stretnú s príkazom `if-else`. Na základe diskusie z časti Zapojenie a na základe textov použitých v úlohe predpokladáme, že funkciu nového príkazu identifikujú žiaci správne.



**Úloha 2** Preskúmajte program v súbore **podmienky1.py**. Nájdite v ňom neznáme príkazy a odhadnite ich funkciu.

```
print('Test')
pocet = input('Koľko mesiacov má jeden rok: ')
pocet = int(pocet)

if pocet == 12:
    print('správna odpoveď')
else:
    print('nesprávna odpoveď')

print('Koniec testu')
```

Riešenie:

Neznáme príkazy a ich funkcionality:

`if` – slúži na testovanie platnosti nejakej podmienky, ak podmienka platí, vykonajú sa príkazy v bloku `if`  
`else` – označuje blok príkazov, ktoré sa vykonajú ak podmienka neplatí.

V tretej úlohe sa žiaci stretnú s viacnásobným vetvením a s časťou `elif`. Aj tu predpokladáme, že funkciu tejto časti žiaci identifikujú správne.

**Úloha 3** Preskúmajte program v súbore **podmienky2.py**. Nájdite v ňom neznáme príkazy a odhadnite ich funkciu.

```
print('Vekové kategórie')
vek = input('Zadaj vek človeka rokoch: ')
vek = int(vek)

if vek == 0:
    print('Novorodenec alebo dojča')
elif vek <= 3:
    print('Batoľa')
elif vek <= 6:
    print('Predškolský vek')
elif vek <= 12:
    print('Mladší školský vek')
elif vek <= 15:
    print('Mladší školský vek')
elif vek <= 18:
    print('Puberta')
else:
    print('Dospelý')
```

Riešenie:

Neznáme príkazy a ich funkcionality:

`elif` – slúži na testovanie ďalších podmienok, ak žiadna z predchádzajúcich podmienok neplatí.

Po nájdení prvej platnej podmienky sa vykonajú príkazy v príslušnom bloku, ďalšie podmienky sa už netestujú a vykonávanie celého bloku `if` sa ukončí.

## VYSVETLENIE (CCA 8 MIN)

V tejto časti by žiaci mali vysvetliť svoje zistenia.

Vysvetlenie žiakov by malo smerovať k tomu, že priebeh výpočtu je možné na základe platnosti nejakej podmienky, resp. podmienok vetviť niekoľkými smermi. Slúži na to príkaz `if-elif-else`. Pri realizácii výpočtu sa vykonávajú príkazy v tej vetve, ktorej podmienka sa vyhodnotila ako prvá splnená. Aj keď sú niektoré z nasledujúcich podmienok platné, vetvenie výpočtu do ďalších vetiev sa už nerealizuje. Preto je dôležité venovať pozornosť samotnému poradiu podmienok. Ak žiadna z uvažovaných podmienok neplatí, je možné výpočet presmerovať do vetvy `else`. Po vykonaní bloku `if` (niektorej z jeho vetví) realizácia výpočtu pokračuje len jednou vetvou, ktorá sa môže prípadným ďalším príkazom `if` rozdeliť. Príkazy v jednotlivých blokoch prislúchajúcim konkrétnym podmienkam sú odsadené.

Odporúčame, aby učiteľ upriamil pozornosť žiakov aj na samotnú podmienku – výraz, vyhodnotením ktorého je logická hodnota `True` alebo `False`.

Odporúčame, aby učiteľ uviedol aj verziu príkazu `if` bez časti `else`.

## ROZPRACOVANIE (CCA 15 MIN)

V nasledujúcich úlohách pracujú žiaci na „vylepšení“ riešenia úlohy o vekových kategóriách. Úlohy 4 a 5 sú len miernym rozšírením programu z úlohy 3.

V úlohách 6 a 7 sa vyžaduje kombinácia viacerých podmienok, resp. vnorenie podmienok.

**Úloha 4** V úlohe o vekových kategóriách (súbor **podmienky2.py**) ste skúmali, ako funguje príkaz `if`.

Pracuje program správne, ak zadáme záporný vek alebo vek ako desatinné číslo? Ak nie, upravte program tak, aby akceptoval na vstupe aj desatinné číslo (napr. ak človek má 3,5 roka) a v prípade záporného čísla program výpisom upozornil na chybné zadanú hodnotu.

Riešenie:

```
print('Vekové kategórie')
vek = input('Zadaj vek človeka rokoch: ')
vek = float(vek)

if vek < 0:
    print('Chyba! Vek nemôže byť záporný.')
elif vek == 0:
    print('Novorodenec alebo dojča')
elif vek <= 3:
    print('Batola')
elif vek <= 6:
    print('Predškolský vek')
elif vek <= 12:
    print('Mladší školský vek')
elif vek <= 15:
    print('Mladší školský vek')
elif vek <= 18:
    print('Puberta')
```

```
else:
    print('Dospelý')
```

**Úloha 5** Doplníte do programu **podmienky2.py** aj kategóriu staroba, do ktorej sa človek dostane v 70. roku života.

Riešenie

```
print('Vekové kategórie')
vek = input('Zadaj vek človeka rokoch: ')
vek = float(vek)

if vek < 0:
    print('Chyba! Vek nemôže byť záporný.')
elif vek == 0:
    print('Novorodenec alebo dojča')
elif vek <= 3:
    print('Batoľa')
elif vek <= 6:
    print('Predškolský vek')
elif vek <= 12:
    print('Mladší školský vek')
elif vek <= 15:
    print('Mladší školský vek')
elif vek <= 18:
    print('Puberta')
elif vek < 70:
    print('Dospelý')
else:
    print('Staroba')
```

**Úloha 6** V letnom tábore organizujú pravidelné súťažné popoludnia. Aby si deti vyskúšali rôzne športy, rozdeľujú ich podľa rôznych kritérií. Pre dnešné popoludnie sa deti rozdeľujú podľa pohlavia a výšky do jednotlivých športov podľa nasledujúcej tabuľky:

Pohlavie\výška	< 160 cm	<= 170 cm	<= 180 cm	> 180 cm
Chlapec	futbal	vodné pólo		basketbal
Dievča	akvabely	tenis	volejbal	

Vytvorte program **sportove\_popoludnie.py**, ktorý pre zadané pohlavie a výšku dieťaťa vypíše, v akom športe bude dieťa súťažiť.

Riešenie:

```
pohlavie = input('Zadaj pohlavie (0-chlapec, 1-dievča): ')
pohlavie = int(pohlavie)
vyska = input('Zadaj výšku v cm: ')
vyska = int(vyska)
```

```

if pohlavie == 0:
    if vyska < 160:
        print('futbal')
    elif vyska <= 180:
        print('vodné pólo')
    else:
        print('basketbal')
else:
    if vyska < 160:
        print('akvabely')
    elif vyska <= 170:
        print('tenis')
    else:
        print('volejbal')

```

S reťazcom sa žiaci stretli v predchádzajúcej metodike, pri načítaní vstupu pomocou príkazu `input()`. Kód pohlavia nemusíme načítavať a konvertovať do číselnej hodnoty. Môžeme ho načítať a ďalej s ním pracovať ako s textom:

```

pohlavie = input('Zadaj pohlavie (c-chlapec, d-dievča): ')
vyska = input('Zadaj výšku v cm: ')
vyska = int(vyska)

if pohlavie == 'c':
    ...

```

Ponechávame na učiteľovi, ku ktorému z riešení bude žiakov smerovať.

**Úloha 7** Využite programový kód z predchádzajúcej úlohy (rozdelenie detí do jednotlivých športov) a vytvorte funkciu `urci_sport()`, ktorá pre zadané pohlavie a výšku zistí a vráti názov športu, v ktorom bude dieťa súťažiť.

Riešte

podľa

pokynov

učiteľa

Riešenie:

```

def urci_sport(pohlavie, vyska):
    if pohlavie == 0:
        if vyska < 160:
            sport = 'futbal'
        elif vyska <= 180:
            sport = 'vodné pólo'
        else:
            sport = 'basketbal'
    else:
        if vyska < 160:
            sport = 'akvabely'
        elif vyska <= 170:
            sport = 'tenis'
        else:
            sport = 'volejbal'
    return sport

pohlavie = input('Zadaj pohlavie (0-chlapec, 1-dievča): ')
pohlavie = int(pohlavie)
vyska = input('Zadaj výšku v cm: ')
vyska = int(vyska)

```

```
print(urci_sport(pohlavie, vyska))
```

**Poznámka:**

V programovacom jazyku Python existujú logické operátory `and`, `or` a `not`. Tieto predstavíme neskôr v metodike 12. Ich nesprávne použitie môže viesť k viacnásobnému testovaniu rovnakých podmienok a následne neefektívnemu výpočtu, napr.

```
if pohlavie == 0 and vyska < 160: # pre každú výšku opakovane testujeme
    pohlavie
```

Test, či hodnota patrí do nejakého intervalu, je možné v Pythone realizovať rovnakým spôsobom ako v matematike:

```
if lavy_okraj <= hodnota <= pravy_okraj
```

**Úloha 8**

Riešte  
podľa  
pokynov  
učiteľa

V pracovnom liste číslo 7 ste v súbore **zdravie.py** vytvorili funkciu `pocitaj_bmi()`, ktorá pre zadanú hmotnosť a výšku človeka vypočítala a vrátila hodnotu BMI. Využite vytvorenú funkciu a vytvorte funkciu `kategoria_bmi()`, ktorá pre zadanú hmotnosť a výšku vráti kategóriu BMI, do ktorej človek patrí. Riešenie uložte do súboru **bmi\_kategorie.py**.

Pomôcka: Pre jednotlivé kategórie môžete využiť nasledujúce hraničné hodnoty:

<code>bmi &lt; 18,5</code>	podvýživa
<code>18,5 &lt;= bmi &lt; 25</code>	ideálna, zdravá hmotnosť
<code>25 &lt;= bmi &lt; 30</code>	nadváha
<code>30 &lt;= bmi &lt; 40</code>	obezita
<code>40 &lt;= bmi</code>	ťažká obezita

Riešenie:

```
def pocitaj_bmi(hmotnost, vyska):
    vysledok = hmotnost / vyska ** 2
    return vysledok

def kategoria_bmi(hmotnost, vyska):
    bmi = pocitaj_bmi(hmotnost, vyska)
    if bmi < 18.5:
        kategoria = 'podvýživa'
    elif bmi < 25:
        kategoria = 'ideálna, zdravá hmotnosť'
    elif bmi < 30:
        kategoria = 'nadváha'
    elif bmi < 40:
        kategoria = 'obezita'
    else:
        kategoria = 'ťažká obezita'
    return kategoria
```

```
hmotnost = 95
```

```
vyska = 1.90
kategoria = kategoria_bmi(hmotnost, vyska)
print('Kategória BMI je:', kategoria)
```

## VYHODNOTENIE (CCA 5 MIN)

Následne požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal a nie klasifikácia známkou. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

### Sebahodnotiaci test

1.	<p>Predajca predáva vianočné stromčeky a cenu stanovil podľa výšky stromčeka. Vytvoril si program, ktorý by mal pre zadanú výšku vypísať cenu stromčeka.</p> <pre>vyska = input('Zadaj výšku stromčeka v cm: ') vyska = int(vyska)  if vyska &lt; 150:     print('cena: 8 EUR') elif vyska &lt; 180:     print('cena: 10 EUR') elif vyska &lt; 200:     print('cena: 12 EUR') else:     print('cena: 15 EUR')</pre>																
	<p>Aký bude výpis programu, ak zadáme výšku stromčeka 165 cm?</p> <table><tr><td>a)</td><td>b)</td><td>c)</td><td>d)</td></tr><tr><td>cena: 15 EUR</td><td><b>cena: 10 EUR</b></td><td>cena: 10 EUR</td><td>cena: 10 EUR</td></tr><tr><td></td><td></td><td>cena: 12 EUR</td><td>cena: 12 EUR</td></tr><tr><td></td><td></td><td></td><td>cena: 15 EUR</td></tr></table>	a)	b)	c)	d)	cena: 15 EUR	<b>cena: 10 EUR</b>	cena: 10 EUR	cena: 10 EUR			cena: 12 EUR	cena: 12 EUR				cena: 15 EUR
a)	b)	c)	d)														
cena: 15 EUR	<b>cena: 10 EUR</b>	cena: 10 EUR	cena: 10 EUR														
		cena: 12 EUR	cena: 12 EUR														
			cena: 15 EUR														
2.	<p>Čo je výstupom nasledovného programu?</p> <pre>koeficient_a = 5 koeficient_b = 7  if koeficient_a &lt; 3 :     if koeficient_b &lt;= 7:         print('Kategória A')     else:         print('Kategória A') else:     if koeficient_b &gt; 7:         print('Kategória C')     else:         print('Kategória D') print('Koniec')</pre>																

	a)	b)	c)	d)
	Kategória A	<b>Kategória D</b>	Kategória D	Kategória A
	Kategória D	<b>Koniec</b>		Kategória D
	Koniec			

## 10 OPAKOVANIE II. + DIDAKTICKÝ TEST

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>pomocou nástrojov na interakciu,</li> <li>pomocou premenných,</li> <li>pomocou cyklov,</li> <li>pomocou vetvenia,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 2 vyučovacie hodiny
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>získavať vstupné dáta (<i>input</i>), zobrazovať výstupné dáta (<i>print</i>), upraviť vstupné dáta (pretypovať),</li> <li>používať cyklus s pevným počtom opakovaní (<i>range</i>), zmeniť štandardné správanie sa premennej v cykle (<i>range(štart, stop, krok)</i>),</li> <li>vytvárať logické podmienky a používať podmienený príkaz (<i>if-elif-else</i>).</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>identifikovať problém, ktorý sa bude riešiť algoritmicky.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba získať vstup,</li> <li>identifikovať vlastnosti vstupnej informácie (obmedzenia, rozsah, formát),</li> <li>rozpoznávať situácie, kedy treba zobrazíť výstup, realizovať akciu,</li> <li>zapisovať algoritmus, ktorý reaguje na vstup.</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>rozpoznávať, aká časť algoritmu sa má vykonať pred, počas aj po skončení cyklu,</li> <li>riešiť problémy, v ktorých treba výsledok získať akumulovaním čiastkových výsledkov v rámci cyklu.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie a podmienky, kedy treba použiť vetvenie,</li> <li>rozpoznávať, aká časť algoritmu sa má vykonať pred, v rámci a po skončení vetvenia,</li> <li>riešiť problémy, ktoré vyžadujú vetvenie so zloženými podmienkami (s logickými</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa jednoduchých programov (vytváranie testovacích dát),</li> <li>(LOG3) využitím logických zdôvodnení detegovať a opravovať chyby v programoch a algoritmoch (vytváranie testovacích dát),</li> <li>(LOG5) logicky zdôvodniť rozdelenie programu na menšie časti (testovanie jednotlivých situácií).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (systém automatického osvetlenia, výpočet sadzby za prenájom autobusu, výpočet výšky cestovného).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia - lineárne rozdeliť problémy na menšie časti tak, aby sa dali využiť na dosiahnutie cieľa (výpočet pre jednotlivé prípady).</li> </ul>



<p>spojkami),</p> <ul style="list-style-type: none"> <li>• riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> <p>Hľadanie a opravovanie chýb:</p> <ul style="list-style-type: none"> <li>• zisťovať, pre aké vstupy, v ktorých prípadoch, situáciách program zle pracuje.</li> </ul> <p>Programovací jazyk <b>Python</b>:</p> <ul style="list-style-type: none"> <li>• systematizovať a precvičiť prácu s riadiacimi štruktúrami jazyka <b>Python</b> (cykly, podmienené príkazy, funkcie s parametrami a návratovou hodnotou), rozpoznávať a odstraňovať chyby, navrhovať testovacie dáta.</li> </ul>	
<p><b>Riešený didaktický problém</b></p>	
<p>Výučba riadiacich štruktúr sa často realizuje na samoučelných úlohách bez prepojenia na reálny svet a reálne problémy. Úlohy sú často koncipované tak, aby sa riadiace štruktúry vzájomne nekombinovali – nevnárali. V tejto metodike žiaci riešia úlohy, ktoré nájdu zmysel aj v reálnom živote, mimo programovania. Takéto úlohy vyžadujú od žiakov skúsenosti s analýzou problémov, ktorá ich vedie k premyslenej dekompozícii úlohy na menšie, čiastkové podproblémy, ktoré sú programovo ľahšie riešiteľné. Dôraz kladieme aj na správnosť a komplexnosť riešení, v tejto fáze prostredníctvom premysleného navrhovania testovacích dát.</p>	
<p><b>Dominantné vyučovacie metódy a formy</b></p>	<p><b>Príprava učiteľa a pomôcky</b></p>
<ul style="list-style-type: none"> <li>• problémové vyučovanie,</li> <li>• frontálna a individuálna forma.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulochy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/10/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného,</li> <li>• <b>ucitel/testy/test2/</b> didaktický test, javová analýza testu, tabuľka pre vyhodnotenie testu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/10/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<p><b>Diagnostika splnenia vzdelávacích cieľov</b></p>	
<p>Výsledky žiackych riešení úloh z pracovného listu, výsledky testu II.</p>	

## Úvod

Toto je desiatu metodiku zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Primárnym cieľom tejto metodiky je systematizácia a precvičovanie učiva. Predpokladáme, že učivo z metodík 1. až 5. si už žiaci osvojili na dostatočnej úrovni. V tejto metodike sa zameriavame na funkcie s parametrami a s návratovou hodnotou, logické podmienky a vetvenie priebehu výpočtu, hľadanie chýb v programoch, navrhovanie testovacích dát a samotné testovanie programov.

Po absolvovaní tejto metodiky by mal učiteľ zaradiť hodinu určenú na preverenie žiackych vedomostí formou testu. Ukážka testu aj s riešením a s javovou analýzou je prílohou tejto metodiky.

## PRIEBEH VÝUČBY

Osnova prvej (opakovacej) vyučovacej hodiny:

- **Rýchla diagnostika vedomostí a zručností žiakov (10 minút)** – riešenie a diskusia o riešení úlohy.
- **Precvičovanie – samostatná práca (27 minút)** – riešenie modelovej úlohy a ďalších úloh z pracovného listu.
- **Zhrnutie (3 minúty)** – diskusia.

Druhá vyučovacia hodina je určená na riešenie testu žiakmi a diskusiu o žiackych riešeniach.

### RÝCHLA DIAGNOSTIKA VEDOMOSTÍ A ZRUČNOSTÍ ŽIAKOV (cca 10 minút)

Na úvod necháme žiakov, aby vyriešili Úlohu 1. Na základe diskusie žiakov o ich riešeniach, môžeme identifikovať problematické prvky učiva a venovať im zvýšenú pozornosť.

Úloha je zameraná na vytváranie a používanie funkcií s parametrami a s návratovou hodnotou, vytváranie podmienok a použité podmieneného vetvenia.

V druhej časti úlohy žiaci navrhujú testovacie dáta pre overenie správnosti riešenia. Situáciu komplikuje fakt, že do výpočtu zasahuje prvok náhodnosti (funkcia `intenzita_osvetlenia()`) a nie je možné predikovať výsledok. Predpokladáme, že žiaci na tento problém narazia. Neodporúčame „odstránenie“ náhody realizovať vo vytváratej funkcii `pocet_diody()`, lebo by sme testovali inú definíciu funkcie než tú, ktorá bude výsledná.

**Úloha 1** *V inteligentnej budove sa osvetlenie zapína automaticky, ak sa deteguje prítomnosť človeka a intenzita svietenia sa nastaví tak, aby boli splnené minimálne požiadavky na úroveň osvetlenia daného priestoru. Pre najnáročnejšie práce je potrebné osvetlenie aspoň 500 luxov.*

*Každá miestnosť obsahuje snímač úrovne osvetlenia. Na základe hodnoty zo snímača lampa v prípade potreby rozsvieti niekoľko zo svojich štyroch LED diód. Každá z diód generuje svetlo o intenzite 150 luxov. V súbore*

**inteligentna\_bodova.py:**

- a) Vytvorte funkciu `pocet_diod()`, ktorá pre zadanú minimálnu požadovanú úroveň osvetlenia vráti, koľko diód je potrebných rozsvietiť. Využite simulátor snímača vo funkcii `intenzita_osvetlenia()` v programe **inteligentna\_bodova.py**.
- b) Navrhnete vhodné testovacie dáta a otestujte svoju funkciu. V prípade potreby upravte časť programu.

```
def intenzita_osvetlenia():
    import random
    intenzita = random.randint(0, 600)
    return intenzita
```

**Riešenie:**

a)

```
def pocet_diod(min_osvetlenie):
    akt_intenzita = intenzita_osvetlenia()
    chybajuca_intenzita = min_osvetlenie - akt_intenzita
    if chybajuca_intenzita <= 0:
        return 0
    elif chybajuca_intenzita <= 150:
        return 1
    elif chybajuca_intenzita <= 300:
        return 2
    elif chybajuca_intenzita <= 450:
        return 3
    else:
        return 4
```

b)

Testovacie dáta, hodnotu premennej `min_osvetlenie`, by sme mali vybrať tak, aby sme otestovali všetky možnosti: 0 až 4 rozsvietené diódy. Funkciu `intenzita_osvetlenia()` môžeme pre potrebu testovania upraviť tak, aby vracala nejakú nami určenú hodnotu (napr. 30). V tomto prípade by testovacie dáta mohli byť napr.: 20, 170, 320, 470, 490. Otestovať môžeme aj hraničné hodnoty: 30, 180, 330, 480, 490. Nastaviť hodnotu na 0 nie je vhodné, pretože by sme neotestovali, či s hodnotou aktuálneho osvetlenie vo funkcii pracujeme alebo nie.

**PRECVIČOVANIE – SAMOSTATNÁ PRÁCA (CCA 27 MIN)**

V tejto časti hodiny žiaci pracujú samostatne. Učiteľ monitoruje prácu žiakov, v prípade potreby im poskytne konzultáciu alebo vysvetlí časť, v ktorej im niečo nie je jasné. Táto fáza hodiny nepredstavuje praktickú previerku, teda žiaci môžu svoje čiastkové problémy pri riešení v prípade potreby konzultovať aj so svojimi spolužiakmi.

Úloha 2 je zameraná na definovanie série podmienok pre rozpoznanie jedného z niekoľkých prípadov a na precvičenie si príkazu opakovania.

Úloha 3 je zameraná na precvičenie si kombinácie cyklu a podmieneného vetvenia.

Riešenie úlohy 4 vyžaduje definíciu vnorených podmienok. Riešenie je možné realizovať aj vytvorením zložených podmienok. Riešenie v tomto prípade ale nebude efektívne, lebo v zložených podmienkach budeme opakovane testovať rovnaké logické výrazy.

**Úloha 2** Pri objednávke autobusu na školský výlet sú stanovené nasledovné sadzby:

Sadzba 1: 0-50 km      0,50 € / km

Sadzba 2: 51-100 km      0,45 € / km

Sadzba 3: 101 km a viac      0,40 € / km

- Vytvorte program **autobus.py**, ktorý načíta prejdenu vzdialenosť a vypíše cenu za prenájom autobusu. Na výpočet prenájmu definujte funkciu `cena_za_prenajom()`, ktorá pre zadaný počet km vráti cenu prenájmu.
- Upravte funkciu `cena_za_prenajom()` tak, aby v prípade zápornej vzdialenosti funkcia vrátila „Chybne zadaná vzdialenosť“.
- Upravte definíciu funkcie `cena_za_prenajom()` tak, aby sa v prípade sadzby 2 účtovalo navyše štartovné vo výške 2 € a v prípade sadzby 3 štartovné vo výške 6 €.
- Vytvorte funkciu `cennik()`, ktorá vypíše ceny za prenájom postupne od 1 km až po 110 km.

Riešenie:

```
def cena_za_prenajom(vzdialenost): # riešenie A
    if vzdialenost <= 50:
        vysledok = vzdialenost * 0.5
    elif vzdialenost <= 100:
        vysledok = vzdialenost * 0.45
    else:
        vysledok = vzdialenost * 0.4
    return vysledok

def cena_za_prenajom(vzdialenost): # riešenie B
    if vzdialenost < 0:
        vysledok = 'Chybne zadaná vzdialenosť'
    elif vzdialenost <= 50:
        vysledok = vzdialenost * 0.5
    elif vzdialenost <= 100:
        vysledok = vzdialenost * 0.45
    else:
        vysledok = vzdialenost * 0.4
    return vysledok

def cena_za_prenajom(vzdialenost): # riešenie C
    if vzdialenost < 0:
        vysledok = 'Chybne zadaná vzdialenosť'
    elif vzdialenost <= 50:
        vysledok = vzdialenost * 0.5
    elif vzdialenost <= 100:
        vysledok = 2 + vzdialenost * 0.45
    else:
        vysledok = 6 + vzdialenost * 0.4
    return vysledok

def cennik(): # riesenie D
    for i in range(1, 111):
        print(i, 'km -', cena_za_prenajom(i), 'EUR')
```

**Úloha 3** Vytvorte funkciu `pocet_delitelov()`, ktorá pre zadané prirodzené číslo zistí a vráti počet jeho deliteľov. Riešenie uložte do súboru **delite.py**.

Riešenie:

```
def pocet_delitelov(cislo):
    pocet = 0
    for delitel in range(1, cislo + 1):
        if cislo % delitel == 0:
            pocet = pocet + 1
    return pocet
```

V riešení úlohy môžu žiaci použiť jednoduchú verziu príkazu `range`. V tom prípade je však potrebné upraviť hodnotu deliteľa:

```
for delitel in range(cislo):
    if cislo % (delitel + 1) == 0:
```

Niektorí žiaci môžu navrhnúť, že stačí testovať deliteľov po odmocninu z čísla a každého deliteľa započítať 2x. Tu je potrebné zvlášť ošetriť prípad, ak číslo je druhou mocninou prirodzeného čísla.

Odporúčame akceptovať aj tieto varianty riešenia.

**Úloha 4** Dopravný podnik stanovil nasledovné ceny časových lístkov:

typ\čas	< 30 min	>= 30 min
obyčajný	0,4 €	0,7 €
zľavnený	0,25 €	0,4 €

- Vytvorte program **dopravny\_podnik.py** a definujte funkciu `cena_dopravy()`, ktorá pre zadaný typ lístka a čas dopravy vráti cenu dopravy.
- Upravte funkciu `cena_dopravy()` tak, aby v prípade nesprávne zadaného typu lístka vrátila „Chybne zadaný typ lístka“.
- Upravte funkciu `cena_dopravy()` tak, aby v prípade záporného času vrátila „Chybne zadaný čas“.
- Navrhnite testovacie dáta pre otestovanie správnosti funkcie.

Riešenie:

```
def cena_dopravy(typ, cas): # riešenie A
    if typ == 'obyčajny':
        if cas < 30:
            vysledok = 0.4
        else:
            vysledok = 0.7
    else:
        if cas < 30:
            vysledok = 0.25
        else:
            vysledok = 0.4
    return vysledok
```

```
def cena_dopravy(typ, cas): # riešenie B
    if typ == 'obyčajny':
```

```

        if cas < 30:
            vysledok = 0.4
        else:
            vysledok = 0.7
    elif typ == 'zlavneny':
        if cas < 30:
            vysledok = 0.25
        else:
            vysledok = 0.4
    else:
        vysledok = 'Chybne zadany typ lístka'
    return vysledok

def cena_dopravy(typ, cas): # riešenie C
    if typ == 'obycajny':
        if cas < 0:
            vysledok = 'Chybne zadany čas'
        elif cas < 30:
            vysledok = 0.4
        else:
            vysledok = 0.7
    elif typ == 'zlavneny':
        if cas < 0:
            vysledok = 'Chybne zadany čas'
        elif cas < 30:
            vysledok = 0.25
        else:
            vysledok = 0.4
    else:
        vysledok = 'Chybne zadany typ lístka'
    return vysledok

```

Riešenie D: Testovacie dáta by sme mali navrhnúť tak, aby sme otestovali každú z možností. Je potrebné si uvedomiť, že navrhujeme dvojice dát (typ a čas). Okrem povolených hodnôt pre typ a čas by sme mali otestovať aj kombináciu povolených a nepovolených hodnôt. Napr.:

typ	obycajny	obycajny	zlavneny	zlavneny	polovicny	zlavneny	polovicny
čas	20	30	20	30	20	-5	-5

## ZHRNUTIE (CCA 3 MIN)

Vyzveme žiakov, aby sa vyjadrili, či mali pri samostatnej práci nejaké problémy s úlohami, čo bolo pre nich ťažké, resp. ktoré z ponúknutých programátorských problémov zvládli bez väčších komplikácií. Odporúčame, aby učiteľ stručne zhrnul kľúčové prvky, ku ktorým precvičovanie smerovalo.

# 11 REŤAZCE

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>pomocou premenných,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>používať premennú vo výrazoch,</li> <li>tabuľkový kalkulátor – výpočty v tabuľkách, relatívne odkazovanie,</li> <li>definovať vlastné funkcie,</li> <li>rozpoznávať situácie a podmienky, kedy treba použiť vetvenie,</li> <li>rozpoznávať situácie a podmienky, kedy treba použiť cyklus so známym počtom opakovaní.</li> </ul>	
<b>Ciele</b>	
Žiakom osvojované vedomosti a zručnosti	Žiakom rozvíjané spôsobilosti
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>používať matematické výrazy pri vyjadrovaní vzťahov,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti.</li> </ul> <b>Pomocou premenných:</b> <ul style="list-style-type: none"> <li>identifikovať zo zadania úlohy, ktoré údaje musia byť zapamätané, resp. sa menia,</li> <li>riešiť problémy, v ktorých si treba zapamätať a neskôr použiť zapamätané hodnoty vo výrazoch,</li> <li>zovšeobecniť riešenie tak, aby fungovalo nielen s konštantami.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba získať vstup,</li> <li>identifikovať vlastnosti vstupnej informácie (obmedzenia, rozsah, formát),</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa jednoduchých programov,</li> <li>(LOG4) vyvodzovať (logicky zdôvodňovať) závery z pozorovaní a experimentov (aj myšlienkových).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém/časti problému (postupnosti krokov na realizáciu nejakej činnosti vedúcej k cieľu, vytvárať scenáre a storyboardy, postup experimentu),</li> <li>(ALG7) vylepšovať existujúce algoritmy.</li> </ul> Vyhodnotenie: <ul style="list-style-type: none"> <li>(VYH3) posúdiť správnosť postupu na základe definovaných kritérií.</li> </ul>

<ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba zobrazíť výstup,</li> <li>zapisovať algoritmus, ktorý reaguje na vstup.</li> </ul> <p>Hľadanie a opravovanie chýb:</p> <ul style="list-style-type: none"> <li>rozlišovať chybu pri realizácii od chyby v zápise.</li> </ul> <p>Používať operácie, operátory a funkcie programovacieho jazyka Python pre prácu s reťazcami pri riešení problémov.</p>	
<b>Riešený didaktický problém</b>	
<p>Python ponúka nové, užitočné nástroje pre prácu s reťazcami. Pre učiteľa informatiky je často náročné opustiť už známe algoritmy využívané pri práci s reťazcami v iných programovacích jazykoch a preorientovať sa na možnosti Pythonu – máme pocit, že žiakov ochudobňujeme o priestor na premýšľanie a hľadanie týchto postupov. Využitie výrezov reťazca, operátora príslušnosti, reťazcových metód nám však umožňuje zamerať pozornosť žiakov na riešenie ostatných, nemenej zaujímavých podúloh riešeného problému.</p>	
<b>Dominantné vyučovacie metódy a formy</b>	<b>Príprava učiteľa a pomôcky</b>
<ul style="list-style-type: none"> <li>Bádateľská metóda (model 5E),</li> <li>frontálna a individuálna forma.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li><b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li><b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li><b>ucitel/pracovne_subory_riesenia/11/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li><b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li><b>ziak/pracovne_subory/11/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<b>Diagnostika splnenia vzdelávacích cieľov</b>	
Sebahodnotiaci test v pracovnom zošite.	



## Úvod

Žiaci už vedia, že doteraz používané vstupy sú v jazyku Python reprezentované vo forme reťazca. Tento vstup vieme vhodne pretypovať na reálne (`float`) alebo celé číslo (`int`), prípadne pretypovať číselnú hodnotu na reťazec (`str`). Pri výpise výstupu programu používame spôsob formátovania reťazcov „f-strings“, ktorý sa objavil vo verzii Pythonu 3.6. Vďaka nemu nemusíme číselné hodnoty pretypovať a kód je tak prehľadnejší.

V rámci tejto metodiky žiakom predstavíme nové operácie, operátory a funkcie na prácu s reťazcami. Vďaka tomu môžeme naše úlohy rozšíriť aj na zadania, ktoré nepracujú (len) s číselnými hodnotami. Niekedy sa aj úlohy, ktoré na prvý pohľad vyzerajú „matematicky“, riešia jednoduchšie s použitím reťazca. Budeme pri tom využívať už spomínané nové nástroje, ktoré nám umožnia riešiť úlohy elegantnejšie.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – diskusia so žiakmi na tému programovanie (doterajšie skúsenosti, ďalšie očakávania) spojená s prezentáciou programu, ktorý žiakom ukazuje ďalšie možnosti jazyka.
- **Skúmanie (10 minút)** – skúmanie ako Python pracuje s výstupmi a vstupmi programu v konzole (úloha 1).
- **Vysvetlenie (10 minút)** – vysvetlenie predchádzajúcich zistení, riešenie úloh (úlohy 2 a 3 z pracovného listu).
- **Rozpracovanie (10 minút)** – riešenie náročnejších úloh (úloha 4).
- **Vyhodnotenie (5 minút)** – vyriešenie sebahodnotiaceho testu, diskusia o odpovediach, zhrnutie nových poznatkov.

## ZAPOJENIE (CCA 5 MIN)

V tejto fáze pracovný list nepoužívame.

Úlohy, ktoré sme zatiaľ riešili, boli úzko prepojené na matematiku (aspoň čo sa týka matematického zápisu výpočtov vedúcich k riešeniu daného problému). Dnes prejdeme k úlohám, pri ktorých budeme pracovať s reťazcami (t. j. postupnosťami znakov) a tento reťazec budeme skúmať a upravovať. Aj v „reálnom“ svete aplikácie spracovávajú nielen číselné údaje, ale aj texty (mená, adresy, národnosť, ...).

K tomuto záveru žiakov navedieme pomocou diskusie, v ktorej zosumarizujeme doterajšie skúsenosti žiakov s chybami, ktoré už mali možnosť zažiť pri ladení a spúšťaní programov. Návrh otázok do diskusie:

- Aké typy údajov od vás požadujú programy/aplikácie, ktoré poznáte a používate? Sú to väčšinou číselné údaje alebo aj iné?
- Číselné údaje dokáže program spracovať prostredníctvom potrebného výpočtu. Ako môže program manipulovať s reťazcami?

Doposiaľ sme spracovávali číselné hodnoty. Naučili sme sa pretypovať (konvertovať) vstup vo formáte reťazca na celé alebo reálne číslo a vykonať s takto získanými hodnotami potrebné výpočty.

Ako programátori sme zatiaľ riešili problémy, pri ktorých sme manipulovali s číselnými vstupmi. Úmyselne sme sa vyhýbali úlohám, ktoré vyžadovali spracovanie reťazca. Ako zabezpečiť korektné zadanie priezviska používateľa (prvé písmeno veľké, ostatné malé, obsahuje len abecedné znaky)? Ako zo zadaného rodného čísla zistiť čo najjednoduchšie, či je zadané korektne, akého pohlavia je jeho majiteľ? Pri riešení týchto a podobných úloh budeme potrebovať funkcie, operácie, operátory a metódy pre prácu s reťazcami.

## SKÚMANIE (CCA 10 MIN)

Žiaci pracujú s pracovným listom a s programom **retazec.py**. Úlohou žiakov je spustiť program, sledovať jeho činnosť/výpis do konzoly a na základe tohto skúmania definovať činnosť jednotlivých príkazov programovacieho jazyka Python pre prácu s reťazcami.

Žiaci pracujú vo dvojiciach.

### Úloha 1 Otvorte program **retazec.py**.

1	<code>meno = 'Juraj'</code>	reťazec/hodnotu 'Juraj' pomenujeme názvom meno
2	<code>priezvisko = 'Bok'</code>	reťazec/hodnotu 'Bok' pomenujeme názvom priezvisko
3	<code>spolu = meno + ' ' + priezvisko</code>	zreťazíme/spojíme tri reťazce do 'Juraj Bok' a výsledok pomenujeme názvom spolu
4	<code>print(spolu)</code>	výpis reťazca spolu – 'Juraj Bok'
5	<code>print(len(spolu))</code>	výpis dĺžky reťazca spolu – 9
6	<code>print(spolu[4])</code>	výpis 4-teho znaku reťazca - 'j'
7	<code>print(spolu[2:5])</code>	výpis časti reťazca určenej indexami 2 .. 5 – 'raj'
8	<code>print(spolu[6:])</code>	výpis časti reťazca určenej indexami 6 .. koniec – 'Bok'
9	<code>if 'raj' in spolu:</code>	test, či reťazec 'raj' je v spolu – True
10	<code>    print('áno')</code>	výpis 'áno'
11	<code>else:</code>	
12	<code>    print('nie')</code>	
13	<code>for znak in spolu:</code>	cyklus cez znaky reťazca spolu
14	<code>    print(znak)</code>	výpis znakov reťazca spolu, každý samostatne

Program spustíte viackrát. Skúšajte meniť reťazce/znaky, čísla, operácie s reťazcami a sledujte výpisy programu do konzoly. Na základe svojich pokusov vyplňte prázdny stĺpec tabuľky, v ktorom vysvetlíte, čo je výsledkom jednotlivých príkazov.

## VYSVETLENIE (CCA 10 MIN)

Na základe predchádzajúceho skúmania by žiaci mali formulovať nové poznatky, ktoré počas neho získali. Žiaci formulujú svoje zistenia, učiteľ ich usmerňuje, koriguje ich odpovede novými otázkami alebo protiargumentmi.

### Poznámka:

Vo fáze skúmania sme žiakom sprostredkovali niekoľko nových operátorov a funkcií na prácu s reťazcami. Ostatné im podáme formou výkladu vo fáze vysvetlenie.

### Poznámka:

Ak žiaci používajú pri písaní programového kódu slovenskú klávesnicu, symbol pre apostrof napíšu pomocou klávesovej skratky ALT + 39 (na numerickej klávesnici).

Pri práci s operátormi a funkciami pre prácu s reťazcami je potrebné upozorniť žiakov na to, že nimi nemeníme obsah reťazca, s ktorým pracujeme. Výsledok operácie alebo funkcie buď prepojíme s novou premennou (alebo s premennou, ktorá bola prepojená s pôvodným reťazcom) alebo ho použijeme ako parameter ďalšej funkcie.

Pri zreťazení viacerých reťazcov sme použili operáciu súčtu (+). Táto operácia však v tomto prípade nie je komutatívna (na rozdiel od súčtu čísel).

Dĺžka reťazca je návratovou hodnotou funkcie `len()`.

### Relačné operátory

Do časti skúmanie sme nezaradili relačné operátory (<, >, <=, >=, ==, !=). Žiakom vysvetlíme, že reťazce môžeme porovnávať, pričom skúmanie ich vzájomnej relácie vychádza z abecedy tak, ako sme zvyknutí, napr. pri práci so slovníkom. Uvedomme si, že znaky veľkej abecedy sú v tabuľke UTF-8 pred znakmi malej abecedy ('A' < 'a').

V skúmanom programe sme v riadku 9 použili tzv. operátor príslušnosti `in` v tvare `retazec1 in retazec2`. Operátor vráti hodnotu `True`, ak sa `retazec1` nachádza v `retazec2` (je jeho podreťazcom), inak vráti hodnotu `False`. (V riadku 13 nie je použitý operátor príslušnosti, je to zápis cyklu `for`.)

Zaujímavým zistením je, že jednotlivým znakom reťazca prislúcha index a pomocou operácie indexovania vieme k týmto znakom podľa potreby pristupovať. Pre žiakov bude zo začiatku ťažké stotožniť sa s tým, že prvý znak má index 0. Posledný znak reťazca má index o 1 menší ako je dĺžka reťazca (dĺžku reťazca získame ako návratovú hodnotu funkcie `len()`).

Zaujímavé je, že môžeme použiť aj záporné hodnoty indexu – vtedy pristupujeme k znakom reťazca od konca. Ak zadáme hodnotu indexu mimo prípustných hodnôt, program vypíše chybovú správu `IndexError: string index out of range`.

Príklad: `retazec = 'Informatika':`

<b>l</b>	<b>n</b>	<b>f</b>	<b>o</b>	<b>r</b>	<b>m</b>	<b>a</b>	<b>t</b>	<b>i</b>	<b>k</b>	<b>a</b>
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Žiaci pri skúmaní objavili, ako prechádzať jednotlivými znakmi reťazca. V prípade použitia zápisu `for znak in spolu` prechádzame pomocou premennej `znak` celým reťazcom, premenná `znak` postupne odkazuje na jednotlivé znaky reťazca. Pozor, `in` v tomto kontexte nie je v pozícii vyššie spomenutého operátora príslušnosti.

### Výrezy

Pri skúmaní žiaci pracovali s výrezmi. Pri vysvetlení je vhodné žiakom ukázať princíp práce, nie všetky možnosti, až nuansy. Uvádzame ich v metodike skôr pre potrebu učiteľa, aby vedel reagovať na správanie sa programu, ak žiaci budú s výrezmi experimentovať.

Niekedy potrebujeme pracovať nie s celým reťazcom, ale s jeho podreťazcom – výrezom (anglicky *slice*), napr. z reťazca reprezentujúceho dátum narodenia potrebujeme získať rok narodenia, t. j. posledné štyri znaky; z reťazca reprezentujúceho zašifrovaný text potrebujeme získať každý druhý znak. Operáciu indexovanie vieme použiť aj v tomto kontexte – na prístup nie len ku konkrétnemu znaku, ale aj na prístup k tzv. **výrezom skúmaného reťazca**. Formálny zápis tejto operácie má tvar `retazec[zaciatok:koniec]`, kde hodnota *zaciatok* určuje prvý znak výrezu, hodnota *koniec* určuje posledný znak výrezu – znak s indexom *koniec* - 1.

Ak neuvedieme prvý index (*zaciatok*), výrez začne od začiatku reťazca. Ak neuvedieme druhý index (*koniec*), výrez skončí posledným znakom reťazca. Ak neuvedieme ani jeden index, budeme pracovať s celým reťazcom:

<code>retazec[:]</code> => Informatika	<code>retazec[5:10]</code> => matik
<code>retazec[0:11]</code> => Informatika	<code>retazec[-10:-3]</code> => nformat
<code>retazec[5:]</code> => matika	<code>retazec[: -4]</code> => Informa
<code>retazec[:4]</code> => Info	<code>retazec[-4:]</code> => tika

### Poznámka:

Pri prístupe k výrezom reťazca nemiešajme dokopy indexovanie so zápornými a nezápornými indexmi.

Pri použití indexovania zápornými indexmi sa nám nikdy nepodarí vypísať pôvodný reťazec:

```
retazec[-11:-1] => Informatik
retazec[-11:0] => prázdny reťazec
```

Už vieme, že ak sa pokúsime pracovať so znakom reťazca a zadáme neexistujúci index (v našom prípade napr. `retazec [44]`), program skončí s chybovou správou. Zvláštnosťou je, že ak

použijeme neexistujúci index pri definovaní výrezu, operácia prebehne bez chyby a neexistujúci index bude nahradený prvým, resp. posledným možným.

```
retazec[-100:-1] => Informatik
```

```
retazec[0:110] => Informatika
```

Ak sa pokúsime pracovať s výrezom, ktorý neexistuje (napr. `retazec[13:10]`), operácia prebehne opäť bez chyby a jej výsledkom bude prázdny reťazec.

Výrezy môžeme vytvárať aj tak, že určíme aj **krok indexov**: `retazec[zaciatok:koniec:krok]` – hovoríme o výrezoch s krokom (rozšírených výrezoch).

Hodnota *krok* určuje hodnotu, o ktorú sa budeme pri výbere znakov výrezu presúvať medzi znakmi vstupného reťazca.

	<b>l</b>	<b>n</b>	<b>f</b>	<b>o</b>	<b>r</b>	<b>m</b>	<b>a</b>	<b>t</b>	<b>i</b>	<b>k</b>	<b>a</b>
	0	1	2	3	4	5	6	7	8	9	10
	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Príklady výrezov – odporúčame ukázať žiakom aspoň prvé tri, tučným písmom zvýraznené príklady:

```
slovo[::2] => Ifraia
slovo[::3] => Ioak
slovo[::-1] => akitamrofni
slovo[9:1:-2] => ktmo
slovo[1:5:-3] => prázdny reťazec
slovo[-11:-1:2] => Ifrai
```

**Úloha 2** Zuzka sa odsťahovala s celou rodinou do Kanady. So svojou najlepšou kamarátkou Katkou komunikujú písomne a na utajenie svojich správ si dohodli šifru: do textu správy vložia za každý znak ľubovoľný znak. Takto upravená správa vyzerá ako motanica nezmyselných slov, napr. text Ahoj Zuzka! po zašifrovaní vyzerá takto: A\*huoXjj QZ8uyzKk+a,!{.

Aby sa im správy ľahšie dešifrovali, obidve vytvorili vlastnú funkciu `desifruj()`, ktorej návratovou hodnotou je dešifrovaná správa. Každá z funkcií však vyzerá odlišne, dievčence sa nevedia dohodnúť, ktorá je správna. Pomôžte im pri rozhodovaní – určte, ktorá z funkcií plní danú úlohu.

Zuzkina funkcia	Katkina funkcia
<pre>def desifruj(s):     vysledok = ''     for i in range(0, len(s), 2):         vysledok = vysledok + s[i]     return vysledok</pre>	<pre>def desifruj(s):     vysledok = s[::2]     return vysledok</pre>

Riešenie:

Správne pracujú obidve funkcie. Líšia sa postupom, akým sa dostanú k správne mu výstupu. Zuzkina funkcia prechádza reťazcom a hodnotu, na ktorú odkazuje premenná `vysledok`, postupne mení tak, že k nej pripája každý nultý, druhý, štvrtý, ... znak pôvodného.

Katkina funkcia využíva operáciu indexovania, presnejšie prácu s výrezmi vstupného reťazca. Do premennej

`vysledok` sa priradí každý druhý znak počnúc nultým až po koniec reťazca.

Žiaci poznajú funkciu `range()` v jej najjednoduchšom zápise: `range(stop)`. Takto zapísaná funkcia označuje postupnosť 0, 1, ..., `stop - 1`. Funkcia má však univerzálnejšie použitie. Pri zápise v tvare `range(start, stop)` označuje postupnosť `start`, `start + 1`, ..., `stop - 1`. A napokon pri zápise `range(start, stop, krok)` označuje postupnosť `start`, `start + krok`, `start + 2 * krok`, ..., ktorej posledný člen je menší ako hodnota prepojená s premennou `stop`.

Prečo v Zuzkinej funkcii musíme najprv premennú `vysledok` prepojiť s prázdny reťazcom? Prečo v Katkinej funkcii tento úkon urobiť nemusíme? V Zuzkinej funkcii pri prvom vykonaní príkazu `vysledok = vysledok + s[i]` sa Python snaží pri vyhodnotení pravej strany nájsť hodnotu, na ktorú odkazuje premenná `vysledok`. Ak by neexistovala, funkcia by skončila chybovou správou.

Samozrejme, nevieme ošetriť to, či šifrovanie naozaj prebehlo bez chyby (ak sme pri šifrovaní nejaký znak vynechali a nezašifrovali; nezašifrovali sme všetky znaky a náhodou je dĺžka výsledného reťazca párne číslo; omylom sme vložili znak nie za, ale pred znak odkazu, ...). Tieto špecifické situácie ošetriť nedokážeme.

**Úloha 3** Vytvorte program **palindrom.py**, ktorý zistí, či slovo zadané na vstupe je palindróm. Slovo zadávame malými písmenami, nepoužívame medzery a diakritiku.

*Poznámka: Palindróm je slovo, veta, číslo (všeobecne akákoľvek postupnosť symbolov), ktorá má tú vlastnosť, že pri čítaní zľava doprava alebo sprava doľava znie rovnako.*

*Riešenie:*

```
retazec = input('Zadajte reťazec: ')

if retazec == retazec[::-1]:
    print(f'Reťazec {retazec} je palindrómom.')
else:
```

```
print(f'Reťazec {retazec} nie je palindrómom.')
```

## ROZPRACOVANIE (CCA 10 MIN)

Na záver žiakom predstavíme ďalší spôsob šifrovania textu – jazykovú hru Pig Latin. Umožní nám precvičiť manipuláciu s indexmi a výrezmi reťazca.

**Úloha 4** Tajomstvo komunikácie Zuzky a Katky odhalil Katkin brat Miško. Preto sa dievčatá rozhodli, že budú komunikovať po anglicky a zároveň budú používať šifru Pig Latin – jazykovú hru, ktorá slúži na pobavenie, aj na utajenie komunikácie pred nepovolanými osobami. Princíp hry spočíva v úprave slov podľa týchto pravidiel

- Ak slovo začína spoluhláskou, táto sa presunie na koniec slova a za ňu sa pridá prípona –ay, napr. door => oorday, pen => enpay.
- Ak slovo začína samohláskou, pridá sa len prípona –way, napr. apple =>appleway, old => oldway.

Vytvorte program **pig\_latin.py**, ktorý na vstupe dostane slovo (zapísané malými písmenami anglickej abecedy) a do konzoly vypíše toto slovo upravené podľa pravidiel jazykovej hry Pig Latin.

Riešenie:

Ako zistiť, ktoré pravidlo máme aplikovať na vstupné slovo? Stačí otestovať, či sa prvý znak slova rovná niektorej zo samohlások (a, e, i, o, u, y) a urobiť príslušné úpravy reťazca:

```
def sifruj(retazec):  
    # prvý znak reťazca je samohláska  
    if retazec[0] in 'aeiou':  
        vysledok = retazec + 'way'  
    # prvý znak je spoluhláska  
    else:  
        vysledok = retazec[1:] + retazec[0] + 'ay'  
    return vysledok  
  
slovo = input('Vstupné slovo: ')  
print(sifruj(slovo))
```

Zatiaľ nevieme zistiť, či zadaný reťazec obsahuje len malé písmená anglickej abecedy. Touto problematikou (reťazcové metódy) sa budeme zaoberať v nasledujúcej metodike.

## HODNOTENIE (cca 5 min)

V záverečnej časti požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal a nie klasifikácia známku či iný „postih“, ak ho nevyplnia správne. Pre učiteľa a žiaka zvlášť je cenná pravdivá informácia o úrovni osvojených poznatkov než umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, napr. na začiatku nasledujúcej hodiny.

### Sebahodnotiaci test

1.	<p>Nasledujúci program dešifruje vstupnú správu, ktorá vznikla podľa tohto pravidla šifrovania – pred a za každý znak správy bol vložený jeden náhodný znak (napr. správa „TAJNÉ“ je zašifrovaná v tvare „aTčýA7iJ8žN89Éí“). Doplňte chýbajúcu časť kódu tak, aby bol program funkčný pre ľubovoľnú správne zašifrovanú správu.</p> <pre> zasifrovaná_správa = input('Zašifrovaná správa: ') odsifrovaná_správa = zasifrovaná_správa[1]:[3] # alebo odsifrovaná_správa = zasifrovaná_správa[1]:len(zasifrovaná_správa):[3] print(f'{zasifrovaná_správa} =&gt; {odsifrovaná_správa}')</pre>
2.	<p>Aký výstup bude mať nasledujúci program, ak mu na vstupe zadáme slovo 'Zima'?</p> <pre> retazec = input('Vstupný reťazec: ') podretazec = retazec[-1] + retazec[::-1] + retazec[0] print(podretazec)</pre> <p>Výstup programu pre vstupné slovo 'Zima' je 'aamiZZ'</p>

Odporúčame, aby učiteľ uviedol správne odpovede. V prípade druhej úlohy testu je vhodné so žiakmi prediskutovať odpoveď na otázku, čo je výstupom programu.

Na záver zhrnieme nové poznatky:

- Pri práci s reťazcami alebo so znakmi môžeme používať:
  - operáciu `+`,
  - operátor príslušnosti `in`,
  - funkciu `len()`, ktorej návratovou hodnotou je dĺžka reťazca.
- K jednotlivým znakom daného reťazca môžeme pristupovať pomocou indexov (nekladných alebo záporných), k podreťazcom daného reťazca môžeme pristupovať pomocou výrezov, resp. výrezov s krokom.
- Reťazcom môžeme prechádzať aj pomocou cyklu `for` – buď ním prechádzame indexmi alebo znakmi reťazca.



## 12 REĽAZCOVÉ METÓDY, ZLOŽENÉ A VNORENÉ PODMIENKY

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>pomocou premenných,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>rozdeliť problém na podproblémy a na ich riešenie definovať vhodné funkcie,</li> <li>používať premennú vo výrazoch,</li> <li>používať príkazy cyklu a vetvenia.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>popisovať očakávané výstupy, výsledky, akcie.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať matematické výrazy pri vyjadrovaní vzťahov,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>identifikovať vlastnosti vstupnej informácie (obmedzenia, rozsah, formát).</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých treba výsledok získať akumulovaním čiastkových výsledkov v rámci cyklu.</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>dopĺňať, dokončovať, modifikovať rozpracované riešenie,</li> <li>uvažovať o rôznych riešeniach, navrhovať vylepšenie.</li> </ul> <b>Hľadanie a opravovanie chýb:</b> <ul style="list-style-type: none"> <li>rozpoznávať, kedy program pracuje nesprávne,</li> <li>hľadať chybu vo vlastnom, nesprávne pracujúcom programe a opraviť ju,</li> <li>zisťovať, pre aké vstupy, v ktorých prípadoch, situáciách program zle pracuje,</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa algoritmov,</li> <li>(LOG4) vyvodzovať (logicky zdôvodňovať) závery z pozorovaní a experimentov,</li> <li>(LOG6) logicky zdôvodniť zmenu algoritmu/programu.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém/časti problému (postupnosti krokov na realizáciu nejakej činnosti vedúcej k cieľu, vytvárať scenáre a storyboardy, postup experimentu),</li> <li>(ALG7) vylepšovať/dotvárať existujúce algoritmy.</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) rozdeliť veci do viacerých úrovní (aj podproblémy rozdeliť na menšie podpodproblémy atď.)</li> </ul>

- uvádzať kontra príklad, kedy niečo neplatí, nefunguje,
- posudzovať a overovať správnosť riešenia (svojho aj cudzieho),
- rozlišovať chybu pri realizácii od chyby v zápise.

Používať operácie, operátory a funkcie programovacieho jazyka Python pre prácu s reťazcami pri riešení problémov.

### **Riešený didaktický problém**

Nadväzujeme na predchádzajúcu metodiku a pomocou nových nástrojov – reťazcových metód – sa orientujeme na riešenie staronových, ale aj nových problémov. Táto téma je náročná najmä pre učiteľa, pretože v tejto chvíli musí „zabudnúť“ na postupy z iných programovacích jazykov a využiť možnosti programovacieho jazyka Python. Zároveň ide o prvý kontakt s objektom – v tomto prípade reťazcom – a jeho metódou.

Pri riešení úloh sa zameriame aj na vhodné použitie zložených a vnorených podmienok tak, aby boli žiaci schopní posúdiť vhodnosť použitia týchto podmienok.

### **Dominantné vyučovacie metódy a formy**

- Bádateľská metóda (model 5E),
- frontálna a individuálna forma.

### **Príprava učiteľa a pomôcky**

Pre učiteľa:

- **ucitel/programovanie\_v\_pythone.pdf** metodika vyučovania,
- **ucitel/pracovny\_zosit\_riesene\_ulohy.docx** pracovný zošit a riešenia úloh,
- **ucitel/pracovne\_subory\_riesenia/12/** riešené pracovné úlohy, tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu, zoznam reťazcových metód.

Pre žiaka:

- **ziak/pracovny\_zosit.docx** pracovný zošit,
- **ziak/pracovne\_subory/12/** pracovné súbory pre žiaka, zoznam reťazcových metód.

Použitie digitálnych nástrojov: NUTNÉ

### **Diagnostika splnenia vzdelávacích cieľov**

Sebahodnotiaci test v pracovnom zošite.



EURÓPSKA ÚNIA  
Európsky sociálny fond  
Európsky fond regionálneho rozvoja



OPERAČNÝ PROGRAM  
ĽUDSKÉ ZDROJE



MINISTERSTVO  
ŠKOLSTVA, VEDY,  
VÝSKUMU A ŠPORTU  
SLOVENSKEJ REPUBLIKY



it akadémia

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu  
v rámci Operačného programu Ľudské zdroje

[www.minedu.sk](http://www.minedu.sk) [www.employment.gov.sk/sk/esf/](http://www.employment.gov.sk/sk/esf/) [www.itakademia.sk](http://www.itakademia.sk)

---

## Úvod

Cieľom metodiky je nadviazať na predchádzajúcu vyučovaciu hodinu a ukázať žiakom nové nástroje pre prácu s reťazcami – reťazcové metódy. Umožní nám to efektívnejšie riešiť problémy, pri ktorých by sme inak museli konštruovať zložité algoritmy. Ak je na to priestor a čas, je pre žiaka určite prínosné, ak pri každej úlohe položí učiteľ otázku, ako by sme ju riešili bez využitia reťazcových metód. Ak to nie je možné, môžeme sa tejto téme venovať v rozširujúcom kurze, napr. u maturantov z informatiky, ktorí by určite mali mať nadhľad a nemali by byť pevne zviazaní s konkrétnym programovacím jazykom.

Aj keď žiaci nepoznajú terminológiu objektového programovania, používame v metodike pojem metóda. Iným pomenovaním upozorníme aj na iný spôsob použitia (porovnajme použitie funkcie, napr. `len(reťazec)` a použitie metódy `reťazec.isalpha()`). Aj keď je správna terminológia dôležitá, nebudeme na nej zbytočne bazírovať.

Pri riešení niektorých úloh budeme používať zložené, resp. vnorené podmienky. Ich správne použitie a zápis môže sprehľadniť zápis programu a jeho efektívnosť.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – diskusia so žiakmi spracovaní reťazcových hodnôt.
- **Skúmanie (10 minút)** – skúmanie metód pre prácu s reťazcom (úlohy 2 a 3).
- **Vysvetlenie (10 minút)** – vysvetlenie predchádzajúcich zistení, riešenie úloh 4 a 5.
- **Rozpracovanie (10 minút)** – riešenie úlohy 6.
- **Vyhodnotenie (5 minút)** – vyriešenie sebahodnotiaceho testu, diskusia o odpovediach, zhrnutie nových poznatkov.

## ZAPOJENIE (CCA 5 MIN)

Cieľom tejto fázy je ukázať žiakom dva rôzne prístupy k riešeniu problému:

1. Riešime problém pomocou vedomostí a zručností, ktoré sme doposiaľ získali.
2. Hľadáme nový, existujúci nástroj, pomocou ktorého vyriešime úlohu jednoduchšie, efektívnejšie alebo prehľadnejšie.

Žiakom predkladáme jednotlivé problémy a vyzývame ich, aby navrhli postup ich riešenia, hľadali úskalia navrhnutého riešenia, prípadne jeho nedostatky (neprogramujeme). Následne môžu žiaci opísať iné problémy, pri ktorých je potrebné manipulovať s reťazcami podľa istých kritérií.

**Úloha 1** Analyzujte nasledovné problémy a navrhnite postup ich riešenia.

1. Pri analýze vstupných údajov aplikácie pre zadávanie sťažností sa zistilo, že používatelia často omylom stlačia kláves CapsLock. Výsledný text teda obsahuje namiesto malých písmen veľké a namiesto veľkých písmen malé. Navrhnite program, ktorý v zadanom vstupnom reťazci zamení malé písmená za veľké a naopak.
2. Jedným zo vstupných údajov programu je meno používateľa v tvare Meno (prvé písmeno mena je veľké, nasledujúce písmená sú malé, reťazec obsahuje len znaky anglickej abecedy). Navrhnite postup, ktorým by ste overili, či vstupný reťazec spĺňa dané podmienky. Ako by ste vstupný reťazec upravili tak, aby spĺňal prvé dve podmienky?

Má vaše riešenie nejaké úskalia alebo nedostatky?

Uvedte iné problémy, pri ktorých je potrebné manipulovať s reťazcami podľa istých kritérií.

Riešenie:

*Možný návrh riešenia problému 1:*

Využijeme znalosť ordinálnych hodnôt abecedných znakov a to, že vieme, aký je posun medzi malým a príslušným veľkým písmenom. V cykle prejdeme po znakoch celý reťazec, znaky budeme postupne „presúvať“ do nového reťazca upraveného podľa týchto pravidiel:

- ak je znak neabecedný, neupravujeme ho,
- ak je znak malé písmeno, zmeníme ho na veľké (cez ordinálnu hodnotu znaku),
- ak je znak veľké písmeno, zmeníme ho na malé (cez ordinálnu hodnotu znaku).

*Možný návrh riešenia problému 2:*

Vstupný reťazec budeme prechádzať od začiatku a kontrolovať podmienky:

- prvý znak (s indexom 0) je z množiny 'A'..'Z',
- všetky nasledujúce znaky až po posledný znak reťazca sú z množiny 'a'..'z'.

Ak sa rozhodneme reťazec upraviť, po skontrolovaní, či obsahuje len abecedné znaky, ho upravíme podobným spôsobom ako pri riešení problému 1.

Diskusia má smerovať k uvedomeniu si dôkladného rozboru každého problému a k použitiu funkcií a operácií, ktoré vieme pri reťazcoch zatiaľ použiť.

## SKÚMANIE (CCA 5 MIN)

Žiaci pracujú s pracovným listom a s programami **metody1.py** a **metody2.py**. Úlohou žiakov je spustiť program, sledovať jeho činnosť/výpis do konzoly a na základe tohto skúmania definovať činnosť konkrétnych reťazcových metód. Žiaci pracujú samostatne.

**Úloha 2** Otvorte program **metody1.py**. Program spustíte viackrát pre rôzne reťazce obsahujúce rôzne znaky.

```
vstup = 'VOLÁM SA JOŽKO MRKVIČKA, MÁM 15 ROKOV'

vystup = vstup.swapcase()
print(f'{vstup} => {vystup}')
```

Čo je výsledkom reťazcovej metódy `reťazec.swapcase()`?

Riešenie:

Výsledkom je kópia vstupného reťazca – je upravená tak, že malé písmená sú veľké a veľké sú malé a je pomenovaná názvom `vystup`.

**Úloha 3** Otvorte program **metody2.py**. Program spustíte viackrát pre rôzne reťazce.

```
meno = input('Zadajte svoje meno: ')

if meno.isalpha() and meno[0].isupper():
    print(f'Zadané meno je v poriadku.')
else:
    print(f'Zadané meno nie je v poriadku.')
```

- Pre ktoré reťazce program vypíše vetu 'Zadané meno je v poriadku.'?
- Pre ktoré reťazce program vypíše vetu 'Zadané meno nie je v poriadku.'?
- Čo je výsledkom príkazu `reťazec.isalpha()`?
- Čo je výsledkom príkazu `reťazec.isupper()`?

Riešenie:

- Pre ktoré reťazce program vypíše vetu 'Zadané meno je v poriadku.'?  
Pre reťazce, ktoré spĺňajú dve podmienky: všetky znaky reťazca sú abecedné a prvé písmeno je veľké.
- Pre ktoré reťazce program vypíše vetu 'Zadané meno nie je v poriadku.'?

- Pre reťazce, ktoré nespĺňajú aspoň jednu z podmienok uvedených pri prvej otázke.*
- Čo je výsledkom príkazu `retazec.isalpha()`?  
Výsledkom je hodnota `True`, ak skúmaný reťazec obsahuje len abecedné znaky. Inak je výsledkom `False`.
  - Čo je výsledkom príkazu `retazec.isupper()`?  
Výsledkom je hodnota `True`, ak skúmaný reťazec (v tomto prípade prvý znak reťazca) nie je prázdny a pozostáva len z veľkých znakov. Inak je výsledkom hodnota `False`.

## VYSVETLENIE (CCA 10 MIN)

Na predchádzajúcej hodine sme sa zoznámili s funkciami a operátormi pre prácu s reťazcami. Na dnešnej hodine budeme pracovať s reťazcovými metódami. Ich zápis je odlišný od zápisu funkcií – začína menom premennej odkazujúcej na príslušný reťazec (alebo samotným reťazcom), pokračuje bodkou a názvom metódy. Ide vlastne o špeciálny zápis volania funkcie.

V časti Skúmanie sme žiakom predstavili reťazcové metódy – v diskusii zistíme, aké sú zistenia žiakov.

Okrem toho sme v druhej úlohe použili logickú spojku `and`. Žiakom uvedieme aj logickú spojku `or`, s logickým operátorom `not` sa už stretli v predchádzajúcej metodike. Popis týchto operátorov nájdú žiaci v pracovnom liste v časti Vedomosti v kocke.

Programy, ktoré žiaci skúmali, ukazujú iný prístup k riešeniu problémov, o ktorých sme diskutovali v časti Zapojenie. Môžeme tak porovnať žiakmi navrhnuté riešenia s novým nástrojom – reťazcovými metódami.

### Ako používať reťazcové metódy

Je veľmi dôležité žiakom neustále zdôrazňovať, že metóda nemení pôvodný reťazec (reťazce sú nemenné), vytvára jeho upravenú kópiu, ktorú musíme priradiť nejakej premennej (`vystup = retazec.upper()`) alebo použiť ako parameter niektorej funkcie (napr. `print(retazec.isupper())`).

Po diskusii upozorníme žiakov na zoznam niekoľkých (z nášho pohľadu pre žiaka užitočných) reťazcových metód s popisom ich činnosti v časti „Vedomosti v kocke“. Nevyžadujeme ich znalosť, skôr aby žiaci pochopili princíp ich použitia a vedeli ich vhodne používať.

### Poznámka:

Podrobný a úplný popis reťazcových metód:

<https://docs.python.org/3/library/stdtypes.html?highlight=swapcase#string-methods>.

**Úloha 4** Otvorte program **pismena.py**. Doplňte tento program tak, aby ku každému znaku vstupného reťazca vypísal veľké písmeno ekvivalentné danému znaku. Vhodnú metódu použite namiesto zápisu `'...'` v príkaze `print()`. Môžete predpokladať, že na vstupe je reťazec pozostávajúci len z malých a veľkých písmen.

Riešenie:

```
vstup = input('Vstupný reťazec: ')
```

```
for znak in vstup:
    print(f'{znak} => {znak.upper()}')
```

**Úloha 5** S využitím skúseností z predchádzajúcej úlohy riešte nasledujúcu úlohu:

Vytvorte program **uprava.py**, ktorý dostane na vstupe meno používateľa. Podmienkou je, aby vstupný reťazec obsahoval len písmená. Program skontroluje, či zadaný vstup obsahuje len písmená – ak nie, vypíše oznam o chybnom vstupe. Následne upraví vstupný reťazec tak, aby prvé písmeno bolo veľké a všetky ostatné malé.

Riešenie:

Pomocou metódy `isalpha()` zistíme, či vstup obsahuje len písmená. Ak je to tak, vyskladáme nový reťazec. Na jeho začiatku bude veľké písmeno, ktoré „vytvoríme“ z prvého znaku vstupu (s indexom 0) pomocou metódy `upper()`. Nasledovať bude podreťazec vstupu – výrez od druhého znaku až do konca, ktorý pomocou metódy `lower()` upravíme tak, aby všetky jeho písmená boli malé. Túto metódu vyhladáme so žiakmi v pomocnom materiáli a spoločne ju zakomponujeme do programu.

```
vstup = input('Vstupný reťazec: ')
if not vstup.isalpha():
    print('Vstup má obsahovať iba písmená.')
else:
    vystup = vstup[0].upper() + vstup[1:].lower()
    print(f'{vstup} => {vystup}')
```

## ROZPRACOVANIE (CCA 10 MIN)

**Úloha 6** Škola v rámci svojho informačného systému ponúka žiakom vlastný komunikačný kanál. Žiak pri registrácii zadá svoju prezývku a vek. Ak ich zadal správne, získa prístup a môže komunikovať s ostatnými žiakmi školy. Vek je prirodzené číslo v rozsahu od 10 do 20, pre prezývku platia podmienky:

- na začiatku prezývky je podreťazec 'ZIAK',
- za podreťazcom 'ZIAK' nasledujú už len číslice, pričom číselnú časť tvoria aspoň 3, najviac však 8 číslic.

Vytvorte program **registracia.py**, ktorý otestuje vstupné hodnoty podľa uvedených podmienok.

Riešenie:

Riešenie úlohy je vhodné rozdeliť na menšie podproblémy: test prezývky a test veku. Pre každý test vytvorme samostatnú funkciu. Riešenie tak bude prehľadnejšie.

```
def test_prezyvka(prezyvka):
    # testujeme, či na začiatku je podreťazec ZIAK
    if prezyvka[:4] != 'ZIAK':
        return False
    # testujeme, či sú ďalšie znaky číslice
    if not prezyvka[4:].isnumeric():
        return False
    # testujeme dĺžku číselnej časti
    if not 3 <= len(prezyvka[4:]) <= 8:
        return False
    return True
```

```
def test_vek(vek):  
    vek = int(vek)  
    # testujeme, či je splnená podmienka pre vek  
    return 10 <= vek <= 20  
  
prezývka = input('Zadajte svoju prezývku: ')  
# testujeme správnosť prezývky  
if test_prezývka(prezývka):  
    # ak je prezývka v poriadku, vypýtame si vek  
    vek = input('Váš vek: ')  
    # testujeme správnosť veku  
    if test_vek(vek):  
        # aj vek je v poriadku  
        print('Registrácia bola úspešná.')  
    else:  
        # vek nie je v zadanom rozpätí  
        print('Chybne zadaný vek.')  
else:  
    # chyba je v prezývke  
    print('Chybne zadaná prezývka.')
```

Príslušné funkcie na testovanie prezývky alebo veku vracajú logickú hodnotu podľa toho, či prezývka alebo vek spĺňajú uvedené požiadavky.

Všimnime si testovanie vo funkcii `test_prezývka()`. Postupne testujeme jednotlivé požiadavky. Ihneď ako narazíme na nesplnenú požiadavku, vraciame hodnotu `False` a funkciu ukončíme. Ďalšie testy nemajú zmysel.

V hlavnom programe najskôr načítame a otestujeme prezývku. Ak je prezývka v poriadku, načítame a otestujeme vek.

Vyššie uvedené riešenie nie je jediné správne. Žiaci zrejme navrhnu iné, možno menej efektívne riešenie. Ak je žiacke riešenie správne, odporúčame ho akceptovať.

Niektorí žiaci sa možno pokúsia vytvoriť program, ktorý by bližšie konkretizoval chybu. V tomto prípade by funkcie nemali vracieť logickú hodnotu, ale reťazec popisujúci typ chyby alebo prázdny reťazec. Priestor pre rozpracovanie tohto programu je naozaj široký.

## VYHODNOTENIE (CCA 5 MIN)



*Sebahodnotiaci test*

1.	<p>Pani učiteľka v rámci záverečného opakovania pripravuje na každú hodinu slovenského jazyka krátky diktát vo forme doplňovačky: namiesto i/y napíše znak _ a žiaci wpisujú na tento znak správne písmeno.</p> <p>Vyberte správny kód, ktorý upraví vstupný text diktátu tak, že nahradí všetky výskyty písmen i, í, y, ý, l, ľ, Y, Ý znakom _, napr. pri vstupnom texte „V našej peci myši pištia. Asi nie sú sýte.“ získame výstupný text „V našej pec_m_š_p_št_a. As_n_e sú s_te.“</p> <p>Pomôcka: Viac o metóde <code>replace()</code> sa dozviete v časti „Vedomosti v kocke“.</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid #ccc; padding: 5px; width: 45%;"> <p><b>Kód 1</b></p> <pre>text = 'V koryte spí milá myš.' pismena = 'ííýÝÍÍÝÝ'  for znak in pismena:     text = text.replace(znak, '_')  print(text)</pre> </div> <div style="border: 1px solid #ccc; padding: 5px; width: 45%;"> <p><b>Kód 2</b></p> <pre>text = 'V koryte spí milá myš.' pismena = 'ííýÝÍÍÝÝ'  for znak in pismena:     text.replace(znak, '_')  print(text)</pre> </div> </div> <p>Správny je kód číslo <u>1</u>.</p>
2.	<p>Preštudujte si nasledujúci program:</p> <pre>vstup = input('Zadajte vstupný reťazec: ') if vstup[0] == 'L' and vstup[-3:] == '035':     if len(vstup) &lt; 8:         print('Výpis1')     else:         print('Výpis2') else:     print('Výpis3')</pre> <p>A. Aký bude výstup nasledujúceho programu, ak na vstupe zadáme reťazec 'Letisko035'? <b>Výpis2</b></p> <p>B. Vytvorte taký vstupný reťazec, aby výstupom programu bol <code>Výpis1</code>. <b>Leto035</b></p>

Odporúčame, aby učiteľ uviedol správne odpovede a na záver zhrnul nové poznatky:

- Pri práci s reťazcom alebo so znakom môžeme používať reťazcové metódy. Je dôležité uvedomiť si, že výsledkom metódy je:
  - kópia reťazca upravená podľa určenia metódy (napr. pri metóde `reťazec.swapcase()`),
  - hodnoty `True` alebo `False` (napr. pri metóde `reťazec.isalpha()`),
  - číselná hodnota (napr. pri metóde `reťazec.count(podreťazec)`).

Návratovú hodnotu reťazcovej metódy preto vždy musíme prepojiť s premennou alebo ju použiť ako parameter funkcie pre ďalšie spracovanie.

- Pri testovaní vstupných hodnôt máme k dispozícii zložené a vnorené podmienky. Zložené podmienky formulujeme pomocou logických operátorov, vnorené podmienky vytvárame štruktúrou viacerých príkazov `if` (v rámci ktorých môžeme – ak je to vhodné – použiť aj

zložené podmienky). Ak je to možné, uprednostníme vnorené podmienky, riešenie sa zefektívni.

**Systematizácia**

Cieľom nasledujúcej metodiky je systematizácia a precvičenie vedomostí a zručností žiakov pri práci s reťazcom. Je preto vhodné, ak žiakov v závere tejto vyučovacej hodiny požiadame, aby si pozorne prečítali **Vedomosti v kocke** na predchádzajúcich dvoch pracovných listoch, a preštudovali si aj úlohy, ktoré sme v rámci týchto listov riešili.

## 13 ALGORITMY S REĽAZCAMI

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>pomocou premenných,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>rozdeliť problém na podproblémy a na ich riešenie definovať vhodné funkcie,</li> <li>používať premennú vo výrazoch,</li> <li>používať príkazy cykly a vetvenia.</li> </ul>	
<b>Ciele</b>	
Žiakom osvojované vedomosti a zručnosti	Žiakom rozvíjané spôsobilosti
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>popisovať očakávané výstupy, výsledky, akcie.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať matematické výrazy pri vyjadrovaní vzťahov,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>identifikovať vlastnosti vstupnej informácie (obmedzenia, rozsah, formát).</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých treba výsledok získať akumulovaním čiastkových výsledkov v rámci cyklu.</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>dopĺňať, dokončovať, modifikovať rozpracované riešenie,</li> <li>uvažovať o rôznych riešeniach, navrhovať vylepšenie.</li> </ul> <b>Hľadanie a opravovanie chýb:</b> <ul style="list-style-type: none"> <li>rozpoznávať, kedy program pracuje nesprávne,</li> <li>hľadať chybu vo vlastnom, nesprávne pracujúcom programe a opraviť ju,</li> <li>zistiť, pre aké vstupy, v ktorých prípadoch, situáciách program zle pracuje,</li> <li>uvádzať kontra príklad, kedy niečo neplatí,</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa algoritmov,</li> <li>(LOG4) vyvodzovať (logicky zdôvodňovať) závery z pozorovaní a experimentov,</li> <li>(LOG6) logicky zdôvodniť zmenu algoritmu/programu.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém/časti problému (postupnosti krokov na realizáciu nejakej činnosti vedúcej k cieľu, vytvárať scenáre a storyboardy, postup experimentu).</li> <li>(ALG4) vytvárať vlastné algoritmy, ktoré pracujú s množinou/modifikujú množinu dát (úprava reťazca, resp. podreťazca pomocou reťazcových metód).</li> </ul>

<p>nefunguje,</p> <ul style="list-style-type: none"> <li>• posudzovať a overovať správnosť riešenia (svojho aj cudzieho),</li> <li>• rozlišovať chybu pri realizácii od chyby v zápise.</li> </ul> <p>Používať operácie, operátory, funkcie a metódy programovacieho jazyka Python pre prácu s reťazcami pri riešení problémov.</p> <p>Používať logické operátory, zložené a vnorené podmienky.</p>	
<p><b>Riešený didaktický problém</b></p>	
<p>Žiaci sa na predchádzajúcich vyučovacích hodinách zoznámili s reťazcom, spoznali operácie, operátory, funkcie a metódy, ktoré umožňujú a často uľahčujú prácu s reťazcom. Dôležité je, aby žiaci tieto nové poznatky a skúsenosti vedeli aplikovať, vzájomne prepájať a efektívne používať. Cieľom metodiky je systematizácia a upevnenie získaných vedomostí a zručností žiakov pri práci s reťazcami.</p>	
<p><b>Dominantné vyučovacie metódy a formy</b></p>	<p><b>Príprava učiteľa a pomôcky</b></p>
<ul style="list-style-type: none"> <li>• Frontálna a individuálna forma.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/13/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/13/</b> pracovný priestor pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<p><b>Diagnostika splnenia vzdelávacích cieľov</b></p>	
<p>Spätná väzba získavaná počas riešenia úloh, v závere formou diskusie.</p>	

## Úvod

Žiaci už vedia, že doteraz používané vstupy sú v jazyku Python reprezentované vo forme reťazca. V metodike „Reťazce“ sme si ukázali operácie (napr. `+`), operátory (napr. príslušnosti, logické, relačné) a funkcie (napr. `len()`) pre prácu s reťazcom. V metodike „Reťazcové metódy, zložené a vnorené podmienky“ sa žiaci zoznámili s reťazcovými metódami, ktoré nám často uľahčujú prácu s reťazcom. Do úloh boli včlenené aj zložené podmienky. V tejto metodike so žiakmi precvičíme získané vedomosti a zručnosti na ďalších typoch úloh, v ktorých sa budeme snažiť prepájať jednotlivé vedomosti a vytvárať komplexnejšie úlohy.

Táto metodika nie je založená na bádateľskej metóde, jej cieľom je systematizácia a precvičenie vedomostí a zručností žiakov pri práci s reťazcom. Je preto vhodné, ak na predchádzajúcej hodine žiakov požiadame, aby si pozorne prečítali Vedomosti v kocke na predchádzajúcich dvoch pracovných listoch, aj úlohy, ktoré sme v rámci týchto listov riešili.

Pri riešení niektorých úloh ukážeme aj rôzne prístupy, a teda aj rôzne výsledné programy. Systém práce, ktorý je vhodný vo fáze fixácie, je závislý od skupiny žiakov. Môžeme úlohy riešiť spoločne, pričom po prečítaní úlohy necháme žiakom priestor na rozmyšľanie. Môžeme nechať žiakov pracovať samostatne, ale každú úlohu by sme potom mali prejsť spolu s nimi, aby sme prípadne mohli usmerniť ich pozornosť aj k takým riešeniam, ktoré využívajú iné nástroje jazyka.

V metodike sa úmyselne venujeme podrobnému popisu myšlienkových postupov pri riešení úloh. Pomôže nám to odhaliť prípadné miskoncepce, ktoré si žiaci mohli osvojiť.

### **Dôležité upozornenie k uvádzaným riešeniam úloh**

Pri väčšine riešení uvádzame ako prvé tzv. „drevorubačské riešenia“, ktoré zvyknú žiakom napadnúť ako prvé. Ak sa žiaci týmto smerom nebudú uberať, nebudeme ich spomínať, aby sme žiakov zbytočne nezavádzali nesprávnym smerom. Tieto riešenia uvádzame pre potreby učiteľa práve ako možné zdroje spomínaných žiackych miskoncepíí.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny:

- **Motivácia (5 minút)** – diskusia so žiakmi o možnostiach využitia práce s reťazcami.
- **Expozícia (5 minút)** – systematizácia vedomostí s využitím časti Vedomosti v kocke z pracovných zošitov k metodikám „Reťazce“ a „Reťazcové metódy, zložené a vnorené podmienky“.
- **Fixácia (25 minút)** – aplikácia vedomostí a zručností, riešenie úloh 1 až 4.
- **Diagnostika (5 minút)** – získanie spätnej väzby v diskusii so žiakmi o riešených úlohách.

## MOTIVÁCIA (CCA 5 MIN)

V tejto fáze pracovný list nepoužívame.

V predchádzajúcich dvoch metodikách sme pozornosť žiakov zamerali na reťazce – ukázali sme, aké nástroje ponúka Python na ich spracovanie. Dôležité je nielen tieto nástroje poznať, ale vedieť ich správne použiť. Naše programy by mali byť efektívne (to je výsledkom dôslednej analýzy riešeného problému a výberom vhodných štruktúr, príkazov, ...), priateľské k používateľovi a čitateľné.

V tejto fáze sa v diskusii so žiakmi môžeme pokúsiť zistiť, či si uvedené vlastnosti programov uvedomili, či sú pre nich dôležité, alebo zotrvávajú na pozícii „hlavne, že program funguje“.

## EXPOZÍCIA (CCA 5 MIN)

Na konci každého pracovného listu sú umiestnené Vedomosti v kocke, ktoré si žiaci mali opäť preštudovať v rámci prípravy na dnešnú hodinu. V tejto fáze ponecháme priestor pre otázky žiakov – v prípade konkrétnych, jednoznačných otázok žiakom odpovieme. Ak si otázka žiada rozsiahlejšiu odpoveď, môžeme si ju poznačiť a vrátiť sa k jej zodpovedaniu pri riešení konkrétnych úloh v nasledujúcej časti hodiny.

## FIXÁCIA (CCA 25 MIN)

### Úloha 1

Katka sa chystá na študijný pobyt do USA. Naštudovala si, že okrem inej meny a dĺžkových jednotiek používajú obyvatelia USA aj inú jednotku teploty – stupeň Fahrenheita. Zaskočilo ju to, lebo zistila, že prepočet medzi stupňami Fahrenheita a Celzia je dosť náročný.

Definujte funkciu `prevod_teploty()`, ktorá pre zadaný teplotný údaj (reťazec reprezentujúci číslo a jednotku teploty) v jednej stupnici, vráti zodpovedajúci teplotný údaj v druhej stupnici. Riešenie uložte do súboru **teplota.py**.

Vstup funkcie:	Výstup funkcie:
'45 F'	'7.22 C'
'37 C'	'98.60 F'

Pomôcka – Vzorec pre prevod teploty zo stupňov Celzia na stupne Fahrenheita:  $F = C * 1.8 + 32$ .

Riešenie:

Riešenie úlohy je založené na analýze vstupného reťazca. Keďže môžeme predpokladať korektný vstup v tvare celé číslo – medzera – jednotka teploty (F alebo C), vieme sa dostať k jednotlivým dôležitým údajom pomocou výrezov. Jednotka teploty sa nachádza na poslednej pozícii reťazca (s indexom -1), číselný údaj od začiatku po tretiu pozíciu od konca (tá má index -3).

Keď z reťazca extrahujeme jednotlivé časti zápisu teploty, môžeme previesť teplotu do inej stupnice a vytvoriť výsledný reťazec – vyjadrenie teploty v druhej stupnici.

```
def prevod_teploty(vstup):
    jednotka = vstup[-1]
    hodnota = vstup[:-2]
    hodnota = float(hodnota)
    if jednotka == 'F':
        # prevod z F na C
```

```

vystupna_hodnota = round((hodnota - 32) / 1.8, 2)
return f'{vystupna_hodnota} C'
else:
    # prevod z C na F
    vystupna_hodnota = round(hodnota * 1.8 + 32, 2)
    return f'{vystupna_hodnota} F'

```

Pri riešení úlohy sme sa museli spoľahnúť, že vstupný údaj je zadaný v očakávanej podobe. V nasledujúcich dvoch metodikách sa budeme venovať práve tomu, ako ošetriť aj nekorektné vstupy tak, aby nespôsobili (pre používateľa nepochopiteľné) zastavenie behu programu. V tejto metodike sa touto problematikou nebudeme zaoberať.

## Úloha 2

Peter poprosil svoju sestru Betku, aby prepísala jeho rukou písaný referát na počítači. Keď mu sestra odovzdala prepísaný referát, zistil, že pri písaní mala zamenené klávesy y a z.

Definujte funkciu `vymen_yz()`, ktorá pre zadaný text vráti upravený text, v ktorom budú vzájomne vymenené všetky výskyty znakov 'y' a 'z'. Riešenie uložte do súboru **referat.py**.

Pomôcka: overte si, aké znaky sa do dokumentu vložia, ak sa pokúsite napísať i s mäkčeňom alebo z s dĺžňom.

Riešenie:

Pri „opravovaní“ takto pokazeného textu môžu nastať dva rôzne typy situácií:

- *chcený znak v jazyku existuje, ale chybné napísaný nie,*  
napr. chceli sme napísať 'ý' a napísali sme 'ž',  
potom stačí nahradiť všetky výskyty chybného znaku pôvodne chceným znakom: 'ž' → 'ý',
- *chcený aj chybné napísaný znak sú v slovenskom jazyku korektné,*  
napr. chceli sme napísať 'y' a napísali sme 'z',  
potom musíme znaky vzájomne vymeniť 'z' ↔ 'y',  
musíme si dať pozor pri výmene, ak nahradíme každé 'z' znakom 'y', už nebudeme vedieť, ktoré 'y' je pôvodné a ktoré práve nahradené, pomôžeme si pomocným reťazcom, ktorý by sa v texte nemal vyskytovať.

```

def vymen_texty(text1, text2, text):
    pomocny_text = '?*!'
    text = text.replace(text1, pomocny_text)
    text = text.replace(text2, text1)
    text = text.replace(pomocny_text, text2)
    return text

```

```

def vymen_yz(text):
    text = text.replace('~y', 'ž')
    text = text.replace('~Y', 'Ž')
    text = text.replace('ž', 'ý')
    text = text.replace('Ž', 'Ý')
    text = vymen_texty('z', 'y', text)
    text = vymen_texty('Z', 'Y', text)
    return text

```

Uvedenú úlohu je možné riešiť na viacerých úrovniach.

- Uvažujeme len znaky a výmeny: 'z' ↔ 'y',
- Uvažujeme aj znaky a výmeny: 'Z' ↔ 'Y',
- Uvažujeme aj znaky a nahradenia: 'ž' → 'ý' a 'Ž' → 'Ý',
- Uvažujeme aj znaky a nahradenia: 'ž' → 'ž' a 'Ž' → 'Ž'.

Nechávame na zvážení učiteľa, pre ktorú z úrovní sa rozhodne.

Úlohu je možné riešiť aj iným spôsobom. Postupným prechodom znakmi reťazca a testovaním, či aktuálny znak nie je jedným z tých, ktoré je potrebné nahradiť. Tento prístup však komplikuje skutočnosť, že v niektorých prípadoch je potrebné nahradiť dva znaky. Z tohto dôvodu je uvedené riešenie jednoduchšie.

### Úloha 3

Pani učiteľka slovenského jazyka posudzuje náročnosť diktátu podľa počtu písmen i, l, í, ľ, y, Y, ý a Ý v diktovanom texte. Definujte funkciu `pocet_iy()`, ktorá pre zadaný reťazec vypíše, koľko písmen i, l, í, ľ, y, Y, ý a Ý reťazec obsahuje. Riešenie uložte do súboru **diktat.py**.

Riešenie:

```
def pocet_iy(text):
    pismena_iy = 'iIíÍyYýÝ'
    pocet = 0
    for i in pismena_iy:
        pocet = pocet + text.count(i)
    return pocet
```

Uvedenú úlohu je možné riešiť aj inak:

```
def pocet_iy(text):
    pismena_iy = 'iIíÍyYýÝ'
    pocet = 0
    for znak in text:
        if znak in pismena_iy:
            pocet = pocet + 1
    return pocet
```

Z hľadiska časovej náročnosti sú obidve riešenia rovnako efektívne.

Riešenie, v ktorom použijeme zložený logický výraz:

```
def pocet_iy(text):
    pocet = 0
    for znak in text:
        if znak == 'i' or znak == 'I' or \
           znak == 'í' or znak == 'Í' or \
           znak == 'y' or znak == 'Y' or \
           znak == 'ý' or znak == 'Ý':
            pocet = pocet + 1
    return pocet
```

je zbytočne komplikované a neprehľadné. Ak to podmienky vyučovania dovoľia, odporúčame so žiakmi prediskutovať uvedené riešenia a poukázať na rozdiely v nich.



**Úloha 4**

Vo vstupnom formulári pre evidenciu nových zamestnancov programátori vytvorili textové pole pre zadanie mena a priezviska zamestnanca. Neskoro si uvedomili, že pre potreby databázy zamestnancov potrebujú samostatne meno a samostatne priezvisko.

Definujte funkcie `meno()` a `priezvisko()`, ktorých vstupným parametrom je reťazec v tvare `'menomedzerapriezvisko'`. Funkcia `meno()` nech vráti meno zamestnanca (prvé písmeno veľké, ostatné malé). Funkcia `priezvisko()` nech vráti priezvisko zamestnanca (všetky písmená veľké). Riešenie uložte do súboru **zamestnanec.py**.

Riešenie:

```
def meno(vstup):
    pozicia_medzery = vstup.find(' ')
    vystup = vstup[:pozicia_medzery]
    vystup = vystup[0].upper() + vystup[1:].lower()
    return vystup

def priezvisko(vstup):
    pozicia_medzery = vstup.find(' ')
    vystup = vstup[pozicia_medzery + 1:]
    vystup = vystup.upper()
    return vystup
```

**DIAGNOSTIKA (CCA 5 MIN)**

Vrátime sa k otázkam, ktoré žiaci kládli v expozičnej alebo fixačnej fáze a overíme si, či žiaci porozumeli problematickým častiam učiva.

Vrátime sa k riešeným úlohám a pri každej zdôrazníme, prečo sme použili práve tento algoritmus. Pre jednoduchšiu orientáciu ponúkame zoznam nástrojov, ktoré sme v jednotlivých úlohách použili:

**1. úloha**

- index reťazca,
- výrez,
- podmienky,
- pretypovanie,
- rozdeľovanie a spájanie reťazcov.

**2. úloha**

- reťazcová metóda `replace()`.

**3. úloha**

- cyklus cez znaky reťazca,
- operátor príslušnosti resp. reťazcová metóda `count()`,
- logické operátory a zložené podmienky (ako príklad neefektívneho riešenia).

## 14 ODCHYTÁVANIE VÝNIMIEK

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
<b>Algoritmické riešenie problémov:</b> <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>pomocou premenných,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy,</li> <li>používať premennú vo výrazoch,</li> <li>definovať vlastné funkcie,</li> <li>rozpoznávať situácie a podmienky, kedy treba použiť vetvenie,</li> <li>rozpoznávať situácie a podmienky, kedy treba použiť cyklus so známym počtom opakovaní.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>používať matematické výrazy pri vyjadrovaní vzťahov,</li> <li>rozpoznávať a odstraňovať chyby v zápise,</li> <li>vytvárať zápisy a interpretovať zápisy podľa stanovených pravidiel (syntaxe) pre zápis algoritmov.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti.</li> </ul> <b>Pomocou premenných:</b> <ul style="list-style-type: none"> <li>identifikovať zo zadania úlohy, ktoré údaje musia byť zapamätané, resp. sa menia,</li> <li>riešiť problémy, v ktorých si treba zapamätať a neskôr použiť zapamätané hodnoty vo výrazoch,</li> <li>zovšeobecniť riešenie tak, aby fungovalo nielen s konštantami.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba získať vstup,</li> </ul>	<b>Koncepty informatického myslenia</b>  <b>Logika:</b> <ul style="list-style-type: none"> <li>(LOG1) využitím logických zdôvodnení predpokladať správanie sa algoritmov (správanie sa programu v závislosti od vstupných údajov),</li> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa jednoduchých programov,</li> <li>(LOG4) vyvodzovať (logicky zdôvodňovať) závery z pozorovaní a experimentov (aj myšlienkových),</li> <li>(LOG6) logicky zdôvodniť zmenu algoritmu/programu.</li> </ul> <b>Algoritmy:</b> <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém,</li> <li>(ALG6) dotvárať nekompletné algoritmy (doplň kód, dokonči program),</li> <li>(ALG7) vylepšovať existujúce algoritmy (zlepšenie efektívnosti, rozšírenie algoritmu na väčšiu množinu vstupov, rozšírenie funkcionality).</li> </ul>

<ul style="list-style-type: none"> <li>identifikovať vlastnosti vstupnej informácie (obmedzenia, rozsah, formát),</li> <li>rozpoznávať situácie, kedy treba zobrazíť výstup,</li> <li>zapisovať algoritmus, ktorý reaguje na vstup.</li> </ul> <p>Hľadanie a opravovanie chýb:</p> <ul style="list-style-type: none"> <li>rozlišovať chybu pri realizácii od chyby v zápise.</li> </ul> <p>Vytvoriť korektné Python programy, ktoré odchyťávajú výnimky a poskytujú používateľovi zrozumiteľnú spätnú väzbu o príčine vzniknutého problému.</p>	
<p><b>Riešený didaktický problém</b></p>	
<p>Ošetreniu chybných vstupných hodnôt (t. j. takých, ktoré spôsobia zastavenie behu programu s chybovou správou) sa zvyčajne venujú záverečné hodiny programovania.</p> <p>Pre korektné riešenie daného problému je však veľmi dôležité poznať, uvedomiť si podmienky pre vstupné údaje (formát, rozsah, obmedzenia) – to je skutočným dôkazom pochopenia komplexnosti riešenej úlohy. Preto sa tejto téme venujeme skôr, ako je zvyčajné. Vyžaduje to dôkladnú analýzu problému, jeho dekompozíciu a zvládnutie potrebného zápisu upraveného algoritmu.</p> <p>V programovacom jazyku Python namiesto množstva testov, či je možné nejakú akciu vykonať, táto akcia sa vykoná v akomsi „pokusnom“ prostredí a po jej vykonaní sa zisťuje, či pri tom došlo k nejakej behovej chybe.</p>	
<p><b>Dominantné vyučovacie metódy a formy</b></p>	<p><b>Príprava učiteľa a pomôcky</b></p>
<ul style="list-style-type: none"> <li>Bádateľská metóda (model 5E),</li> <li>frontálna a individuálna forma.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li><b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li><b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li><b>ucitel/pracovne_subory_riesenia/14/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li><b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li><b>ziak/pracovne_subory/14/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<p><b>Diagnostika splnenia vzdelávacích cieľov</b></p>	
<p>Sebahodnotiaci test v pracovnom zošite.</p>	

## Úvod

Na predchádzajúcej vyučovacej hodine žiaci vytvárali programy, ktoré komunikovali s používateľom prostredníctvom konzoly. Pri ladení programov a ich spustení sme spolu so žiakmi určite narazili na chyby. S chybami sa stretávame počas celej výučby programovania. Viac sme s nimi experimentovali na prvej vyučovacej hodine programovania, keď sme objavovali Python ako kalkulačku. Objavili sa určite aj pri riešení úloh v prostredí korytnačej grafiky, aj pri riešení nového typu úloh na minulej vyučovacej hodine. Už vtedy sme nabádali k tomu, aby – ak tak neurobí žiak – urobil chybu učiteľ a ukázal, ako na túto chybu program reaguje.

Chyby sú súčasťou práce programátora. Mýliť sa je ľudské, ale pre programátora to platí len vo fáze ladenia programu. Výsledný program by mal mať „vychytané“ všetky možné problematické situácie.

V rámci tejto metodiky si ukážeme, ako pracovať s tzv. **výnimkami**. Ide o behové chyby, s ktorými sa už žiaci určite stretli. Najprv si ukážeme, ako tieto chyby identifikovať a ako to dať na vedomie používateľovi zrozumiteľnou formou. Takto ošetrené programy budú zrozumiteľnejšie a odolnejšie voči chybám zo strany používateľa. Neskôr na nasledujúcej vyučovacej hodine ukážeme žiakom, ako výnimky generovať (v prípade chýb, ktoré nespôsobujú zastavenie programu s chybovým hlásením, ale kvôli nim dochádza napr. k nesprávnej interpretácii výsledku, zväčša kvôli nekorektnému vstupu).

Vopred uvádzame, že v tejto metodike nesmerujeme riešenie úloh k tvorbe vlastných funkcií. Budeme v nej klásť dôraz na pochopenie významu a spôsobu odchyťavania výnimiek. Na jednoduchých úlohách chceme vysvetliť myšlienku a postup odchyťavania výnimiek a v závere metodiky ladenie (krokovanie) programu. V nasledujúcej metodike už budú mať vlastné funkcie svoje miesto.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – diskusia so žiakmi na tému chýb, s ktorými sa pri programovaní stretli.
- **Skúmanie (10 minút)** – skúmanie s akými behovými chybami sa môžeme stretnúť, ako ich Python prezentuje (úlohy 2 a 3).
- **Vysvetlenie (10 minút)** – vysvetlenie predchádzajúcich zistení, riešenie úlohy (úloha 4).
- **Rozpracovanie (10 minút)** – riešenie náročnejšej úlohy (úloha 5).
- **Vyhodnotenie (5 minút)** – vyriešenie sebahodnotiaceho testu, diskusia o odpovediach, zhrnutie nových poznatkov.

### ZAPOJENIE (CCA 5 MIN)

Na predchádzajúcej hodine žiaci vytvárali vlastné programy, ktoré komunikovali s používateľom prostredníctvom konzoly. Programy spracovávali reťazce, celé aj reálne čísla (podľa zadania úlohy). Pri zadávaní týchto vstupov sa žiaci možno stretli s prerušením behu programu spôsobeným chybne zadaným údajom. Napríklad namiesto čísla sme zadali reťazec, hodnotu použítú v menovateli sme zadali nulovú alebo hodnotu pod odmocninou zápornú.

Na tejto hodine doplníme naše programy o nástroje, ktoré umožnia oznámiť príčinu problému zrozumiteľnou formou, nie štandardným chybovým hlásením. Po oznámení chyby program skončí, zatiaľ totiž nepoznáme mechanizmus, ktorý by nám umožnil opakovane žiadať používateľa o korektný vstup (pôjde o príkaz cyklu `while`). Naším cieľom je presvedčiť žiakov, že je dôležité ošetriť možné behové chyby vopred, a nenechať používateľa v neistote, čo sa vlastne s programom stalo (napr. nerozumie angličtine alebo nechápe, čo a prečo spôsobilo chybu). Využijeme na to diskusiu, v ktorej zosumarizujeme doterajšie skúsenosti žiakov s chybami, ktoré už mali možnosť zažiť pri ladení a spúšťaní programov.

**Úloha 1** Diskutujte o chybách vo vašich doterajších programoch.

- S akými chybami ste sa pri spúšťaní programov doteraz stretli?
- Ako na chyby zareagoval Python?
- Je vôbec potrebné „preložiť“ chybové hlásenia programu do zrozumiteľnej reči?

### SKÚMANIE (CCA 10 MIN)

Žiaci v dvojiciach pracujú s pracovným listom a s programami **kecup.py** a **kecup\_uprava.py**. Cieľom ich skúmania je objaviť čo najviac problematických vstupov (t. j. takých vstupných hodnôt, ktoré spôsobia chybové hlásenie programu alebo nelogický výsledok). Mohla by to byť zaujímavá výzva pre žiakov – „nachytať“ cudzí program.

**Úloha 2** Nasledujúci program počíta počet fliaš potrebných na uskladnenie kečupu.

Otvorte program **kecup.py** a preštudujte si ho.

```
objem_kecupu = input('Objem kečupu (v litroch): ')
objem_kecupu = float(objem_kecupu)
objem_flase = input('Objem fľaše (v litroch): ')
objem_flase = float(objem_flase)
print(f'Počet fliaš: {objem_kecupu // objem_flase}')
```

Spustíte viackrát program pre rôzne prirodzené čísla. Čo je výsledkom operácie `//`?

Pokúste sa zadávať také vstupné hodnoty, ktoré spôsobia zastavenie behu programu alebo nelogický výsledok. Zistené poznatky zapíšte do tabuľky, pokúste sa vysvetliť, čo bolo príčinou správania sa programu.

skúmané hodnoty	moja očakávanie (prečo som zvolil/a práve tieto skúmané hodnoty)	výstup programu (v prípade chybového hlásenia zapíšte len názov chyby)
<code>objem_kecupu =</code> <code>objem_flase =</code>		
<code>objem_kecupu =</code> <code>objem_flase =</code>		
<code>objem_kecupu =</code> <code>objem_flase =</code>		
<code>objem_kecupu =</code> <code>objem_flase =</code>		

Riešenie:

Výsledkom operácie je celočíselný podiel. Pomocou neho zistíme, koľko krát sa hodnota `objem_flase` „zmestí“ do hodnoty `objem_kecupu`.

**Zastavenie programu:** nulový objem fľaše – chyba `ZeroDivisionError`.  
vstup, ktorý nie je číslo a nedá sa pretypovať na float – chyba `ValueError`

**Nelogický výsledok:** aspoň jedna záporná vstupná hodnota.

**Úloha 3** Predchádzajúci program sme vylepšili. Spustíte program **kecup\_uprava.py** a otestujete jeho správne fungovanie pre problematické vstupy, ktoré ste objavili v úlohe 2. Následne porovnajete kódy oboch programov.

```
objem_kecupu = input('Objem kečupu (v litroch): ')
objem_flase = input('Objem fľaše (v litroch): ')
try:
    objem_kecupu = float(objem_kecupu)
    objem_flase = float(objem_flase)
    pocet_fliaš = objem_kecupu // objem_flase
except ValueError:
    print('Na vstupe nebol číselný údaj.')
```

```
except ZeroDivisionError:
    print('Objem fľaše nemôže byť nulový.')
else:
    print(f'Počet fliaš: {pocet_flias}')
```

Odpovedzte na nasledujúce otázky:

- Pri akých vstupoch sa upravený program správa rovnako ako neupravený?  
Odpoveď: Ak zadávame ako vstupy korektné hodnoty alebo záporné čísla.
- Kedy sa upravený program správa inak ako neupravený? Zdôvodnite svoju odpoveď.  
Odpoveď: Správa sa odlišne, keď zadáme reťazec nereprezentujúci číslo alebo keď zadáme nulový objem fľaše.
- Ktorú časť programového kódu umiestňujeme do časti `try`?  
Odpoveď: Do tejto časti umiestňujeme tú časť kódu, ktorá môže spôsobiť zastavenie programu s chybovým hlásením.
- Vysvetlite, v akých situáciách sa vykonajú príkazy v častiach `except`.  
Odpoveď: Vykonajú sa vtedy, ak nastane pri vykonávaní príkazov v časti `try` chyba, ktorú daná časť `except` testuje.
- Vysvetlite, v akej situácii sa vykoná blok príkazov v časti `else`.  
Odpoveď: Príkazy v tejto časti sa vykonajú len vtedy, ak v časti `try` nedošlo k žiadnej chybe uvedenej v častiach `except` (v našom prípade `ValueError` a `ZeroDivisionError`).

## VYSVETLENIE (CCA 10 MIN)

Žiakom necháme priestor na vysvetlenie podstatných rozdielov v správaní sa oboch programov. Smerujeme diskusiu k odpovediam na nasledujúce otázky:

- Uskladnili sme vždy všetok kečup do fliaš? Ako by sme zistili, či nejaký kečup zvýšil?
- V čom sa líšia výstupy oboch programov pri problematických vstupoch?
- Čo spôsobilo zastavenie behu programu a vypísanie chybového hlásenia v úlohe 1?
- Ktorú časť kódu pôvodného programu **kecup.py** sme presunuli do časti `try` v programe **kecup\_uprava.py**?

Žiakov by sme v prípade potreby mali smerovať k týmto objavom, vysvetleniam, záverom pre problematiku chýb:

Najjednoduchšie – z pohľadu programátora – je opraviť syntaktickú chybu. Logická chyba vyžaduje opätovnú analýzu riešenia problému (nesprávny vzorec, nesprávny počet opakovaní cyklu, nesprávne zapísaná podmienka a pod.) Behové chyby spôsobia zastavenie behu programu (tieto sa naučíme odchytiť a spracovať dnes) a následné chybové hlásenie so špecifikáciou chyby, ktorá to spôsobila. V prípade programu **kecup.py** išlo o chyby `ZeroDivisonError` (0 ako hodnota objemu fľaše) a `ValueError` (nečíselný vstup). V programe **kecup\_uprava.py** sú tieto chyby ošetrené. Ak si žiaci všimnú, že ani program **kecup\_uprava.py** nie je dokonalý (dáva výstupy aj pre záporné hodnoty na vstupe), oznámime im, že tento typ chyby (nezmyselné (z pohľadu výpočtu) údaje na vstupe) ošetríme na nasledujúcej vyučovacej hodine. V tejto chvíli by žiaci dokázali vyriešiť tieto špecifické situácie pomocou podmieneného vetvenia programu. Ak si ale uvedomíme, že program môže mať viacero vstupných hodnôt, štruktúra prípadných vnorených príkazov `if` nezní príliš lákavo. S touto situáciou sa stretneme pri všetkých nasledujúcich úlohách.



Učiteľ zhrnie to, čo žiaci v časti skúmanie objavili. Diskusiu využijeme na to, aby sme žiakom predstavili ciele vyučovacej hodiny: naučíme sa upraviť program tak, aby program – ak skončí predčasne – oznámil informáciu, prečo sa tak stalo. Na tzv. odchytyvanie výnimiek používame konštrukciu `try - except`. To, čo je pred touto konštrukciou, sa vykoná v riadnom režime. Za `try` je blok príkazov, ktoré bude Python spúšťať „pokuse“ – vykoná ich a bude sledovať, či by došlo k chybovému hláseniu. Ak áno, v časti `except` nájde príslušný názov chyby a vykoná príkazy na ošetrovanie tejto chyby (v našom prípade oznam o type chyby). Príkazy, ktoré nasledujú za konštrukciou `try - except`, sa vykonávajú vždy (aj keď v časti `try` došlo ku chybe) – aj to je rozdiel v porovnaní s „neošetrenou“ verziou programu **kecup.py** (program skončil v mieste chyby).

Samozrejme, chyby typu `ZeroDivisionError` a `ValueError` nie sú jediné, ktoré môžu nastať. Ak chceme zistiť, aký typ chyby môže pri konkrétnej akcii nastať, využijeme na to konzolu (prácu s ňou žiaci poznajú z prvej hodiny). Ako si to už žiaci mohli vyskúšať, je to vhodné prostredie na experimentovanie a objavovanie.

Možno sa počas diskusie objaví otázka, či je možné zistiť zvyšok po takomto delení (ak sa neobjaví, navodíme ju). Využiť môžeme už známy operátor `%` (zvyšok po celočíselnom delení) a odpoveďou na otázku je zápis `objem_kecupu % objem_flase`. S operáciami celočíselného delenia a zvyšku po delení sa však žiaci mali možnosť oboznámiť už v metodike 9.

#### Poznámka:

Pri tvorbe ďalších programov odporúčame, aby žiaci najprv zapísali program bez odchytenia výnimiek, a až po jeho vytvorení doplnili programový kód tak, aby možné výnimky odchytili.

**Úloha 4** V záhradkárskej osade „Štvorčekovo“ si záujemca môže prenajať len pozemok v tvare štvorca. Stačí, ak oznámi, na akej ploche chce záhradkáriť a geodet mu v osade určí štvorcový pozemok s požadovanou plochou. K tomu však geodet potrebuje poznať rozmery tohto pozemku. Pokúsil sa vytvoriť program **pozemok.py**, ktorý mu pomôže vypočítať rozmery pozemku s požadovanou plochou. Nedokázal ho však dokončiť tak, aby v prípade nekorektného vstupu zrozumiteľne oznámil túto chybu.

Vstupom tohto programu je plocha pozemku (v metroch štvorcových), výstupom rozmery pozemku (v metroch). Doplníte program **pozemok.py** tak, aby pre korektný vstup vypísal rozmery pozemku, v opačnom prípade vypísal správu, že vstupný údaj nebol správny.

Riešenie:

```
import math

plocha = input('Zadajte požadovanú plochu pozemku v metroch štvorcových: ')
try:
    plocha = float(plocha)
    strana = math.sqrt(plocha)
except ValueError:
    print(f'Chybný vstup pre zadaný vstup: {plocha}.')
else:
    print(f'Požadovaný pozemok má rozmery {strana} m x {strana} m.')
```



Pri zápise riešenia do programu sa žiaci po prvýkrát stretnú s modulom `math` a jeho funkciou `sqrt()`. Tu žiakom pomôžeme – už sa stretli s modulom `turtle`, no to nie je jediný modul, ktorý má Python k dispozícii. Matematické funkcie môžeme používať vďaka modulu `math`. Na výpočet druhej odmocniny sme v programe použili funkciu `math.sqrt()`.

Samozrejme, k dispozícii máme širokú ponuku matematických funkcií, ich popis si žiaci môžu pozrieť na stránke <https://docs.python.org/3/library/math.html>.

#### Poznámka:

Ak by sme sa rozhodli nevyužiť modul `math` a pracovať s mocninou s reálnym exponentom  $(S)^{\frac{1}{2}}$  (v programe zapíšeme v tvare `S ** 0.5`), kde `S` reprezentuje zadanú plochu pozemku) a zadali ako plochu pozemku záporné číslo, dostali by sme zaujímavý výsledok – nie chybové hlásenie, ale imaginárne číslo. Preto sme tento spôsob nevyužili.

#### Poznámka

Na tomto príklade môžeme žiakom vysvetliť, prečo nespájame do jedného kroku načítanie údajov a ich konvertovanie. Ak by sme to urobili, premenná `plocha` by neodkazovala na žiadnu hodnotu (pretože príkaz `plocha = float(input('Zadajte požadovanú plochu pozemku v metroch štvorcových: '))` by nebol vykonaný). Program by síce prešiel do bloku `except`, ale vo výpise by nedokázal nájsť v mennom priestore hodnotu prepojenú s premennou `plocha`.

## ROZPRACOVANIE (CCA 10 MIN)

**Úloha 5** Peter je fanúšik adrenalínových zážitkov. Najnovšie ho láka bungee jumping (skok strmhlav z výšky). Našiel ponuky rôznych firiem, líšia sa najmä dĺžkou lana, s ktorým sa skáče (práve od dĺžky lana závisí doba trvania voľného pádu). Peter preto vytvoril program **pad.py**, ktorý na základe hodnoty dĺžky lana vypočíta dobu trvania voľného pádu skokana. Upravte program **pad.py** tak, aby ste odchytili všetky možné výnimky.

Pomôcka: Čas  $t$ , za ktorý dopadne teleso z výšky  $h$  nad zemským povrchom, získame pomocou vzorca  $t = \sqrt{\frac{2 \cdot h}{g}}$ .

Samozrejme, Peter na zem nedopadne, zachytí ho lano – Petrov voľný pád trvá, kým mu to dovoľí dĺžka lana.

Riešenie:

```
import math

g = 9.81 # približná hodnota gravitačného zrýchlenia
dlzka = input('Dĺžka lana: ')
try:
    dlzka = float(dlzka)
    cas_padu = math.sqrt(2 * dlzka / g)
except ValueError:
    print(f'Nečíselná hodnota alebo záporné číslo pre dĺžku: {dlzka}.')
else:
    print(f'Voľný pád bude trvať {cas_padu} s.')
```

Vzorec pre výpočet dráhy voľného pádu žiaci poznajú z fyziky:  $s = \frac{1}{2} \cdot g \cdot t^2$ . Vieme z neho vyjadriť vzťah pre čas  $t$ :  $t = \sqrt{\frac{2 \cdot s}{g}}$  (uvádzame ho ako pomôcku v pracovnom liste pre prípad, že žiaci tento vzorec nepoznajú). Vstupnú hodnotu tohto programu je teda potrebné ošetriť v dvoch prípadoch: vstup nie je reťazec reprezentujúci celé číslo alebo reprezentuje záporné celé číslo.

Chybou, ktorú vieme v tejto úlohe ošetriť pomocou konštrukcie `try - except`, je nekorektný vstup – reťazec nereprezentujúci číslo a zadanie zápornej hodnoty pre dĺžku lana (reálna odmocnina zo záporného čísla nie je definovaná). Čiže ide o chyby, ktoré vzniknú vo funkciách jazyka Python a my ich odchytíme. Zatiaľ nedokážeme rozlíšiť, ktorá chyba vlastne nastala – preto môžeme použiť len všeobecný výpis. Riešením by bolo každú zo situácií umiestniť do samostatného bloku `try - except`.

## VYHODNOTENIE (CCA 5 MIN)

V záverečnej časti požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Na prediskutovanie problematických odpovedí budeme mať priestor na nasledujúcej vyučovacej hodine (v jednej z úloh ošetríme tento program pomocou generovania výnimiek).

### Sebahodnotiaci test

1.	<p>Program na výpočet obvodu trojuholníka sme ošetrili konštrukciou <code>try - except</code>.</p> <pre> 1  def obvod(a, b, c): 2      return a + b + c 3 4 5  strana_a = input('a = ') 6  strana_b = input('b = ') 7  strana_c = input('c = ') 8  try: 9      print('Program počíta obvod trojuholníka.') 10     print('Zadaj postupne dĺžky strán trojuholníka (a, b, c)') 11     strana_a = float(strana_a) 12     strana_b = float(strana_b) 13     strana_c = float(strana_c) 14     print(f'Obvod trojuholníka je {obvod(strana_a, strana_b, strana_c)}') 15 except ValueError: 16     print('Na vstupe sa vyskytli nenumerné hodnoty.') 17 except ZeroDivisionError: 18     print('Počas výpočtu došlo k deleniu nulou.') 19 else:</pre> <p>Ktoré riadky môžeme presunúť pred konštrukciu <code>try - except</code>? <b>9, 10</b></p> <p>Ktoré riadky presunieme do časti <code>else</code> konštrukcie <code>try - except</code>? <b>14</b></p> <p>Sú všetky ošetrenia chýb (v časti <code>except</code>) potrebné? Ak nie, ktoré riadky by sme mohli vypustiť? <b>17, 18, nikde v programe nedelíme</b></p>
----	--

Odporúčame, aby učiteľ uviedol správne odpovede a na záver zhrnul nové poznatky:

- počas behu programu môže dôjsť k rôznym chybám (syntaktické, logické a behové chyby),
- behové chyby vieme odchytiť a informovať o nich používateľa zrozumiteľnou formou pomocou konštrukcie `try - except` (delenie nulou, odmocnina zo záporného čísla, nesprávny typ vstupnej hodnoty).

## 15 GENEROVANIE VÝNIMIEK

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>pomocou premenných,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>rozdeliť problém na podproblémy a na ich riešenie definovať vhodné funkcie,</li> <li>používať premennú vo výrazoch,</li> <li>používať príkazy cykly a vetvenia,</li> <li>vysvetliť, čo je to výnimka a ako ju odchytiť.</li> </ul>	
<b>Ciele</b>	
Žiakom osvojované vedomosti a zručnosti	Žiakom rozvíjané spôsobilosti
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>popisovať očakávané výstupy, výsledky, akcie.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať matematické výrazy pri vyjadrovaní vzťahov,</li> <li>rozpoznávať a odstraňovať chyby v zápise.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba získať vstup,</li> <li>identifikovať vlastnosti vstupnej informácie (obmedzenia, rozsah, formát),</li> <li>rozpoznávať situácie, kedy treba zobrazíť výstup,</li> <li>zapisovať algoritmus, ktorý reaguje na vstup.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie a podmienky, kedy treba použiť vetvenie,</li> <li>riešiť problémy, ktoré vyžadujú vetvenie so zloženými podmienkami (s logickými spojkami).</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>upraviť riešenie úlohy vzhľadom na rôzne obmedzenia.</li> </ul> <b>Hľadanie a opravovanie chýb:</b> <ul style="list-style-type: none"> <li>rozlišovať chybu pri realizácii od chyby v zápise,</li> <li>rozpoznávať, kedy program pracuje</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa jednoduchých programov,</li> <li>(LOG3) využitím logických zdôvodnení detegovať a opravovať chyby v programoch a algoritmoch,</li> <li>(LOG4) vyvodzovať (logicky zdôvodňovať) závery z pozorovaní a experimentov (aj myšlienkových).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG6) dotvárať nekompletné algoritmy (doplň kód, dokonči program).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) rozdeliť veci do viacerých úrovní (aj podproblémy rozdeliť na menšie podpodproblémy atď.)</li> </ul>

<p>nesprávne,</p> <ul style="list-style-type: none"> <li>• zisťovať, pre aké vstupy program zle pracuje,</li> <li>• uvádzať kontra príklad, kedy niečo nefunguje,</li> <li>• posudzovať a overovať správnosť riešenia.</li> </ul> <p>Vytvoriť korektné Python programy, ktoré generujú výnimky a poskytujú používateľovi zrozumiteľnú spätnú väzbu o príčine vzniknutého problému.</p>	
<p><b>Riešený didaktický problém</b></p>	
<p>Väčšina úloh na vyučovacích hodinách programovania na strednej škole je formulovaná tak, že vopred špecifikuje prípustné vstupné hodnoty. Žiak tak hľadá postup výpočtu, spracovania korektných vstupných údajov, a predpokladá sa, že nekorektná situácia nenastane. Ak však žiak dokáže analyzovať vstupné údaje, ich vzťah k výpočtu a výstupu, znamená to, že chápe podstatu riešeného problému. Preto sa téme generovania výnimiek venujeme už v prvej polovici našich metodík.</p>	
<p><b>Dominantné vyučovacie metódy a formy</b></p>	<p><b>Príprava učiteľa a pomôcky</b></p>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• frontálna a individuálna forma.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/15/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/15/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<p><b>Diagnostika splnenia vzdelávacích cieľov</b></p>	
<p>Sebahodnotiaci test v pracovnom zošite.</p>	

## Úvod

Na predchádzajúcej vyučovacej hodine žiaci vytvárali programy, ktoré v prípade istých vstupných hodnôt vedeli odchytiť niektoré typy chýb. Program potom zrozumiteľným spôsobom oznámil používateľovi, že nastala chyba typu napr.:

- **ValueError**
  - ak na vstupe nebol reťazec reprezentujúci číslo,
  - ak sme očakávali číslo celé a používateľ zadal číslo reálne,
  - počas výpočtu došlo k výpočtu druhej odmocniny záporného čísla.
- **ZeroDivisionError**
  - ak počas výpočtu došlo k deleniu číslom nula.

Počas riešenia predchádzajúcich úloh si žiaci určite všimli, že to nie sú všetky typy chýb, s ktorými sa môže programátor stretnúť. Pri istých nelogických vstupoch program vykoná určené kroky a vypíše nezmyselný výsledok (napr. ak počítame obvod kružnice so zápornou hodnotou polomeru; používateľ zadá strany trojuholníka, ktoré nespĺňajú trojuholníkovú nerovnosť a pod.) Tieto chyby, špecifické pre konkrétny problém, musí programátor identifikovať a definovať zodpovedajúcu reakciu programu. Túto činnosť nazývame generovanie výnimiek a predpokladá sa, že programátor vopred podrobne analyzoval problém, ktorý rieši.

Aby sme dokázali generovať výnimky aj pre programy so špecifickými požiadavkami na vstupné údaje, oceníme znalosť práce so zloženými podmienkami a ich spracovanie pomocou príkazu vetvenia `if`. Problematické situácie musíme najskôr na základe testov detegovať.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou hodiny je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – diskusia so žiakmi na tému chýb, s ktorými sa pri programovaní stretli.
- **Skúmanie (10 minút)** – skúmanie s akými behovými chybami sa môžeme stretnúť, ako ich Python prezentuje (úlohy 2 a 3).
- **Vysvetlenie (10 minút)** – vysvetlenie predchádzajúcich zistení, riešenie úlohy 4.
- **Rozpracovanie (10 minút)** – riešenie úlohy 5.
- **Vyhodnotenie (5 minút)** – vyriešenie sebahodnotiaceho testu, diskusia o odpovediach, zhrnutie nových poznatkov.

## ZAPOJENIE (CCA 5 MIN)

V tejto fáze pracovný list nepoužívame. Na predchádzajúcej hodine žiaci vytvárali vlastné programy, ktoré pomocou konštrukcie `try - except` ošetrili, aby niektoré behové chyby (výnimky) odchytili a zrozumiteľnou formou oznámili používateľovi príčinu problematického stavu. Na tejto hodine upravíme naše programy tak, že dokážu zareagovať na tie situácie, ktoré my, programátori, zdefinujeme ako problémové (napr. vstup mimo požadovaného rozsahu).

Programátor sa nemôže spoliehať na to, že používateľ bude akceptovať prípustné vstupné hodnoty alebo že sa nepomýli. Práve preto by mal byť každý program, ktorý vytvoríme, odolný voči úmyselným aj neúmyselným chybám zo strany používateľa. Dôležité je, aby program zareagoval v takýchto situáciách zrozumiteľným oznamom o tom, čo sa stalo.

K tomuto záveru žiakov navedieme pomocou diskusie, v ktorej zosumarizujeme doterajšie skúsenosti žiakov s chybami, ktoré už mali možnosť zažiť pri ladení a spúšťaní programov. Návrh otázok do diskusie je uvedený v zadaní nasledujúcej úlohy.

### Úloha 1 *Diskutujte o chybách v programoch.*

- Odhalí Python všetky nekorektné vstupy? Teda, ak program skončil bez chybového hlásenia, môžeme si byť istí, že sme dostali správny výsledok?
- Skúsili ste niekedy „nachytať“ nejakú aplikáciu/program (online kalkulačky na internete, aplikáciu v mobile) tak, že ste experimentovali so vstupnými hodnotami?
- Aké typy chýb sme nedokázali odchytiť pomocou konštrukcie `try - except`?
- Vo firmách, ktoré vyrábajú počítačové aplikácie, existuje špeciálna pracovná pozícia „tester“. Jeho úlohou je okrem iného objavovanie a testovanie chýb. Stačí, ak tento pracovník len sedí pri počítači a zadáva náhodné dáta?

## SKÚMANIE (CCA 10 MIN)

Žiaci pracujú s pracovným listom a s programami **stvorec1.py** a **stvorec2.py**. Úlohou žiakov je spustiť jednotlivé programy, sledovať ich činnosť/výpis do konzoly pre rôzne vstupy a na základe tohto

skúmania programy porovnať a všimnúť si rozdiely. Cieľom je objaviť spôsob, ako generovať ďalšie typy chýb (výnimky), napr. chybný rozsah vstupu.

V programe **stvorec1.py** spoznáme odchytyvanie výnimiek, s ktorým sme sa zoznámili na predchádzajúcej vyučovacej hodine. Ošetrili sme nekorektný vstup vzhľadom na typ hodnoty (ak zadáme reťazec, ktorý nereprezentuje žiadne číslo, t. j. chyba `ValueError` nastala pri konvertovaní reťazca na reálne číslo funkciou `float()`). Úlohou žiakov je objaviť ďalšie problémové situácie, ktoré síce nespôsobia zastavenie behu programu, dajú však nezmyselný výsledok.

Žiaci pracujú vo dvojiciach.

### Úloha 2 Otvorte program **stvorec1.py**.

```
def obsah_stvorca(a):
    return a ** 2

strana = input('Dĺžka strany štvorca: ')

try:
    strana = float(strana)
except ValueError:
    print('Nečíselná hodnota pre dĺžku strany štvorca.')
else:
    print(f'Obsah štvorca je {obsah_stvorca(strana)}')
```

Program spustíte viackrát. Nájdite také vstupné hodnoty, ktoré spôsobia nezmyselný výsledok.

skúmané vstupné hodnoty	dôvod ich voľby	skutočný výstup programu	aký by mal byť správny výstup programu
text	nie je to číslo	Nečíselná hodnota ...	výstup je správny
-10	strana nemôže byť < 0	Obsah .. 100	oznámenie o chybe
0	strana nemôže byť = 0	Obsah .. 0	oznámenie o chybe
prázdny vstup	strana nemôže byť "	Nečíselná hodnota ...	výstup je správny
2,5	nie celé číslo	Obsah .. 6,25	výstup je správny

Dokážete popísať v akých situáciách program skončil bez chybového hlásenia, ale dal nezmyselný výsledok?

Pre zápornú a nulovú hodnotu dĺžky strany.

### Úloha 3 Otvorte program **stvorec2.py**. Skôr, než program spustíte, zapíšte do stĺpca „predpokladaný výstup programu“ tabuľky výstup, ktorý predpokladáte pre uvedené vstupné hodnoty.

```
def obsah_stvorca(a):
    try:
        a = float(a)
    except ValueError:
        raise ValueError('Nečíselná hodnota pre dĺžku strany štvorca.')
    if a <= 0:
        raise ValueError('Záporná alebo nulová dĺžka strany štvorca.')
    return a ** 2

strana = input('Dĺžka strany štvorca: ')
```



```
try:
    print(f'Obsah štvorca je {obsah_stvorca(strana)}..')
except ValueError as chyba:
    print(chyba)
```

Spustte tento program viackrát, otestujte hodnoty z tabuľky, do tabuľky doplňte skutočné výstupné hodnoty.

vstupné hodnoty	predpokladaný výstup programu	skutočný výstup programu	Aký je dôvod rozdielného výstupu v porovnaní s predchádzajúcim programom?
3		Obsah .. 9	výstup sa nezmenil
-2		Záporná/nulová ..	test na nekladnú hodnotu a generovanie výnimky raise
xyz		Nečíselná hodnota ...	výstup sa nezmenil
3,1		Obsah .. 9,6100	výstup sa nezmenil

## VYSVETLENIE (CCA 10 MIN)

Na základe predchádzajúceho skúmania by žiaci mali formulovať nové poznatky, ktoré počas neho získali. Aj keď je táto téma náročná, počas diskusie by učiteľ nemal prejsť do formy výkladu. Je dobré, ak žiaci formulujú svoje koncepcie, učiteľ ich usmerňuje, koriguje ich odpovede novými otázkami alebo protiargumentmi, a podnecuje diskusiu napr. pomocou týchto otázok:

- Čím sa od seba líšia predchádzajúce dva programy z pohľadu spracovania vstupných údajov a výstupov (žiaci majú pred sebou obidve vyplnené tabuľky)? Ako to bolo dosiahnuté?
- Čo je výsledkom použitia príkazu `raise`?
- Nebolo by jednoduchšie v prípade nevhodne zadanej strany upraviť funkciu tak, aby túto informáciu priamo vypísala do konzoly?

V diskusii by mali žiaci smerovať (alebo ich učiteľ smeruje) k týmto poznatkom a záverom:

Na predchádzajúcej vyučovacej hodine sme sa naučili odchytať výnimky pomocou konštrukcie `try - except`. Odchyťovali sme výnimky, ktoré vznikli pri vykonaní niektorej z funkcií jazyka Python (delenie, druhá odmocnina, konvertovanie reťazca na celé alebo reálne číslo a pod.) Dnes sa naučíme generovať výnimky, ktoré boli spôsobené z pohľadu zadania úlohy nekorektnými hodnotami (čiže samotné funkcie jazyka Python by výpočet s týmito hodnotami vykonali bez chybového hlásenia – napr. obsah štvorca, i keď je vstupná hodnota nekladná, my však tieto hodnoty považujeme za nezmyselné).

V programe **stvorec1.py** sme v konštrukcii `try - except` použili už známy blok `else` – návratovú hodnotu funkcie pre výpočet obsahu sa pokúsime vypísať len vtedy, ak je vstupná hodnota číselná (a teda pri funkcii `float()` nenastala chyba).

V programe **stvorec2.py** sme umiestnili funkciu `obsah_stvorca` do konštrukcie `try - except`. V bloku `try` voláme funkciu – ak nedôjde k žiadnej výnimke, do konzoly sa vypíše výsledok výpočtu. Inak funkcia pomocou príkazu `raise` vygeneruje nami definovanú výnimku (pošle hlavnému programu informáciu – chybové hlásenie).

Funkcia generuje dve výnimky – nečíselná hodnota na vstupe a nekladné číslo na vstupe. Ak niektorá z týchto výnimiek nastane, vykoná sa príkaz `raise`.

Prvú výnimku generujeme použitím už známej konštrukcie `try - except`. Pomocou príkazu vetvenia `if` testujeme podmienku pre korektný výpočet (nezáporná hodnota na vstupe). Ak je vstupná hodnota záporné číslo, generujeme výnimku. V oboch situáciách generujeme výnimku pomocou príkazu `raise ValueError(reťazec)`.

Hlavná časť programu načíta vstupné hodnoty, pokúsi sa zavolať funkciu `obsah_stvorca()` – ak vo funkcii došlo k výnimke, prepojí jej popis s premennou `chyba` a následne ju vypíše do konzoly.

Žiaci možno navrhnú, aby funkcia `obsah_stvorca()` namiesto generovania výnimky príkazom `raise` vypísala do konzoly oznam o nekorektnom vstupe. Bolo by to jednoduchšie z pohľadu doterajších vedomostí. Ak by sme však použili toto riešenie, hlavný program by sa „nedozvedel“ o tom, že došlo ku chybe (a k akej) a pokračoval by v behu ďalej, spoliehajúc sa na návratovú hodnotu funkcie.

Učiteľ môže využiť diskusiu na to, aby predstavil ciele vyučovacej hodiny žiakom (prípadne ich v závere diskusie žiakom stručne prezentuje): naučíme sa „generovať výnimky“, tzn. upraviť program tak, aby jeho výsledkom boli správne odpovede založené na správnych vstupoch, a to pomocou príkazu `raise`.

**Poznámka:**

Využime túto chvíľu a zopakujeme so žiakmi zápis porovnania dvoch hodnôt (<, <=, >, >=, ==, !=).

Vidíme, že po presunutí ošetrenia výnimiek (či už odchytávaním alebo generovaním) do funkcie sa programový kód stáva prehľadným, hlavný program je stručnejší. Vedíme žiakov k tomu, aby ošetrenie výnimiek riešili v kóde funkcie – oceníme to pri tvorbe rozsiahlejších programov (takúto „autonómnú“ funkciu môžeme my alebo iný programátor preniesť v prípade potreby z jedného programu do druhého a používať ju, môžeme ju umiestniť do vlastného modulu).

Funkcie, ktoré budeme vytvárať, budú vracaať buď hodnotu alebo generovať chybové hlásenie – nebudú nič písať do konzoly (výpis, resp. spracovanie návratovej hodnoty bude realizovať hlavná časť programu). Podľa reakcie funkcie vieme, ako má ďalší výpočet pokračovať. Ak by funkcia svoju reakciu len vypísala, k tomuto výpisu sa v hlavnom programe nedostaneme a nevieme teda zabezpečiť relevantnú reakciu.

**Úloha 4** Kamil naprogramoval online hru. Časť programu, ktorá umožní hráčom registrovať sa, však ešte nie je dokončená. K registrácii je potrebná prezývka, pod ktorou bude hráč v hre vystupovať. Kamil však chce, aby si hráči volili prezývky, ktoré spĺňajú nasledovné pravidlá:

- prvý znak musí byť abecedný,
- prezývka musí mať minimálne 5 a maximálne 15 znakov,
- prezývka môže obsahovať len alfanumerické znaky.

Ak prezývka spĺňa uvedené podmienky, prezývka sa upraví tak, aby prvý znak bol veľký a ostatné malé.

Definujte funkciu `spracuj_prezyvku()`, ktorá zadanú prezývku upraví do požadovaného tvaru. V prípade, ak prezývka nespĺňa požiadavky, funkcia nech vygeneruje zmysluplnú výnimku. Riešenie uložte do súboru **registracia.py**.

Otestujte správnosť funkcie v hlavnom programe.

Riešenie:

```
def spracuj_registraciu(prezyvka):
    if not 5 <= len(prezyvka) <= 15:
        raise ValueError('Prezývka musí mať dĺžku 5 .. 15 znakov.')
    if not prezyvka[0].isalpha():
        raise ValueError('Prvý znak prezývky musí byť abecedný.')
    if not prezyvka.isalnum():
        raise ValueError('Prezývka musí obsahovať len znaky abecedy alebo čísllice')
    upravena_prezyvka = prezyvka.capitalize()
    return upravena_prezyvka

prezyvka = input('Zadaj prezývku: ')
try:
    prezyvka = spracuj_registraciu(prezyvka)
except ValueError as chyba:
    print(chyba)
else:
    print(f'Registrácia prebehla úspešne. Upravená prezývka: {prezyvka}')
```

V tejto úlohe nepracujeme s číslami, ale s reťazcami. Splnenie prvého testu zaručí nenulovú dĺžku prezývky. Zvyšný kód by sa tak vykonal bez chyby. Keďže na prezývku sú kladené isté požiadavky, otestujeme ich platnosť. Ak nie sú splnené, generujeme vlastné výnimky aj s popisom chyby (ktorej požiadavke prezývka nevyhovela).

Pri kontrole hodnôt odporúčame najskôr otestovať (vylúčiť) problematické prípady a generovať zodpovedajúce výnimky. Ak zbehnú všetky testy vieme, že hodnota je platná a môžeme ju spracovať. Keďže vyhodnenie výnimky spôsobí ukončenie vykonávania tela funkcie (samozrejme, ak ju ešte vo funkcii neodchytíme), nemusíme testy vnárať do seba. Môžeme ich prehľadne zapísať za sebou. Odporúčame podobne usmerniť aj žiakov.

## ROZPRACOVANIE (CCA 10 MIN)

To, či žiaci porozumeli téme, zistíme pri riešení nasledujúcej úlohy. Keďže pracuje s číselnými hodnotami, využijeme súčasne odchyťávanie aj generovanie výnimiek.

**Úloha 5** Škola cestovateľov často organizuje pre svojich žiakov výlety do blízkeho aj vzdialenejšieho okolia. Podľa počtu záujemcov o výlet objednávajú potrebný počet autobusov vo firme „Cestujte s nami“.

Táto firma má k dispozícii postačujúci počet autobusov. Škola firme nahlási počet účastníkov výletu, firma pripraví potrebný počet autobusov v závislosti od počtu účastníkov.

Na tento výpočet majú vo firme špeciálny program **autobus.py**, v ktorom však nie sú ošetrené nekorektné vstupy.

Upravte program **autobus.py** tak, aby ste ošetrili všetky chybné vstupy a vygenerovali k nim zodpovedajúce výnimky. Otestujte správnosť riešenia v hlavnom programe

```
import math

def pocet_autobusov(pocet_ucastnikov, kapacita_autobusu):
    autobusy = math.ceil(pocet_ucastnikov / kapacita_autobusu)
    return autobusy

# načítanie vstupných hodnôt: počet účastníkov a kapacita vybraného autobusu
pocet_ucastnikov = input('Počet účastníkov výletu: ')
pocet_ucastnikov = float(pocet_ucastnikov)
kapacita_autobusu = input('Kapacita autobusu: ')
kapacita_autobusu = float(kapacita_autobusu)

print(f'Počet autobusov: {pocet_autobusov(pocet_ucastnikov, kapacita_autobusu)}')
```

Riešenie:

```
import math

def pocet_autobusov(pocet_ucastnikov, kapacita_autobusu):
    try:
        # pretypovanie hodnôt na číselné (reálne čísla)
        pocet_ucastnikov = int(pocet_ucastnikov)
        kapacita_autobusu = int(kapacita_autobusu)
    except ValueError:
        # ak pri pretypovaní došlo ku chybe:
        raise ValueError('Počet účastníkov alebo kapacita autobusu nie je celé číslo.')
    if pocet_ucastnikov <= 0:
        raise ValueError('Počet účastníkov je záporný alebo 0.')
    if kapacita_autobusu <= 0:
        raise ValueError('Kapacita autobusu je záporná alebo 0.')
    autobusy = math.ceil(pocet_ucastnikov / kapacita_autobusu)
    return autobusy

# načítanie vstupných hodnôt: počet účastníkov a kapacita vybraného autobusu
pocet_ucastnikov = input('Počet účastníkov výletu: ')
kapacita_autobusu = input('Kapacita autobusu: ')

try:
    # vypísanie počtu autobusov
    print(f'Počet autobusov: {pocet_autobusov(pocet_ucastnikov, kapacita_autobusu)}')
except ValueError as chyba:
    print(chyba)
```

Pôvodný návrh riešenia mal niekoľko nedostatkov:

- vstupné hodnoty sme pretypovali na `float`, takže program by akceptoval aj nie celočíselné hodnoty,
- ak by vstupom neboli čísla, program by generoval pre používateľa nezrozumiteľnú výnimku,
- pretypovanie sme robili v hlavnom programe, preniesť funkciu do iného programu by mohlo byť problematické,
- funkcia akceptovala ľubovoľné číselné hodnoty,
- v prípade nulovej kapacity autobusu by funkcia generovala pre používateľa nezrozumiteľnú výnimku delenie nulou.

Výsledný návrh funkcie sa nespolieha na nejaké kontroly mimo nej. Pre prípustné hodnoty vracia správnu hodnotu. V opačnom prípade generuje zmysluplné výnimky.

**Poznámka:**

Pre výpočet počtu autobusov sme použili pre žiakov zatiaľ neznámu funkciu `math.ceil()`, ktorá zadané číslo zaokrúhli smerom hore.

Rovnaký výsledok dosiahneme použitím operátora `//` (celočíselné delenie) a otestovaním zvyšku po delení (operátor `%`).

Ponechávame na zvážení učiteľa, ktorý zo spôsobov uprednostní.

## VYHODNOTENIE (CCA 5 MIN)

### Sebahodnotiaci test

1. V predajni má každý druh tovaru pridelený kód. Kód je tvorený aspoň štvormiestnym prirodzeným číslom. Majiteľ predajne sa rozhodol všetky kódy skrátiť tak, že odstráni poslednú cifru kódu. Pre tento účel vytvoril program, ktorý neskôr doplní o zápis nového kódu do súboru. Súbor následne vytlačí na špeciálny etiketovací papier a kódy umiestni na svoj tovar.

```

1  def uprav_kod(vstup):
2      try:
3          vstup = int(vstup)
4      except ValueError:
5          raise ValueError('Chyba1')
6      if vstup < 1000:
7          raise ValueError('Chyba2')
8      return vstup // 10
9
10
11  kod = input('Zadaj kód tovaru: ')
12
13  try:
14      novy_kod = uprav_kod(kod)
15  except ValueError as chyba:
16      print(chyba)
17  else:
18      print(f'{kod} => {novy_kod}')
```

Určte výstup tohto programu pre jednotlivé vstupné hodnoty:

a. 3

Chyba2

d. 68974

6897

	b. Xy.k	<u>Chyba1</u>	e. 0	<u>Chyba2</u>
	c. -4567	<u>Chyba2</u>	f. 3.456	<u>Chyba1</u>

Odporúčame, aby učiteľ uviedol správne odpovede a na záver zhrnul nové poznatky:

- to, že sa program vykoná bez chybového hlásenia, neznamená, že je program správny,
- počas behu programu môže dôjsť k výpočtom, ktoré síce nespôsobia zastavenie programu s chybovou správou, sú však vzhľadom na zadanie úlohy nekorektné/nezmyselné,
- tieto chyby musíme identifikovať my, programátori, a určiť, ako má na ne program reagovať (pomocou príkazu `raise`) – žiada si to podrobnú analýzu vstupných údajov a výpočtov na týchto údajoch,
- správnou voľbou poradia testovania podmienok vieme zabezpečiť, aby vstupy postupne prechádzali špecifickejšími „sitami“, ktoré dovoľia vykonať výpočet len s korektnými hodnotami.

## 16 OPAKOVANIE III. + DIDAKTICKÝ TEST

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>pomocou premenných,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 2 vyučovacie hodiny
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy, popisovať očakávané výstupy, identifikovať vlastnosti vstupnej informácie,</li> <li>rozdeliť problém na podproblémy a na ich riešenie definovať vhodné funkcie,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov,</li> <li>používať príkazy cykly a vetvenia,</li> <li>riešiť problémy, v ktorých treba výsledok získať akumulovaním čiastkových výsledkov v rámci cyklu,</li> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia,</li> <li>upraviť riešenie úlohy vzhľadom na rôzne dané obmedzenia,</li> <li>hľadať vzťah medzi vstupom, algoritmom a výsledkom,</li> <li>zisťovať, pre aké vstupy, v ktorých prípadoch, situáciách program zle pracuje.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
Programovací jazyk Python: <ul style="list-style-type: none"> <li>systematizovať a precvičiť príkazy, štruktúry a postupy jazyka Python pri riešení problémov (z metodík 11 až 15).</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa algoritmov,</li> <li>(LOG4) vyvodzovať (logicky zdôvodňovať) závery z pozorovaní a experimentov,</li> <li>(LOG6) logicky zdôvodniť zmenu algoritmu/programu.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG7) vylepšovať/dotvárať existujúce algoritmy.</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) rozdeliť veci do viacerých úrovní (aj podproblémy rozdeliť na menšie podproblémy atď.)</li> </ul>
<b>Riešený didaktický problém</b>	
V metodikách 11 až 13 žiaci riešili úlohy zamerané na prácu s reťazcom. V ďalších dvoch metodikách sme sa venovali odchyťavaniu a generovaniu výnimiek. Úspešné zvládnutie jednotlivých nových poznatkov môžeme overiť až pri riešení komplexnejších úloh. Žiaci musia dekomponovať problém na podproblémy, analyzovať aké hodnoty vstupu a výstupu potrebujeme, zvoliť vhodný formát vstupných a výstupných hodnôt, spracovávať textové vstupy pomocou pretypovania, odchyťavania a generovania výnimiek, výberom vhodných operátorov, funkcií a metód pre prácu s reťazcom.	

<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• Problémové vyučovanie,</li> <li>• frontálna a individuálna forma.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/16/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu,</li> <li>• <b>ucitel/testy/test3/</b> didaktický test, javová analýza testu, tabuľka pre vyhodnotenie testu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/16/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu a didaktický test.	



## Úvod

Cieľom metodiky je precvičenie a upevnenie poznatkov získaných v rámci metodík 11 až 15. Úlohy sú zamerané na spracovanie reťazcov, odchyťovanie a generovanie výnimiek.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny:

- **Úvod (5 minút)** – vytvorenie pojmovej mapy na podporu zopakovania a systematizácie poznatkov získaných v rámci metodík 11 až 15.
- **Precvičovanie, samostatná práca (30 minút)** – riešenie úloh z pracovného listu.
- **Zhrnutie (5 minúty)** – diskusia.

### ÚVOD (CCA 5 MIN)

Pomocou pojmovej mapy (<https://www.mindmeister.com/1091269253?t=HszsrwmrHL>) spolu so žiakmi prejdeme jednotlivé nové poznatky. Vytvárať kompletnú pojmovú mapu k témam, ktoré sú predmetom našej systematizácie, by bolo časovo náročné. Preto sme pripravili pojmovú mapu, do ktorej žiaci vpisujú informácie podľa osobnej potreby (slovné vysvetlenie, príklad použitia atď.) Pojmová mapa je súčasťou pracovných listov.

### PRECVIČOVANIE (CCA 30 MIN)

#### Aké riešenia od žiakov očakávať?

Uvedené autorské riešenia nie sú jediné správne. Riešenia sme vytvorili tak, aby sme efektívne využili poznatky, ktoré by žiaci už mali mať. Odporúčame, aby učiteľ smeroval žiakov k riešeniam, ktoré efektívne využívajú prvky jazyka Python. Pridržiavať sa len požiadavky typu „Nech program vracia správne výsledky“ nie je tá najlepšia stratégia. Ponechávame na učiteľovi, ktorý najlepšie pozná svojich žiakov, aby zvolil tú správnu stratégiu pre smerovanie svojich žiakov.

#### Úloha 1

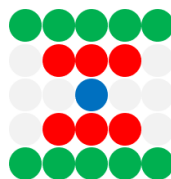
Členovia školskej firmy sa rozhodli, že ako predmet svojej činnosti budú v škole prevádzkovať automat, ktorý bude pomocou nažehľovacích korálikov vytvárať vzor, ktorý si navolí zákazník. Takto vytvorený vzor si zákazník vezme a doma nažehlí na tričko, tašku alebo čiapku.

Špecifikácia činnosti automatu je takáto:

- k dispozícii sú tri farby nažehľovacích korálikov (červená, zelená a modrá),
- vzor zadaný zákazníkom je v automate reprezentovaný v podobe reťazca, v ktorom majú jednotlivé znaky nasledovný význam:
  - 'r' korálik červenej farby,
  - 'g' korálik zelenej farby,
  - 'b' korálik modrej farby,
  - ' ' prázdne miesto vo vzore,
  - '/' nový riadok vo vzore,

- 'k' koniec vzoru.

Napríklad reťazec 'ggggg/ rrr/ b/ rrr/ggggk' reprezentuje nasledovný vzor:



- Definujte funkciu `cena_celkom()`, ktorá pre zadaný reťazec reprezentujúci vzor nažehľovacích korálikov vráti výslednú cenu. Za každý korálik červenej farby zaplatí zákazník 1 cent, za zelený korálik 1,5 centa a za modrý korálik 2 centy.
- Definujte funkciu `je_vzor_ok()`, ktorá otestuje zadaný reťazec, či je definovaný podľa pravidiel a vráti zodpovedajúcu logickú hodnotu.
- Upravte funkciu `cena_celkom()`, tak, aby v prípade, že zadaný reťazec je chybný vyhodila funkcia zmysluplnú výnimku.
- Upravte hlavný program tak, aby načítal vzor od používateľa a následne vypísal cenu, resp. chybovú správu.

Riešenie uložte do súboru **koraliky.py**.

Riešenie:

```
def je_vzor_ok(vzor): # riešenie B
    try:
        if vzor[-1] != 'k':
            return False
    except IndexError:
        return False
    povolene_znaky = 'rgb/k'
    pocet_povolenych = 0
    for znak in povolene_znaky:
        pocet_povolenych = pocet_povolenych + vzor.count(znak)
    if pocet_povolenych != len(vzor):
        return False
    return True

def cena_celkom(vzor): # riešenie A
    pocet_cervene = vzor.count('r')
    pocet_zelene = vzor.count('g')
    pocet_modre = vzor.count('b')
    cena = pocet_cervene + pocet_zelene * 1.5 + pocet_modre * 2
    cena = round(cena)
    return cena

def cena_celkom(vzor): # riešenie C
    if not je_vzor_ok(vzor):
        raise ValueError('Zadaný vzor nie je korektný.')
    pocet_cervene = vzor.count('r')
    pocet_zelene = vzor.count('g')
    pocet_modre = vzor.count('b')
    cena = pocet_cervene + pocet_zelene * 1.5 + pocet_modre * 2
    cena = round(cena)
    return cena
```

```
# riešenie D
vzor = input('Zadaj vzor korálikov: ')
try:
    print(f'Cena navrhnutého vzoru: {cena_celkom(vzor)}')
except ValueError as chyba:
    print(chyba)
```

Výslednú cenu vypočítame tak, že počty jednotlivých korálikov násobíme ich cenou. Keďže zelené koráliky stoja necelé centy, musíme výslednú sumu zaokrúhliť (matematicky, smerom hore alebo smerom dole) na celé centy.

Pri kontrole správnosti zadaného reťazca overujeme, či je na jeho konci ukončovaci znak. Pozor, ak je reťazec prázdny, odkaz na posledný znak reťazca spôsobí chybu `IndexError`. Preto tento test realizujeme v bloku `try .. except`. Následne stačí spočítať „povolené“ znaky a porovnať ich s dĺžkou reťazca. Ak niektorý z testov „zlyhá“, vraciamе hodnotu `False` (vykonávanie tela funkcie sa ukončí). Ak reťazec vyhovел všetkým testom, vraciamе hodnotu `True`.

## Úloha 2

Časť B  
riešte  
podľa  
pokynov  
učiteľa

Určite poznáte mnoho návodov, ako zistiť svoje šťastné číslo. V triede 3.A sa žiaci dohodli, že ho vypočítajú tak, že sčítajú všetky cifry vo svojom rodnom čísle.

Žiaci prišli za správcom informačného systému s prosbou: v systéme sú uložené rodné čísla (v tvare rmmdd/xxxx) každého žiaka školy – mohlo by sa každému žiakovi po prihlásení do systému vypísať jeho šťastné číslo?

Riešenie tejto úlohy realizujte v pripravenom súbore **informacny\_system.py**.

- Vytvorte funkciu `stastne_cislo()`, ktorá pre zadané rodné číslo žiaka vráti šťastné číslo žiaka.
- Občas sa stane, že rodné číslo je chybné. Definujte funkciu `je_rc_ok()`, ktorá overí, či rodné číslo je korektné a vráti zodpovedajúcu logickú hodnotu. Rodné číslo je korektné, ak má vyššie uvedený tvar, je deliteľné číslom 11 (po odstránení lomky) a dátumová časť predstavuje korektný dátum (ženám sa k mesiacu pripočíta číslo 50). Pre overenie korektnosti dátumu môžete použiť pripravenú funkciu `je_datum_ok()`.
- Upravte funkciu `stastne_cislo()` tak, aby v prípade chybného rodného čísla vygenerovala zmysluplnú výnimku.
- Upravte hlavný program tak, aby program načítal rodné číslo od používateľa a následne vypísal jeho šťastné číslo.

Riešenie:

```
def je_rc_ok(rc): # Riešenie B
    if len(rc) != 11: # testujeme dĺžku rc, musí byť 11
        return False
    if rc[6] != '/': # testujeme, či obsahuje znak / na správnej pozícii
        return False
    rc = rc[:6] + rc[7:] # odstránime / z pozície 6
    if not rc.isdigit(): # testujeme, či po odstránení ostali len číslice
        return False
    if int(rc) % 11 != 0: # testujeme deliteľnosť 11
        return False
    rok = '20' + rc[:2] # rok narodenia, predpokladáme že žiaci sa
    narodili v 20xx
    rok = int(rok)
    mesiac = rc[2:4] # mesiac narodenia
    mesiac = int(mesiac)
    if mesiac > 50: # ak ide o ženu, upravíme hodnotu mesiaca
        mesiac = mesiac - 50
```

```
den = rc[4:6] # deň narodenia
den = int(den)
if not je_datum_ok(den, mesiac, rok): # otestujeme korektnosť dátumu
    return False
return True

def stastne_cislo(rc): # Riešenie A
    rc = rc.replace('/', '')
    stastne_cislo = 0
    for cifra in rc:
        stastne_cislo = stastne_cislo + int(cifra)
    return stastne_cislo

def stastne_cislo(rc): # Riešenie C
    if not je_rc_ok(rc):
        raise ValueError('Zadané rodné číslo nie je korektné')
    rc = rc.replace('/', '')
    stastne_cislo = 0
    for cifra in rc:
        stastne_cislo = stastne_cislo + int(cifra)
    return stastne_cislo

# riešenie D
rc = input('Zadaj svoje rodné číslo: ')
try:
    print(f'Tvoje šťastné číslo je: {stastne_cislo(rc)}')
except ValueError as chyba:
    print(chyba)
```

Uvedená úloha je zameraná primárne na prácu s reťazcami. Pre jej správne vyriešenie je potrebná dôsledná analýza riešeného problému. Vyhodnotenie správnosti rodného čísla môže byť pre žiakov náročné. Žiaci by si mali premyslieť, aké podmienky musí rodné číslo spĺňať, ako podmienky otestovať prostriedkami programovacieho jazyka a v akom poradí ich testovať. Ponechávame na učiteľovi, ktoré kontroly správnosti by mali žiaci realizovať.

Pre kontrolu korektnosti dátumu môžu žiaci využiť pripravenú funkciu `je_datum_ok()`. Funkcia využíva modul `datetime`, ktorý pri vytváraní dátumového objektu prevádza rovnakú kontrolu.

## ZHRNUTIE (CCA 5 MIN)

Odporúčame, aby si žiaci do pojmovej mapy poznačili, v ktorej úlohe jednotlivé poznatky použili; aby si zapísali popis funkcií, príklady metód a podobne.

## 17 ZOZNAMY A METÓDY ZOZNAMOV

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>interpretácia zápisu riešenia.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>odchytať a generovať výnimky,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus a podmienený príkaz pri riešení problémov.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>vytvoriť zoznam priamym zápisom hodnôt do zátvoriek,</li> <li>získovať a využívať vlastnosti zoznamov pri riešení úloh.</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG8) z existujúcich pravidiel logicky odvodzovať iné pravidlá (využitie podobnosti zoznamov a reťazcov).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (riešenie problémov z pracovného listu),</li> <li>(ALG8) zapísať algoritmy v konkrétnom formálnom jazyku (zápis v programovacom jazyku).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia – lineárne rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (rozpoznávanie jednotlivých stavov problému).</li> </ul> Abstrakcia: <ul style="list-style-type: none"> <li>(ABS3) využiť podstatné prvky problémov (navrhnuť testovacie dáta pre overenie správnosti riešenia).</li> </ul>
<b>Riešený didaktický problém</b>	
<p>Pre zoznamy a reťazce používa Python množstvo rovnakých prístupov. Zoznamy nepredstavujeme žiakom ako úplne neznámy koncept, ale využívame podobnosti s reťazcami a upozorňujeme na ich rozdiely.</p> <p>Zoznam je iterovateľná dátová štruktúra. Častou chybou pri iterovaní zoznamu je prechod cez indexy zoznamu aj v situácii, keď to nie je potrebné. Tento prístup môže byť dôsledkom skúsenosti programátora z iného programovacieho jazyka.</p> <p>Zoznamy sa sémanticky používajú primárne ako homogénne dátové štruktúry. Aj keď zoznam môže obsahovať heterogénne hodnoty, v metodikách sa to snažíme nevyužívať. Zoznamy využívame v situáciách, keď</p>	

potrebujeme pracovať s nejakým, často vopred neznámym, počtom typovo rovnakých vecí.

<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• individuálna a skupinová forma práce žiakov.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/17/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/17/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.	

## Úvod

Toto je 17. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. V nej sa žiaci prvýkrát stretnú s novým dátovým typom – zoznamom. Zoznam je iterovateľná dátová štruktúra, analógiu ktorej nachádzame aj v bežnom živote človeka. Pri zoznamoch Python využíva podobné prístupy ako pri reťazcoch. Tento fakt využívame aj v metodike a žiakov na to priebežne upozorňujeme. Rovnako upozorňujeme žiakov aj na rozdiely medzi zoznamom a reťazcom.

Zoznam primárne predstavujeme ako homogénnu dátovú štruktúru (kontajner), kde prvky majú svoje poradie. Zoznam je meniteľný. Aj keď Python umožňuje zoznamy používať aj iným spôsobom (napr. nehomogénne štruktúry, kontajner s konštantným obsahom), primárne ich v metodikách používame takto.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Predpokladáme, že žiaci si už osvojili prácu s pythonovskými programami a vedia si ich vhodne organizovať. Nie je preto nutné všetky riešenia zapisovať aj do pracovných listov. Primárne by riešenia mali byť ľahko lokalizovateľné žiakom v jeho priečinkoch a vhodne komentované. Odporúčame, aby pre aspoň jednu funkciu žiaci vytvorili dokumentačný reťazec.

Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – rozhovor so žiakmi – vyvodenie myšlienky zoznamov z reálneho života, spoločná frontálna práca s pracovným listom (úloha 1).
- **Skúmanie (6 minút)** – práca vo dvojiciach s pracovným listom (úloha 2).
- **Vysvetlenie (7 minút)** – vysvetlenie zistení z časti Skúmanie.
- **Rozpracovanie (19 minút)** – samostatné programovanie náročnejších úloh (úlohy 3 až 5).
- **Vyhodnotenie (3 minúty)** – riešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 5 MIN)

Hodinu začneme diskusiou k ukážkam informácií v úlohe 1. V ukážkach je uvedených niekoľko príkladov zoznamov z bežného života. Žiaci by mali diskutovať o tom, čo majú uvedené informácie spoločné, v čom sa odlišujú a uviesť ďalšie podobné typy informácií.

Cieľom tejto aktivity je, aby si žiaci uvedomili, že zoznamy a manipulácia s nimi sú našou každodennou súčasťou. Benefity zoznamov (viac hodnôt v jednom kontajneri, možnosť modifikácie,

určené poradie a pod.) môžeme využiť aj pri programovaní. Cieľ vyučovania, zoznamy a ich využitie pri riešení úloh, predstavíme žiakom po diskusii.

**Poznámka:**

Zámerne sme v ukážkach nepoužili informácie typu „mesiace v roku“, „zoznam mien v kalendári“ a pod. Tieto informácie síce môžeme formálne vyjadriť v tvare zoznamu, ale sémanticky by bolo vhodnejšie použiť n-ticu (tuple). Uvedené informácie sú nemenné. V tejto chvíli nepovažujeme za potrebné touto informáciou žiakov zaťažovať.

Ak by žiaci v riešení úlohy 1 navrhli podobné typy informácií, akceptujte ich s poznámkou, že v týchto „zoznamoch“ sa nepredpokladá zmena.

**Úloha 1** Čo majú spoločné a čo rozdielne nasledovné typy informácií? Uvedte ďalšie podobné príklady.

Nákupný lístok:	Ranná teplota pacienta (°C):	Trasa výletu:
jogurt, 3	38,9	Parkovisko Kežmarská Biela voda
mlieko, 2	38,9	Nad Matliarmi
maslo, 1	38,6	Šalviový prameň
chlieb, 1	37,8	Kovalčíková poľana
vrecúško jablák, 1	37,8	Veľké Biele pleso
		Dolina Zeleného plesa

*Spoločné vlastnosti:*

*Rozdielne vlastnosti:*

*Ďalšie podobné príklady:*

*Riešenie:*

*Spoločné vlastnosti:*

- Sú to zoznamy nejakých, podobných hodnôt.
- Do každého môžeme zmysluplne pridať ďalšie hodnoty (zaznamenať aktuálnu rannú teplotu pacienta).
- Z každého zoznamu môžeme zmysluplne nejakú hodnotu zmazať (na chatu nepôjdeme cez Veľké Biele pleso).
- Vieme zistiť (má zmysel), či nejaká hodnota je v zozname („máš poznačené maslo?“).
- Vieme zistiť (má zmysel) poradie nejakého prvku („ktorý v poradí je bod výletu Šalviový prameň“).
- Môžeme (má zmysel) zistiť veľkosť („koľko dní je pacient u nás“).
- S uvedenými informáciami môžeme manipulovať ako s celkom (na nákup si zoberieme jeden nákupný lístok, na ktorom je uvedené všetko).

*Rozdielne vlastnosti:*

- Niekde na poradí záleží (teploty, trasa), niekde nemusí (nákup), aj pri nákupe môže byť poradie zámerne zvolené tak, aby sme v obchode absolvovali čo najkratšiu trasu.
- Niekde sú hodnoty čísla (teplota), niekde reťazce (výlet), niekde je to kombinácia (nákup).

*Ďalšie podobné príklady:* zoznam zastávok autobusu, prezenčná listina, zoznam zapísaných pacientov na ošetrovanie u lekára, zoznam žiakov triedy.



## SKÚMANIE (CCA 6 MIN)

Žiakov necháme, aby preskúmali uvedené príkazy. Ak uznajú za potrebné, môžu si v malých skupinkách prediskutovať svoje predpovede alebo závery.

**Úloha 2** V konzolovom režime realizujte nasledovné príkazy. Najskôr odhadnite, čo bude výsledkom príkazu, potom si svoj predpoklad overte.

*Pomôcka: všimnite si, že niektoré zápisy sa podobajú na zápisy, ktoré ste používali pri reťazcoch. Táto podobnosť nie je náhodná a môžete ju využiť.*

```
znamky = [2, 1, 3, 2, 3, 1, 2]
znamky
```

**predpoveď:**

**skutočnosť:**

príkaz	predpoveď:	skutočnosť:
<code>len(znamky)</code>		
<code>znamky[2]</code>		
<code>znamky[-2]</code>		
<code>znamky.count(3)</code>		
<code>znamky.index(2)</code>		
<code>5 in znamky</code>		
<code>sum(znamky)</code>		
<code>for znamka in znamky:     print(znamka)</code>		

**Riešenie:**

príkaz	skutočnosť:
<code>znamky = [2, 1, 3, 2, 3, 1, 2]</code> <code>znamky</code>	vytvorí zoznam obsahujúci uvedené známky vypíše zoznam známk
<code>len(znamky)</code>	vráti dĺžku zoznamu, počet jeho prvkov, t. j. 3
<code>znamky[2]</code>	vráti hodnotu na pozícii 2, t. j. 3
<code>znamky[-2]</code>	vráti hodnotu na pozícii -2, t. j. 1

<code>znamky.count(3)</code>	vráti počet výskytov hodnoty 3, t. j. 2
<code>znamky.index(2)</code>	vráti pozíciu hodnoty 2 v zozname, t. j. 0 ak prvkov s danou hodnotou je viac, vracia pozíciu prvého
<code>5 in znamky</code>	otestuje, či hodnota 5 je v zozname, t. j. False
<code>sum(znamky)</code>	vráti súčet prvkov v zozname, t. j. 14
<code>for znamka in znamky:     print(znamka)</code>	cyklus na postupný prechod prvkami zoznamu a ich výpis, výstup 2 1 3 2 3 1 2 (každé v novom riadky)

## VYSVETLENIE (CCA 7 MIN)

V tejto časti by mali žiaci vysvetliť svoje zistenie. Ich vysvetlenie by malo smerovať k tomu, že zoznam je dátová štruktúra (kontajner) umožňujúca udržiavať (spravovať) iné hodnoty v jednom celku. Zoznam vytvoríme tak, že hodnoty oddelené čiarkou uzatvoríme do zátvoriek `[]`. Prvky v zozname sú určené svojim indexom, pričom indexujeme zľava od 0 alebo sprava od -1. Vieme testovať, či nejaký prvok je v zozname, vieme nájsť jeho pozíciu alebo spočítať počet výskytov prvku v zozname. Vieme zistiť počet prvkov zoznamu a súčet jeho prvkov (ak sa hodnoty dajú spočítať).

Odporúčame, aby učiteľ k žiackym zisteniam uviedol aj nasledovné informácie:

- podobnosť s reťazcami:
  - indexovanie, dĺžka, počet výskytov, hľadanie pozície hodnoty, prechod cez cyklus,
- rozdiel oproti reťazcom:
  - prvky zoznamov uzatvárame do zátvoriek `[]` a oddeľujeme čiarkou, znaky reťazca uzatvárame do apostrofov `' '` a neoddeľujeme,
- ak pracujeme s indexom, používame zátvorky `[]`, ak voláme nejakú funkciu zoznamu, používame zátvorky `()` oddelené od zoznamu bodkou,
- ďalšie operácie a metódy zoznamov sú uvedené v časti „Vedomosti v kocke“.

## ROZPRACOVANIE (CCA 19 MIN)

Nasledujúce úlohy sú zamerané na prácu so zoznamami a ich prvkami. Zoznamy zámerne nemodifikujeme. Touto problematikou sa budeme zaoberať v nasledujúcej metodike. Úlohy obsahujú časti zamerané primárne na prácu so zoznamami. Niektoré z podúloh sú zamerané na prácu s testovacími dátami a výnimkami.

Jednotlivé podúlohy v úlohe 3 sú koncipované tak, aby sa riešenie úlohy postupne vylepšovalo.

### Poznámka:

Úlohu 3 je možné riešiť spôsobom postupného spočítavania prvkov v cykle. Neodporúčame tento spôsob použiť. Prednostne smerujme žiakov k využívaniu existujúcich príkazov, v tomto prípade funkcia `sum()`. Vo väčšine prípadov sú pythonovské príkazy (zabudované funkcie) efektívnejšie než vlastný implementovaný postup. Ďalším dôvodom je zjednodušenie práce programátora.

**Úloha 3** *Riešenie úlohy uložte do súboru **znamky.py**.*

Časti  
b) až c)  
riešte  
podľa  
pokynov  
učiteľa.

- Jeden zo spôsobov ako zistiť výslednú známku na vysvedčení je vypočítať a zaokrúhliť priemer všetkých známok, ktoré žiak za polrok z predmetu získal. Vytvorte funkciu `vysledna_znamka()`, ktorá pre zadaný zoznam známok vypočíta a vráti výslednú známku na vysvedčení.
- Preskúmajte vami navrhnutú funkciu. Otestujte funkciu `vysledna_znamka()`, či pre všetky vstupy pracuje korektne. Pre ktoré vstupy funkcia nepracuje správne?
- Na základe poznatkov z bodu b) upravte funkciu `vysledna_znamka()` tak, aby v prípade nekorektného vstupu funkcia vyhodila relevantnú výnimku.
- Pri hodnotení nemajú všetky známky rovnakú váhu (vplyv na výslednú známku). Posledná známka je známka zo záverečného testu a jej váha je trojnásobná. Upravte funkciu `vysledna_znamka()` tak, aby posledná známka mala trojnásobnú váhu. Môžete predpokladať, že záverečný test písal každý žiak.

**Riešenie A:**

```
def vysledna_znamka(znamky):
    priemer = sum(znamky) / len(znamky)
    znamka = round(priemer)
    return znamka
```

**Riešenie B:**

Problematické vstupy:    prázdny zoznam známok, nastane chyba delenie 0,  
                                   zoznam obsahujúci hodnoty, ktoré nie sú korektnou hodnotou známky (napr. 10),  
                                   zoznam obsahujúci hodnoty, ktoré nie sú číslom, nastane chyba typu,

**Riešenie C:**

```
def vysledna_znamka(znamky):
    if len(znamky) == 0:
        raise ValueError('Chyba! Zoznam známok je prázdny.')
    for znamka in znamky:
        if znamka not in [1, 2, 3, 4, 5]:
            raise ValueError('Chyba! Nekorektná známka.')
    priemer = sum(znamky) / len(znamky)
    znamka = round(priemer)
    return znamka
```

**Riešenie D:**

```
def vysledna_znamka(znamky):
    if len(znamky) == 0:
        raise ValueError('Chyba! Zoznam známok je prázdny.')
    for znamka in znamky:
        if znamka not in [1, 2, 3, 4, 5]:
            raise ValueError('Chyba! Nekorektná známka.')
    priemer = (2 * znamky[-1] + sum(znamky)) / (len(znamky) + 2)
    znamka = round(priemer)
    return znamka
```

**Úloha 4** Riešenie úlohy uložte do súboru **projekty.py**.

Časť b)  
riešte  
podľa  
pokynov  
učiteľa.

- a) V súťaži hodnotenia žiackych projektov hodnotia projekt jednotliví členovia komisie. Každý člen komisie pridelí projektu nejaký počet bodov. Celkové hodnotenie projektu sa vypočíta tak, že z hodnotení sa vyhodí jedno najlepšie a jedno najhoršie hodnotenie. Zo zvyšných hodnotení sa vypočíta a zaokrúhli priemer. Vytvorte funkciu `hodnotenie_projektu()`, ktorá pre zadaný zoznam hodnotení projektu členmi komisie vypočíta celkové hodnotenie zaokrúhlené na jedno desatinné miesto..
- b) Počet členov komisie je každý rok iný. Upravte funkciu `hodnotenie_projektu()` tak, aby sa v prípade malého počtu čiastkových hodnôt počítal priemer zo všetkých hodnotení. Ak by sa celkové hodnotenie ani tak nedalo vypočítať, nech funkcia vyhodí výnimku.

**Riešenie A:**

```
def hodnotenie_projektu(hodnotenia): # riešenie A
    najlepsie = max(hodnotenia)
    najhorsie = min(hodnotenia)
    sucet_bodov = sum(hodnotenia) - najlepsie - najhorsie
    pocet_hodnoteni = len(hodnotenia) - 2
    priemer = sucet_bodov / pocet_hodnoteni
    vysledok = round(priemer, 1)
    return vysledok
```

**Riešenie B:**

```
def hodnotenie_projektu(hodnotenia):
    if len(hodnotenia) == 0:
        raise ValueError('Chyba! Zoznam hodnotení je prázdny')
    if len(hodnotenia) <= 2:
        sucet_bodov = sum(hodnotenia)
        pocet_hodnoteni = len(hodnotenia)
    else:
        najlepsie = max(hodnotenia)
        najhorsie = min(hodnotenia)
        sucet_bodov = sum(hodnotenia) - najlepsie - najhorsie
        pocet_hodnoteni = len(hodnotenia) - 2
    priemer = sucet_bodov / pocet_hodnoteni
    vysledok = round(priemer, 1)
    return vysledok
```

Úlohu 5 je možné riešiť viacerými spôsobmi. Riešenia sa môžu líšiť efektívnosťou. Pokiaľ riešenie nebudú príliš komplikované alebo neefektívne, odporúčame akceptovať žiacke riešenia.

Prehľadávanie zoznamu mien je možné realizovať cyklom. Výhodnejšie je využiť funkciu zoznamu `index()`. Toto riešenie je efektívnejšie a navyše sa v riešení c) vyhneme vnoreniu dvoch cyklov do seba.

Pri výpočte času (napr. minúta objednania, minúta obedňajšej prestávky) je v autorskom riešení použitý výraz namiesto hodnoty celého výrazu. Odporúčame tento prístup dodržať. Ak použijeme len výslednú hodnotu, zrozumiteľnosť programu sa zníži, pretože nie je jasné, ako sme sa k danej hodnote dopracovali.

**Úloha 5** Riešenie úlohy uložte do súboru **pacienti.py**.

Časti

- a) Sestrička zaznamenáva mená pacientov do zoznamu tak, ako sa príbežne telefonicky objednávajú.

b) a c)  
riešte  
podľa  
pokynov  
učiteľa.

- Občas sa stane, že pacient zavolá a chce vedieť, kedy sa dostane na rad. Aj keď výpočet nie je komplikovaný, sestričku to zbytočne zdržiava. Vytvorte funkciu `zisti_cas()`, ktorá pre zadané meno a zoznam pacientov vráti čas (ako reťazec `h:m`), kedy sa pacient dostane na vyšetrenie.
- b) Občas zavolá pacient, ktorý si pomýli deň a v zozname pacientov jeho meno nie je. Upravte funkciu `zisti_cas()` tak, aby v takomto prípade vrátila hodnotu `pacient neobjednaný`.
- c) Sestrička si do zoznamu značí len priezviská pacientov. Občas sa stane, že sa v jeden deň objednávajú menovci. Upravte funkciu `zisti_cas()` tak, aby v takomto prípade vrátila všetky časy, kedy pacienti s týmto meno prídu na rad.

Predpokladajte, že lekár začína ordinovať o 8:00, na každého pacienta si rezervuje 20 min a v čase 13:00 – 13:30 má obedňajšiu prestávku.

V súbore **pacienti.py** je pripravený zoznam pacientov na testovanie.

#### Riešenie A:

```
def zisti_cas(meno, pacienti):
    poradie = pacienti.index(meno)
    cas = 8 * 60 + poradie * 20
    if cas > 12 * 60 + 40:
        cas = cas + 30
    hodiny = cas // 60
    minuty = cas % 60
    return f'{hodiny}:{minuty}'
```

#### Riešenie B:

```
def zisti_cas(meno, pacienti):
    try:
        poradie = pacienti.index(meno)
    except ValueError:
        return 'pacient neobjednaný'
    cas = 8 * 60 + poradie * 20
    if cas > 12 * 60 + 40:
        cas = cas + 30
    hodiny = cas // 60
    minuty = cas % 60
    return f'{hodiny}:{minuty}'
```

#### Riešenie C:

```
def zisti_cas(meno, pacienti):
    pocet_menovcov = pacienti.count(meno)
    if pocet_menovcov == 0:
        return 'pacient neobjednaný'
    od_index = 0
    vysledok = ''
    for i in range(pocet_menovcov):
        pozicia = pacienti.index(meno, od_index)
        od_index = pozicia + 1
        cas = 8 * 60 + pozicia * 20
        if cas > 12 * 60 + 40:
            cas = cas + 30
        hodiny = cas // 60
        minuty = cas % 60
        vysledok = vysledok + f'{hodiny}:{minuty}\n'
    return vysledok
```

## VYHODNOTENIE (CCA 3 MIN)

Následne požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal a nie klasifikácia známok. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

### Sebahodnotiaci test

1.	<p>Tinka dostala z informatiky nasledovné známky: 1, 2, 1, 3, 2, 2.</p> <p>Zoznam známok Tinky z informatiky vytvoríme nasledovne: Vyberte správne možnosti.</p> <div><div><p>a)</p><pre>znamky = [1, 2, 1, 3, 2, 2]</pre></div><div><p>b)</p><pre>znamky = (1, 2, 1, 3, 2, 2)</pre></div><div><p>c)</p><pre>znamky = [1, 1, 2, 2, 2, 3]</pre></div><div><p>d)</p><pre>znamky = 1, 2, 1, 3, 2, 2</pre></div></div>																				
2.	<p>Čo je výstupom nasledovného programu?</p> <pre>data = [1, 2, 3, 2, 1] for hodnota in data:     print(hodnota, data.index(hodnota))</pre> <div><div><p>a)</p><table><tr><td>1 1</td></tr><tr><td>2 2</td></tr><tr><td>3 3</td></tr><tr><td>2 4</td></tr><tr><td>1 5</td></tr></table></div><div><p>b)</p><table><tr><td>1 0</td></tr><tr><td>2 1</td></tr><tr><td>3 2</td></tr><tr><td>2 1</td></tr><tr><td>1 0</td></tr></table></div><div><p>c)</p><table><tr><td>1 0</td></tr><tr><td>2 1</td></tr><tr><td>3 2</td></tr><tr><td>2 3</td></tr><tr><td>1 4</td></tr></table></div><div><p>d)</p><table><tr><td>1 1</td></tr><tr><td>2 2</td></tr><tr><td>3 3</td></tr><tr><td>2 2</td></tr><tr><td>1 1</td></tr></table></div></div>	1 1	2 2	3 3	2 4	1 5	1 0	2 1	3 2	2 1	1 0	1 0	2 1	3 2	2 3	1 4	1 1	2 2	3 3	2 2	1 1
1 1																					
2 2																					
3 3																					
2 4																					
1 5																					
1 0																					
2 1																					
3 2																					
2 1																					
1 0																					
1 0																					
2 1																					
3 2																					
2 3																					
1 4																					
1 1																					
2 2																					
3 3																					
2 2																					
1 1																					

V úlohe 1 môžeme uznať správne riešenia a) aj c). Upozorníme však žiakov, že pri možnosti c) prichádzame o poradie známok, čo v niektorých prípadoch môže byť dôležité (napr. ak poradie známok určuje ich váhu).

V úlohe 2 by si žiaci mali uvedomiť, že poradie hodnôt v zozname sa čísluje od 0 a funkcia zoznamu `index()` vracia index prvého výskytu hodnoty v zozname.

# 18 VYTVÁRANIE A MODIFIKÁCIA ZOZNAMOV, POUŽITIE NÁHODY

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov,</li> <li>interpretácia zápisu riešenia.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>odchytať a generovať výnimky,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus (na prechod zoznamom) a podmienený príkaz pri riešení problémov,</li> <li>vytvoriť zoznam priamym zápisom hodnôt do zátvoriek,</li> <li>získovať a využívať vlastnosti zoznamov pri riešení úloh.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>vytvárať zápisy a interpretovať zápisy podľa nových stanovených pravidiel (syntaxe) pre zápis algoritmov.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>vytvárať a modifikovať zoznamy.</li> </ul>	Koncepty informatického myslenia  <b>Logika:</b> <ul style="list-style-type: none"> <li>(LOG8) z existujúcich pravidiel logicky odvodzať iné pravidlá (využitie podobnosti zoznamov a reťazcov).</li> </ul> <b>Algoritmy:</b> <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (riešenie problémov z pracovného listu).</li> </ul> <b>Dekompozícia:</b> <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia – lineárne rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (rozpoznávanie jednotlivých stavov problému).</li> </ul> <b>Abstrakcia:</b> <ul style="list-style-type: none"> <li>(ABS2) z konkrétnych prípadov (inštancií) problémov abstrahovať postupy (konkrétne úpravy zoznamov nahradiť funkciami pre úpravu zoznamov),</li> <li>(ABS3) využiť podstatné prvky problémov (navrhnuť testovacie dáta pre overenie správnosti riešenia).</li> </ul>
<b>Riešený didaktický problém</b>	
<p>Pre zoznamy a reťazce používa Python množstvo rovnakých prístupov (napr. výrezy). Pri predstavovaní funkcií zoznamov, ktoré sú na rozdiel od reťazcov meniteľné, upozorňujeme na rozdiely medzi nimi.</p> <p>Niektoré úpravy zoznamov je možné robiť aj na nižšej úrovni. V Pythone, ktorý obsahuje „silné“ nástroje pre</p>	

úpravu zoznamov, to považujeme za kontraproduktívne a žiakov primárne smerujeme k použitiu funkcií zoznamov.

Náhoda sa pri programovaní často predstavuje ako „zábavný“ prvok v samoúčelných úlohách. Sme presvedčení, že v tejto etape vyučovania programovania sú žiaci už natoľko vyspelí, aby pochopili užitočnosť pseudonáhodného generátora pri riešení problémov (napr. pri simulácií reálnych dejov alebo tvorbe testovacích dát).

<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• individuálna a skupinová forma práce žiakov.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/18/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/18/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.	





## Úvod

Toto je 18. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. V tejto metodike sa žiaci oboznámia s funkciami zoznamov pre ich zmenu.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Predpokladáme, že žiaci si už osvojili prácu s pythonovskými programami a vedia si ich vhodne organizovať. Nie je preto nutné všetky riešenia zapisovať aj do pracovných listov. Primárne by riešenia mali byť ľahko lokalizovateľné žiakom v jeho priečinkoch a vhodne komentované. Odporúčame, aby pre aspoň jednu funkciu žiaci vytvorili dokumentačný reťazec.

Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – rozhovor so žiakmi o reálnych situáciách, kde je potrebné modifikovať zoznamy (úloha 1).
- **Skúmanie (8 minút)** – skúmanie nových príkazov (úloha 2).
- **Vysvetlenie (5 minút)** – vysvetlenie zistení z časti Skúmanie.
- **Rozpracovanie (17 minút)** – samostatné programovanie náročnejších úloh (úlohy 3 až 5).
- **Vyhodnotenie (5 minút)** – kontrola žiackych prác, vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 5 MIN)

V predchádzajúcej metodike sme využívali zoznamy len ako statické (nemenné) kontajnery na dáta. Možnosť zmeny obsahu kontajnera sme pred žiakmi síce neskrývali, ale v riešených úlohách sme to nevyužívali.

V úlohe 1 sú formulované situácie, v ktorých sa vyžaduje modifikácia obsahu nejakého zoznamu. Žiaci by si mali uvedomiť, že zmena reálne využívaných zoznamov je pomerne častá. Situácie sú formulované tak, aby priamo smerovali ku konkrétnym metódam zoznamov.

**Úloha 1** *Nižšie sú popísané konkrétne situácie, kde využívame zoznamy. Popíšte činnosti, ktoré by sa mali s uvedenými zoznamami realizovať v konkrétnej situácii.*

**Nákupný zoznam: zubná niť, mlieko, cibuľa, biely chlieb, mydlo, prací prášok.**

Minuli sa nám aj špagety, je potrebné ich kúpiť.

Dopíšeme do zoznamu špagety. Najrýchlejšie to

	bude, ak ich dopíšeme na koniec zoznamu.
Kúpme namiesto bieleho chleba radšej čierny chlieb, je zdravší.	Nájdeme v zozname biely chlieb a zmeníme ho na čierny chlieb.
Mydlo nekupuj, ešte ho máme!	Zmažeme mydlo zo zoznamu.
Do tabuľky si kvôli dlhodobému prehľadu značíme všetky nákupy. Kvôli jednoduchšej orientácii sú položky v tabuľke uvedené podľa abecedy. Ako upraviť nákupný zoznam, aby sme položky z neho čo najrýchlejšie zaznamenali do tabuľky.	Usporiadame položky v nákupe podľa abecedy.
Susedka nás požiadala, aby sme jej nakúpili suroviny na koláč podľa jej zoznamu.	Pridáme jej zoznam k nášmu zoznamu.
Aby sme nákup realizovali čo najrýchlejšie, rozdelíme sa!	Rozdelíme nákupný zoznam na dve časti. Najlepšie na polovice. Ak sa to nedá, jeden bude mať o jednu položku viac.

## SKÚMANIE (CCA 15 MIN)

V tejto časti predstavíme žiakom príkazy a metódy na vytváranie a modifikáciu zoznamov. S vytvoreným zoznamom postupne realizujeme niekoľko úprav. Aby žiaci zmeny zoznamu ľahšie priradili k relevantným príkazom, je uvedený kód a výpis zmien rozdelený do očíslovaných blokov. Žiaci by mali v každom bloku určiť neznámy príkaz a po spustení (prípadne aj opakovaným spúšťaním s upraveným zoznamom) určiť jeho funkcionálnosť. Aby žiaci len mechanicky neprepísali svoje odpovede z úlohy 1, zmenili sme poradie úprav zoznamu.

**Úloha 2** Otvorte súbor **zoznamy.py**. V súbore je niekoľko blokov príkazov, ktoré pracujú so zoznamami. Bloky postupne okomentujte a identifikujte v každom bloku nový, pre vás neznámy príkaz. Spustite uvedený kód a poznačte si, názov nového príkazu a jeho funkcionálnosť.

<pre>nakup = ['zubná nit', 'mlieko', 'cibuľa', 'biely chlieb', 'mydlo', 'prací prášok'] print(nakup)</pre>	
<pre>print('Blok 1') pozicia = nákup.index('biely chlieb') nakup[pozicia] = 'čierny chlieb' print(nakup)</pre>	<pre>nakup[pozicia] = 'čierny chlieb'</pre> <p>zmenili sme hodnotu na danej pozícii v zoznam</p>
<pre>print('Blok 2') nakup.append('špagety') print(nakup)</pre>	<pre>nakup.append('špagety')</pre> <p>vloží hodnotu na koniec zoznamu, príkaz nič nevracia, len modifikuje existujúci zoznam, výsledok príkazu nemusíme nikam priraďovať</p>
<pre>print('Blok 3') nakup.sort() print(nakup)</pre>	<pre>nakup.sort()</pre>

	usporiada prvky zoznamu podľa abecedy, znaky s diakritikou usporiada zvlášť
<pre>print('Blok 4') nakup.remove('mydlo') print(nakup)</pre>	<code>nakup.remove('mydlo')</code>  odstráni uvedený prvok zo zoznamu
<pre>print('Blok 5') stred = len(nakup) // 2 nakup1 = nakup[:stred] nakup2 = nakup[stred:] print(nakup) print(nakup1) print(nakup2)</pre>	<code>nakup1 = nakup[:stred]</code> <code>nakup2 = nakup[stred:]</code>  vytvorí nový zoznam ako výrez z pôvodného zoznamu, pôvodný zoznam sa nezmenil, podobne to fungovalo aj pri reťazcoch
<pre>print('Blok 6') susedka_nakup = ['múka', 'kvasnice'] nakup.extend(susedka_nakup) print(nakup)</pre>	<code>nakup.extend(susedka_nakup)</code>  pridá prvky zadaného zoznamu do pôvodného zoznamu

## VYSVETLENIE (CCA 15 MIN)

V tejto časti by mali žiaci vysvetliť svoje zistenie. Ich vysvetlenie by malo smerovať k tomu, že obsah zoznamu je možné pomocou funkcií zoznamu modifikovať a z existujúceho zoznamu je možné vytvárať pomocou výrezov ďalšie zoznamy:

- pomocou funkcie zoznamu `append()` vieme pridať hodnotu na jeho koniec,
- vieme zmeniť hodnotu na konkrétnom mieste zoznamu (`nakup[pozicia] = 'čierny chlieb'`),
- vieme usporiadať prvky zoznamu metódou zoznamu `sort()`, táto funkcia však nezohľadňuje poradie znakov s diakritikou tak, ako je to uvedené v našej abecede, pri číselných zoznamoch tento problém nie je,
- pomocou funkcie zoznamu `remove()` vieme odstrániť prvok zo zoznamu,
- pomocou funkcie zoznamu `extend()` vieme pridať do zoznamu (na jeho koniec) prvky iného zoznamu,
- vieme vytvoriť nový zoznam ako výrez z nejakého iného zoznamu (podobne ako pri reťazcoch).

Odporúčame, aby učiteľ k žiackym zisteniam uviedol/zdôraznil aj nasledovné informácie:

- funkcia `remove()` odstráni len prvý výskyt hodnoty v zozname, ak hodnota v zozname nie je, vyhodí výnimku,
- zoznamy sú na rozdiel od reťazcov meniteľné,
  - pri reťazcoch sa „zmena“ prejavila vznikom nového reťazca, napr.:  
`novy_retezec = retazec.replace('y', '_')`,
  - pri zoznamoch sa zmena prejaví v pôvodnom zozname, napr.:  
`zoznam.remove('mydlo')`
- ďalšie metódy zoznamov sú uvedené v časti „Vedomosti v kocke“.

## ROZPRACOVANIE (CCA 25 MIN)

**Úloha 3** Priemerná denná teplota sa vypočíta na základe troch meraní teploty počas dňa (7:00, 14:00 a 21:00.) nasledovne:  $t_{\text{denná}} = (t_7 + t_{14} + 2 * t_{21}) / 4$ . Meteorologická stanica zaznamenáva teplotu v dané hodiny a ukladá ich do príslušných zoznamov: `teploty7`, `teploty14` a `teploty21`.

- a) Vytvorte funkciu `denne_teploty()`, ktorá pre zadané zoznamy teplôt z meraní vráti zoznam priemerných denných teplôt za sledované obdobie. Riešenie uložte do súboru **teploty.py**. Predpokladajte, že záznamy teplôt sú rovnako dlhé.
- b) Upravte funkciu `denne_teploty()` tak, aby v prípade rozdielných dĺžok vstupných zoznamov funkcia vyhodila zmysluplnú výnimku. Riešenie si overte.

Riešenie:

```
def denne_teploty(teploty7, teploty14, teploty21): # Riešenie A
    teploty = []
    for index in range(len(teploty7)):
        denna = (teploty7[index] + teploty14[index] + 2 * teploty21[index]) / 4
        denna = round(denna, 2)
        teploty.append(denna)
    return teploty

t7 = [-7, -9, -17, -10, -6, 5]
t14 = [-1, -1, -10, -6, 0, 1]
t21 = [-2, -7, -4, -5, 2, 0]
print(denne_teploty(t7, t14, t21))

def denne_teploty(teploty7, teploty14, teploty21): # Riešenie B
    if not len(teploty7) == len(teploty14) == len(teploty21):
        raise ValueError('Záznamy teplôt majú rôzne počty prvkov')
    teploty = []
    for index in range(len(teploty7)):
        denna = (teploty7[index] + teploty14[index] + 2 * teploty21[index]) / 4
        denna = round(denna, 2)
        teploty.append(denna)
    return teploty

t7 = [-7, -9, -17, -10, -6, 5, 6]
t14 = [-1, -1, -10, -6, 0, 1]
t21 = [-2, -7, -4, -5, 2, 0]
try:
    print(denne_teploty(t7, t14, t21))
except ValueError as chyba:
    print(chyba)
```

V nasledujúcej úlohe sa žiaci oboznámia s jednoduchým použitím generátora pseudonáhodných čísel. Spôsob jeho použitia je uvedený priamo v zadaní úlohy.

**Úloha 4** Žiaci na hodine matematiky preberali tému štatistika. Janku zaujala problematika náhody a rozhodla sa, že na riešenie domácej úlohy využije počítač.

Zadanie domácej úlohy z matematiky: Urči si postupnosť troch čísel z intervalu 1 .. 6. Hod' hracou kockou 100 krát. Zaznamenaj si jednotlivé čísla, ktoré na kocke padli. Zisti nasledovné informácie:

- 1 Padli čísla z tvojej trojice ihneď za sebou a v rovnakom poradí?
- 2 Padli čísla z tvojej trojice ihneď za sebou, ale v opačnom poradí?

Vytvorte funkcie, ktoré môže Janka pri riešení domácej úlohy využiť.

- a) Vytvorte funkciu `generuj_hody()`, ktorá vráti zoznam 100 náhodných hodov kockou.
- b) Vytvorte funkciu `padla_trojica()`, ktorá pre zadaný zoznam hodov a určenú trojicu zistí, či táto

trojica padla.

Riešenie uložte do súboru **kocky.py**.

Na generovanie hodu môžete využiť modul `random` a funkciu `randrange()`. Funkcia `randrange()` sa používa nasledovne:

```
import random

# vygeneruje náhodné celé číslo z intervalu <1, 6>
cislo = random.randrange(1, 7)
print(cislo)
```

Riešenie:

```
import random

def generuj_hody():
    hody = []
    for i in range(100):
        hod = random.randrange(1, 7)
        hody.append(hod)
    return hody

def padla_trojica(hody, trojica):
    for index in range(len(hody) - 3):
        if hody[index : index + 3] == trojica:
            return True
    return False

hody = generuj_hody(100)
trojica = [6, 6, 6]
print(f'trojica {trojica} padla: {padla_trojica(hody, trojica)}')
print(f'trojica {trojica[::-1]} padla: {padla_trojica(hody, trojica[::-1])}')
```

Pri hľadaní, či sa zadaná trojica vyskytuje v zozname hodnôt budú žiaci pravdepodobne postupovať spôsobom:

```
def padla_trojica(hody, trojica):
    for index in range(len(hody) - 3):
        if hody[index] == trojica[0] and \
            hody[index + 1] == trojica[1] and \
            hody[index + 2] == trojica[2]:
            return True
    return False
```

Odporúčame žiakov smerovať k riešeniu, v ktorom vytvárame trojprvkové výrezy zo zoznamu hodov a tie priamo porovnávame s hľadanou trojicou. Takéto riešenie je prehľadnejšie a pri prípadnej zmene hľadanej postupnosti menej prácne na úpravu.

Podobne pri hľadaní opačnej trojice uprednostníme programové vytvorenie opačnej trojice (výrez, alebo funkcia zoznamu `reverse()`) pred ručným definovaním novej trojice.

**Úloha 5** Na tanečnom krúžku učiteľka tanca páruje dvojice „chlapec, dievča“ podľa toho, v akom poradí žiaci na hodinu tanca prišli. Chlapci sa pri príchode zapisujú do jedného a dievčatá do druhého zoznamu. Vytvorte program **tanec.py** a v ňom funkciu `paruj()` podľa nasledovných požiadaviek:

Riešte  
podľa  
pokynov  
učiteľa

- Funkcia `paruj()` pre zadané zoznamy chlapcov a dievčat vráti zoznam dvojíc „chlapec, dievča“, ktoré budú spolu tancovať. Predpokladajte rovnaký počet chlapcov a dievčat.
- Upravte funkciu `paruj()` tak, že nebudete predpokladať rovnaké počty chlapcov a dievčat. Žiaci, ktorým sa pár nenájde, nebudú tancovať.
- Upravte funkciu `paruj()` tak, že žiaci, ktorí nemajú pár opačného pohlavia sa budú párovať medzi sebou „chlapec, chlapec“ alebo „dievča, dievča“.
- Upravte funkciu `paruj()` tak, aby sa žiaci do dvojíc vyberali náhodne. Najskôr dvojice rozdielneho pohlavia, potom dvojice rovnakého pohlavia.
- Upravte funkciu `paruj()` tak, aby sa žiaci do dvojíc vyberali v poradí podľa abecedy. Najskôr dvojice rozdielneho pohlavia, potom dvojice rovnakého pohlavia.

Na náhodné preusporiadanie prvkov zoznamu môžete využiť funkciu `shuffle()` v module `random`.

```
import random

zoznam = [1, 2, 3]
# náhodne preusporiadame prvky zoznamu
random.shuffle(zoznam)
print(zoznam)
```

Riešenie:

```
def paruj(chlapci, dievcata): # Riešenie A
    pary = []
    for index in range(len(chlapci)):
        # vytvárame dvojicu [chlapec, dievča]
        par = [chlapci[index], dievcata[index]]
        pary.append(par)
    return pary

def paruj(chlapci, dievcata): # Riešenie B
    pary = []

    # zistíme koho je menej a toľko dvojíc vytvoríme
    if len(chlapci) < len(dievcata):
        pocet = len(chlapci)
    else:
        pocet = len(dievcata)

    for index in range(pocet):
        # vytvárame dvojicu [chlapec, dievča]
        par = [chlapci[index], dievcata[index]]
        pary.append(par)
    return pary

def paruj(chlapci, dievcata): # Riešenie C
    pary = []
    # zistíme koho je menej a toľko dvojíc vytvoríme
    if len(chlapci) < len(dievcata):
        pocet = len(chlapci)
    else:
        pocet = len(dievcata)
```

```
for index in range(pocet):
    par = [chlapci[index], dievcata[index]]
    pary.append(par)

# do zoznamu zostávajúci umiestnime nezaradených žiakov
if len(chlapci) < len(dievcata):
    zostavajuci = dievcata[pocet:]
else:
    zostavajuci = chlapci[pocet:]

# ak je počet zostávajúcich nepárny, posledného žiaka vynecháme
if len(zostavajuci) % 2 == 1:
    zostavajuci = zostavajuci[:-1]

# postupne párujeme nezaradených žiakov
for index in range(0, len(zostavajuci), 2):
    par = zostavajuci[index:index+2]
    pary.append(par)
return pary

import random

def paruj(chlapci, dievcata): # Riešenie D E
    pary = []
    # náhodne preusporiadame žiakov v zoznamoch
    random.shuffle(chlapci)
    random.shuffle(dievcata)
    # alebo
    # žiakov usporiadame podľa abecedy
    # chlapci.sort()
    # dievcata.sort()

    # zistíme koho je menej a toľko dvojíc vytvoríme
    if len(chlapci) < len(dievcata):
        pocet = len(chlapci)
    else:
        pocet = len(dievcata)
    for index in range(pocet):
        par = [chlapci[index], dievcata[index]]
        pary.append(par)

    # do zoznamu zostávajúci umiestnime nezaradených žiakov
    if len(chlapci) < len(dievcata):
        zostavajuci = dievcata[pocet:]
    else:
        zostavajuci = chlapci[pocet:]

    # ak je počet zostávajúcich nepárny, posledného žiaka vynecháme
    if len(zostavajuci) % 2 == 1:
        zostavajuci = zostavajuci[:-1]

    # postupne párujeme nezaradených žiakov
    for index in range(0, len(zostavajuci), 2):
        par = zostavajuci[index:index+2]
        pary.append(par)
```

```
return pary
```

Úloha je zameraná na použitie funkcií zoznamov a manipuláciu so zoznamami. Výsledkom úloh je zoznam zoznamov, ktorý vznikne postupným spájaním prvkov z dvoch zoznamov. S podobným konceptom sa žiaci už stretli v predchádzajúcej metodike. Odporúčame, aby žiaci najskôr vnútorný zoznam vytvorili a potom ho vložili do vonkajšieho zoznamu. Práca s takýmto prvkom sa v princípe nelíši od prvkov iného typu. Jednotlivé podúlohy sú len rôzne variácie ako zadania. Ponechávame na učiteľovi, ktoré časti pre svojich žiakov vyberie.

V tejto úlohe žiakom predstavíme ďalšiu z funkcií modulu `random`. Pri náhodnom párovaní do dvojíc, môžu niektorí žiaci smerovať riešenie k náhodnému výberu prvkov z jednotlivých zoznamov a ich následne vymazanie. Je to komplikovanejšie riešenie vyžadujúce násobný prechod zoznamom, preto ho neodporúčame.

#### Poznámka:

Uvádame len niekoľko funkcií z modulu `random`. Niektorým učiteľom môže chýbať „základná“ funkcia `random()`, pomocou ktorej sa dajú „vyskladať“ ostatné funkcie pre prácu s náhodou. Nepovažujeme však za potrebné ju žiakom ukazovať a vytvárať pomocou nej ďalšie funkcie, pretože Python tieto funkcie už má definované.

## VYHODNOTENIE (CCA 5 MIN)

Následne požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal a nie klasifikácia známku. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

#### Sebahodnotiaci test

1.	<p>V zozname <code>mena</code> sú uvedené mená žiakov triedy. Zmenu poradia mien tak, aby boli uvedené v opačnom poradí zrealizujeme nasledovne:</p> <div> <div>a) <code>mena.reverse()</code></div> <div>b) <code>mena = mena.reverse()</code></div> <div>c) <code>mena[::-1]</code></div> <div>d) <code>mena = mena[::-1]</code></div> </div>
2.	<p>Aký je výstup nasledovného kódu:</p> <pre>z = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] print(z[1:7:2])</pre> <div>[2, 4, 6]</div>

Všetky odpovede v prvej úlohe nejakým spôsobom „otáčajú“ poradie prvkov v zozname. Správna odpoveď je len možnosť a). Výsledkom v d) je zoznam, ktorý obsahuje očakávané poradie prvkov.



Nie je to však pôvodný zoznam, ale jeho kópia, ktorú sme pomenovali menom pôvodného zoznamu. Odporúčame usmerniť žiakov tak, aby používali možnosť a)

Druhá úloha je zameraná na overenie pochopenia spôsobu, ako sa vytvárajú výrezy zo zoznamov. Výsledkom výrezu je zoznam prvkov, nie len jeho prvky. Odpoveď „2, 4, 6“ (bez zátvoriek) preto nie je správna. Odporúčame žiakom pripomenúť význam hodnôt uvedených vo výreze.

## 19 ALGORITMY SO ZOZNAMAMI

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>jazyk na zápis riešenia,</li> <li>pomocou postupnosti príkazov.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>odchytávať a generovať výnimky,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus (na prechod zoznamom) a podmienený príkaz pri riešení problémov,</li> <li>vytvoriť zoznam priamym zápisom hodnôt do zátvoriek,</li> <li>získovať a využívať vlastnosti zoznamov pri riešení úloh,</li> <li>priamo vytvárať zoznamy a modifikovať zoznamy pomocou funkcií zoznamov.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>vytvárať zápisy a interpretovať zápisy podľa nových stanovených pravidiel (syntaxe) pre zápis algoritmov.</li> </ul> <b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>riešiť problém skladaním príkazov do postupnosti,</li> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>vytvárať zoznamy pomocou generátorovej notácie zoznamu.</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG8) z existujúcich pravidiel logicky odvádzať iné pravidlá (využitie podobnosti zoznamov a reťazcov).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (riešenie problémov z pracovného listu).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia – lineárne rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (rozpoznávanie jednotlivých stavov problému).</li> </ul> Abstrakcia: <ul style="list-style-type: none"> <li>(ABS2) z konkrétnych prípadov (inštancií) problémov abstrahovať postupy (konkrétne úpravy zoznamov nahradiť funkciami pre úpravu zoznamov).</li> </ul>
<b>Riešený didaktický problém</b>	
Zoznamy sú pravdepodobne najuniverzálnejšia dátová štruktúra v jazyku Python. Jedna z možností ako zoznamy efektívne vytvárať je použitie generátorovej notácie. Využívajú sa pritom iné sekvencie a ich prvky. Aj keď je možné tieto situácie riešiť využitím cyklu (alebo cyklov) a s prípadnou vnorenou podmienkou (alebo podmienkami) je generátorová notácia zväčša ľahšie čitateľnejšia a s kratším zápisom.	
<b>Dominantné vyučovacie metódy a formy</b>	<b>Príprava učiteľa a pomôcky</b>
<ul style="list-style-type: none"> <li>Bádateľská metóda (model 5E),</li> <li>individuálna a skupinová forma práce žiakov.</li> </ul>	Pre učiteľa: <ul style="list-style-type: none"> <li><b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> </ul>

	<ul style="list-style-type: none"><li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li><li>• <b>ucitel/pracovne_subory_riesenia/19/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li></ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"><li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li><li>• <b>ziak/pracovne_subory/19/</b> pracovné súbory pre žiaka.</li></ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<b>Diagnostika splnenia vzdelávacích cieľov</b>	
Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.	

## Úvod

Toto je 19. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. V tejto metodike sa žiaci oboznámia s generátorovou notáciou zoznamu.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Predpokladáme, že žiaci si už osvojili prácu s pythonovskými programami a vedia si ich vhodne organizovať. Nie je preto nutné všetky riešenia zapisovať aj do pracovných listov. Primárne by riešenia mali byť ľahko lokalizovateľné žiakom v jeho priečinkoch a vhodne komentované. Odporúčame, aby pre aspoň jednu funkciu žiaci vytvorili dokumentačný reťazec.

Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – rozhovor so žiakmi o reálnych situáciách, kde je potrebné modifikovať zoznamy (úloha 1).
- **Skúmanie (9 minút)** – skúmanie nových príkazov (úloha 2).
- **Vysvetlenie (5 minút)** – vysvetlenie zistení z časti Skúmanie.
- **Rozpracovanie (16 minút)** – samostatné programovanie náročnejších úloh (úlohy 3 až 5).
- **Vyhodnotenie (5 minút)** – kontrola žiackych prác, vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 5 MIN)

V predchádzajúcich dvoch metodikách sme využívali zoznamy. Buď ako statické (nemenné) kontajnery na dáta alebo sme menili už existujúce zoznamy pomocou ich funkcií. V tejto metodike predstavíme žiakom efektívny spôsob vytvárania zoznamov pomocou generátorovej notácie. Aj napriek odstrašujúcemu názvu ide o ľahko pochopiteľný koncept, ktorý šetrí množstvo potrebného kódu. Podobnosť s generátorovou notáciou nachádzame, napr. v matematike, pri definovaní množín prvkov s danou vlastnosťou.

V úlohe 1 sú formulované situácie, v ktorých sa vyžaduje vytvorenie nového zoznamu využitím prvkov iného zoznamu. Podobné zmeny realizujeme aj v reálnych situáciách. Situácie sú formulované tak, aby priamo smerovali ku konkrétnym spôsobom vytvárania zoznamov.

**Úloha 1** Nižšie sú popísané konkrétne situácie, kde využívame zoznamy. Popíšte činnosti, ktoré by sme mali realizovať v konkrétnej situácii.

Situácia:	Potrebné činnosti na vyriešenie situácie:
Zuzka má zoznam slov, pomocou ktorých vytvára krížovky. Zistila však, že omnoho jednoduchšie použije kratšie slová než tie dlhé. Ako by mala postupovať, ak by si chcela vytvoriť len zoznam krátkych slov?	Môže postupne prechádzať pôvodným zoznamom slov a ak je slovo krátke, zapísať si ho do nového zoznamu krátkych slov.
Predškôlčica Hanka dostala pri zápise do základnej školy zoznam pomôcok, ktoré bude potrebovať. Jej škôlkarsky spolužiak Jakubko, ktorý bol v tom čase chorý, takýto zoznam nemá. Ako doceliť, aby aj Jakubko mal zoznam rovnakých pomôcok?	Stačí zoznam prekopírovať alebo prepísať položky z Hankinho zoznamu do Jakubkovho.
Andrej si na internetovom e-shope vyhliadol niekoľko hračiek a ich ceny si poznačil do zoznamu. Až neskôr si všimol, že ceny boli uvedené bez DPH. Ako by mal vytvoriť nový zoznam cien, v ktorom by boli ceny aj s DPH (20 %)?	Stačí, ak ku každej cene z pôvodného zoznamu Andrej pripočíta DPH a takto upravenú cenu uloží do nového zoznamu.

## SKÚMANIE (CCA 9 MIN)

V tejto časti predstavíme žiakom konštrukciu generátorovej notácie, pomocou ktorej vieme vytvárať nové zoznamy využitím prvkov iných zoznamov. Aby žiaci ľahšie priradili kód k výstupu, je uvedený kód a výpis zmien rozdelený do očíslovaných blokov. Žiaci by mali v každom bloku príkazov popísať, akú činnosť sme pomocou príkazov vykonali a určiť, aký je vzťah medzi pôvodným a novovytvoreným zoznamom. Odporúčame, aby žiaci s programovým kódom experimentovali a tak si lepšie potvrdili svoje hypotézy. Aby žiaci len mechanicky neprepísali svoje odpovede z úlohy 1, zmenili sme poradie problémov a pridali sme ďalšie.

**Úloha 2** Otvorte súbor **zoznamy.py**. V súbore je niekoľko blokov príkazov, ktoré pracujú so zoznamami. Bloky postupne okomentujte a identifikujte v každom bloku nový, pre vás neznámy príkaz. Spustite uvedený kód a zistite, čo je výsledkom neznámeho príkazu. V popise výsledku identifikujte vzťah medzi novým a pôvodným zoznamom.

```
print('Blok 1')
janka_ulohy = ['umyt' riad, 'vyniest' smeti, 'utriet' prach']
danka_ulohy = [uloha for uloha in janka_ulohy]
print(danka_ulohy)
```

Príkaz vytvoril nový zoznam `danka_ulohy` s rovnakými prvkami, ako mal zoznam `janka_ulohy`.

```
print('Blok 2')
hmotnosti_t = [1.1, 2, 5]
hmotnosti_kg = [hmotnost * 1000 for hmotnost in hmotnosti_t]
print(hmotnosti_kg)
```

Príkaz vytvoril nový zoznam `hmotnosti_kg`, v ktorom sú 1000 násobky prvkov zo zoznamu `hmotnosti_t`.

```
print('Blok 3')
slova = ['aby', 'beh', 'cez', 'les', 'eso', 'dal', 'iba', 'raz']
nove = [slovo for slovo in slova if slovo[0] in 'aeiouy']
print(nove)
```

Príkaz vytvoril nový zoznam `nove`, v ktorom sú tie slová zo zoznamu `slova`, ktoré začínajú samohláskou.

```
print('Blok 4')
cisla = [cislo for cislo in range(20) if cislo % 5 != 0]
print(cisla)
```

Príkaz vytvoril nový zoznam `cisla`, v ktorom sú tie čísla z rozsahu `range(20)`, ktoré nie sú deliteľné 5.

```
print('Blok 5')
pismena = [znak for znak in 'toto je text' if znak not in 'aeiouy']
print(pismena)
```

Príkaz vytvoril nový zoznam `pismena`, v ktorom sú spoluhlásky z reťazca `'toto je text'`.

## VYSVETLENIE (CCA 5 MIN)

V tejto časti by mali žiaci vysvetliť svoje zistenie. Ich vysvetlenie by malo smerovať k tomu, že nové zoznamy je možné vytvárať využitím prvkov iných zoznamov alebo iterovateľných štruktúr (napr. reťazec alebo `range()`). Pre každú hodnotu z takejto štruktúry je možné do nového zoznamu vložiť rovnakú hodnotu alebo hodnotu nejakého výrazu, prípadne podmieniť toto vloženie platnosťou nejakej podmienky.

Odporúčame, aby učiteľ na záver formálne uviedol rôzne tvary generátorovej notácie a zdôraznil, že výsledkom je nový zoznam:

```
[vyraz for hodnota in sekvencia]
```

alebo

```
[vyraz for hodnota in sekvencia if podmienka]
```

kde sekvencia môže byť zoznam, reťazec alebo `range()`.

Nový zoznam vznikne aj prípade `danka_ulohy`. Pozor, nie je to to isté ako `danka_ulohy = danka_ulohy`. V tomto prípade by sa nový zoznam nevytvoril, ale pre pôvodný zoznam sme vytvorili ďalšie pomenovanie.

Neodporúčame, aby učiteľ žiakom ukazoval aj ďalšie formy generátorovej notácie (prechod cez viac sekvencií, viac podmienok, podmienka pre výraz a pod.) V základnom kurze programovania to nie je potrebné.

Odporúčame, aby učiteľ zdôraznil, že generátorová notácia nie je vždy správnu voľbou. Napriek tomu, že je často elegantným a efektívnym riešením, nepoužijeme ju v prípade, ak by zápis bol príliš dlhý alebo zložitý. V týchto prípadoch je lepšie použiť „obyčajný“ cyklus s prípadnou vnorenou podmienkou.

## ROZPRACOVANIE (CCA 16 MIN)

Nasledovné úlohy sú zamerané na precvičenie a pochopenie generátorovej notácie, jej využitie pri riešení problémov.

**Úloha 3** Na jednej z predchádzajúcich hodín ste riešili úlohu, vytvoriť funkciu `generuj_hody()`, ktorá vráti zoznam 100 náhodných hodov kockou. Upravte vytvorenú funkciu tak, aby na vytvorenie zoznamu využila generátorovú notáciu. Pôvodné riešenie je v súbore **kocka.py**.

Porovnajte obidve riešenia.

Riešenie:

```
import random

def generuj_hody():
    return [random.randrange(1, 7) for i in range(100)]
```

Nasledovnú úlohu je možné riešiť viacerými spôsobmi. Predpokladáme, že prvý prístup bude prevládať v žiackych riešeniach.

V druhom riešení sa už vyžaduje úvaha, výsledkom ktorej je, že môžeme použiť prvých 10 čísiel z premiešaného zoznamu. Paradoxne, tento prístup sa najviac približuje k reálnej situácii.

Tretie riešenie uvádzame len pre úplnosť. Využívame v ňom funkciu `sample()` z modulu `random`. Táto funkcia robí to, čo nami požadovaná funkcia. Navyše všeobecne. Ak sa toto riešenie neobjaví medzi žiackymi riešeniami, neodporúčame ním žiakov zaťažovať.

**Úloha 4** Škola každý rok organizuje dobročinný ples. Každý účastník plesu si môže kúpiť lístok do tomboly. Lístky sú očíslované od 100 do 999. Tento rok sa škola rozhodla, že výherné lístky bude zberovať počítač. Vytvorte funkciu `zrebuje()`, ktorá vráti 10 náhodne vybraných čísiel lístkov. Riešenie uložte do súboru **tombola.py**.

Riešte  
podľa  
pokynov  
učiteľa

Riešenie:

Najskôr vygenerujeme zoznam všetkých čísiel lístkov a potom z neho 10 x náhodne jedno vyberieme. Vybrané číslo musíme zo zoznamu vymazať, aby sme ho nemohli náhodou vybrať viackrát.

```
def zrebuje():
    vyhry = []
    listky = [cislo for cislo in range(100, 1000)]
```

```
for i in range(10):
    vyhra = random.choice(listky)
    listky.remove(vyhra)
    vyhry.append(vyhra)
return vyhry
```

Šikovnejšie je riešenie, v ktorom vygenerujeme zoznam všetkých čísiel, potom ho náhodne premiešame a prvých 10 čísiel označíme za výherné.

```
def zrebuj():
    listky = [cislo for cislo in range(100, 1000)]
    random.shuffle(listky)
    vyhry = listky[:10]
    return vyhry
```

Uvádžame aj riešenie, v ktorom využijeme funkciu `sample()` z modulu `random`. V tomto prípade ani nemusíme vytvárať zoznam všetkých čísiel.

```
def zrebuj():
    vyhry = random.sample(range(100, 1000), 10)
    return vyhry
```

**Úloha 5** V pracovnom liste č. 9 ste vytvorili funkciu `je_prvocislo()`. Vytvorte funkciu `prvocisla_z_intervalu()`, ktorá pre zadané krajné hodnoty z číselného intervalu vráti zoznam všetkých prvočísiel z tohto intervalu. Časť riešenia je už pripravená v súbore **prvocisla.py**.

Riešenie:

```
def prvocisla_z_intervalu(od, do):
    prvocisla = [cislo for cislo in range(od, do) if je_prvocislo(cislo)]
    return prvocisla
```

## VYHODNOTENIE (CCA 5 MIN)

Následne požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal a nie klasifikácia známku. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

### Sebahodnotiaci test

1. Doplňte nasledujúci kód tak, aby výstupom bol zoznam [2, 4, 6, 8]

```
zoznam = [1, 2, 3, 4, 5, 6, 8, 9, 9, 9]
vystup = [cislo for cislo in zoznam if cislo % 2 == 0]
print(vystup)
```



2.	<p>Čo je výstupom nasledujúceho programu?</p> <pre>cislo = input('Zadaj prirodzené číslo: ') cislo = int(cislo) vystup = [x ** 2 for x in range(cislo)] print(vystup)</pre> <p>Výstupom je zoznam druhých mocnín čísiel z intervalu &lt;0, zadané číslo)</p>
----	--

V prvej úlohe je potrebné do zoznamu doplniť ľubovoľnú päťicu čísiel, pričom len dve z nich (6 a 8) budú deliteľné číslom 2 a číslo 6 bude niekde pred číslom 8.

## 20 CYKLUS S PODMIENKOU

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>pomocou cyklov,</li> <li>pomocou vetvenia.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>odchytať a generovať výnimky,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus (na prechod zoznamom) a podmienený príkaz pri riešení problémov,</li> <li>získovať a využívať vlastnosti zoznamov pri riešení úloh,</li> <li>používať zoznamy pri riešení problémov.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Jazyk na zápis riešenia:</b> <ul style="list-style-type: none"> <li>používať jazyk na zápis algoritmického riešenia problému,</li> <li>vytvárať zápisy a interpretovať zápisy podľa nových stanovených pravidiel (syntaxe) pre zápis algoritmov.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>vytvárať hypotézu, ako neznámy algoritmus spracováva zadaný vstup, ak sú dané páry vstup–výstup/akcia.</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>rozpoznávať opakujúce sa vzory,</li> <li>rozpoznávať, aká časť algoritmu sa má vykonať pred, počas aj po skončení cyklu,</li> <li>riešiť problémy, v ktorých treba výsledok získať akumulovaním čiastkových výsledkov v rámci cyklu,</li> <li>riešiť problémy, ktoré vyžadujú neznámy počet opakovaní,</li> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia,</li> <li>stanoviť hranice a podmienky vykonávania cyklov.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>riešiť algoritmické problémy pomocou cyklu</li> </ul>	Koncepty informatického myslenia  <b>Logika:</b> <ul style="list-style-type: none"> <li>(LOG4) vyvodzovať (logicky zdôvodňovať) závery z pozorovaní a experimentov (analýza kódov obsahujúcich cyklus while).</li> </ul> <b>Algoritmy:</b> <ul style="list-style-type: none"> <li>(ALG7) vylepšovať existujúce algoritmy (vylepšený Euklidov algoritmus),</li> <li>(ALG8) zapísať algoritmy v konkrétnom formálnom jazyku (Euklidov algoritmus, hádanie čísla).</li> </ul> <b>Vyhodnotenie:</b> <ul style="list-style-type: none"> <li>(VYH3) posúdiť efektívnosť postupu na základe vybraných kritérií (napr. posúdiť efektívnosti algoritmov – časovú náročnosť).</li> </ul>

s podmienkou.

### Riešený didaktický problém

Vo veľa prípadoch, ktoré vyžadujú opakovanie, si pri riešení problémov v jazyku Python vystačíme s pomerne silným príkazom cyklu `for`. „Môžeme“ ho použiť aj v situáciách, keď počet opakovaní nie je vopred známy a nedá sa ani implicitne zistiť. Cyklu `for` môžeme nastaviť nejaký veľmi veľký počet opakovaní a v prípade potreby ho ukončiť príkazom `break`. Tento prístup však nepovažujeme za správny. Po prvé je to sémanticky nesprávne použitie. Po druhé, náš horný odhad počtu opakovaní nemusí byť dostatočný. A po tretie, takéto použitie je ťažko čitateľné (cyklus sa opakuje veľký počet krát a v jeho tele sú ďalšie „riadiace podmienky behu cyklu“ kombinované s príkazmi `break` alebo `continue`). Preto sme do základného kurzu zaradili aj cyklus `while`.

Častou žiackou chybou pri použití cyklu `while` býva nesprávne definovanie podmienky behu cyklu, čo môže spôsobiť predčasné ukončenie cyklu alebo naopak, nekonečný cyklus. Tieto chyby môžu byť ovplyvnené aj skúsenosťou žiakov z prostredia Scratch, v ktorom tiež existuje cyklus s podmienkou behu na začiatku. Telo tohto cyklu sa vykoná ak podmienka behu neplatí. V Pythone musí podmienka behu platiť, aby sa telo cyklu vykonalo.

### Dominantné vyučovacie metódy a formy

- Bádateľská metóda (model 5E),
- individuálna a skupinová forma práce žiakov.

### Príprava učiteľa a pomôcky

Pre učiteľa:

- **ucitel/programovanie\_v\_pythone.pdf**  
metodika vyučovania,
- **ucitel/pracovny\_zosit\_riesene\_ulohy.docx**  
pracovný zošit a riešenia úloh,
- **ucitel/pracovne\_subory\_riesenia/20/**  
riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.

Pre žiaka:

- **ziak/pracovny\_zosit.docx**  
pracovný zošit,
- **ziak/pracovne\_subory/20/**  
pracovné súbory pre žiaka.

Použitie digitálnych nástrojov: NUTNÉ

### Diagnostika splnenia vzdelávacích cieľov

Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.

## Úvod

Toto je 20. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. V tejto metodike sa žiaci naučia používať cyklus s podmienkou (na začiatku) pri riešení problémov.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Predpokladáme, že žiaci si už osvojili prácu s pythonovskými programami a vedia si ich vhodne organizovať. Nie je preto nutné všetky riešenia zapisovať aj do pracovných listov. Primárne by riešenia mali byť ľahko lokalizovateľné žiakom v jeho priečinkoch a vhodne komentované. Odporúčame, aby pre aspoň jednu funkciu žiaci vytvorili dokumentačný reťazec.

Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

V iných programovacích jazykoch môžeme nájsť cyklus s podmienkou behu na začiatku (`while`, `while do`) aj cyklus s podmienkou behu na konci (`repeat until`, `do while`). V Pythone je prístupný len cyklus s podmienkou behu na začiatku, ktorej platnosť umožní vykonanie tela cyklu.. V prípadoch, keď potrebujeme nejakú `postupnost_prikazov` zopakovať aspoň raz, v metodikách preferujeme konštrukciu typu:

```
postupnost_prikazov
while podmienka:
    postupnost_prikazov
```

Príkazy `break` alebo `continue` v tele cyklu v tejto metodike nepoužívame. Primárnym cieľom je, aby žiaci rozpoznali situáciu, kde je potrebné opakovanie, správne identifikovali minimálny počet opakovaní a správne nastavili podmienku pre opakovanie. Rovnako v tejto metodike nepoužívame nekonečný cyklus (`while True:`), aj keď je to štandardná pythonovská konštrukcia. K týmto schémam cyklov sa môžeme venovať v niektorej z nasledujúcich metodík alebo nadstavbových kurzov programovania.

Cyklus `while` vo svojej základnej podobe nie je náročný na pochopenie (navyše, ak žiaci majú skúsenosť s cyklom ako takým a s príkazom vetvenia). Úlohy v tejto metodike sú zamerané na implementáciu alebo nájdanie nie triviálneho (z pohľadu úrovne žiakov) algoritmu.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (5 minút)** – rozhovor so žiakmi o reálnych situáciách, v ktorých je potrebné opakovane vykonávať nejaké činnosti (úloha 1).
- **Skúmanie (7 minút)** – skúmanie nového príkazu (úloha 2).
- **Vysvetlenie (7 minút)** – vysvetlenie zistení z časti Skúmanie.

- **Rozpracovanie (16 minút)** – samostatné programovanie náročnejších úloh (úlohy 3 až 6).
- **Vyhodnotenie (5 minút)** – kontrola žiackych prác, vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 5 MIN)

V predchádzajúcich metodikách sme pri riešení problémov využívali cyklus `for`. Aj keď cyklus `for` je pomerne silný nástroj, doteraz sme ho použili len v situáciách, v ktorých sme vedeli predikovať počet opakovaní (či už priamo (napr. `for i in range(10)`) alebo nepriamo (napr. `for i in zoznam_prvkov`). V situáciách, kde počet opakovaní vopred nevieme predikovať, je jeho použitie problematické.

V úlohe 1 sú formulované situácie, v ktorých sa vyžaduje cyklus, ale počet opakovaní je vopred neznámy. V niektorých situáciách je minimálny počet opakovaní 0, v iných 1.

**Úloha 1** Nižšie sú popísané konkrétne situácie, v ktorých sa môžu opakovane vykonávať nejaké činnosti. Popíšte slovné, ako sa rozhodovať, či dané činnosti opakovať. Aký je najmenší a aký najväčší počet opakovaní?

Situácia:	Koľkokrát je potrebné činnosti opakovať? Koľko najmenej a koľko najviac?
<p>Sestrička u lekára volá na vyšetrenie len objednaných pacientov. Pred sebou má zoznam pacientov a pacientov volá podľa tohto zoznamu.</p> <p>Koľkokrát musí sestrička otvoriť dvere a zavolať pacienta z čakárne na vyšetrenie?</p>	<p>Pokiaľ/kým je na zozname pacient, ktorý nebol vyšetrený, otvorí sestrička dvere a zavola pacienta z čakárne.</p> <p>Najmenej to môže byť 0x (nikto nie je objednaný). Koľko najviac, to nevieme povedať (nepoznáme maximálny počet pacientov na deň, resp. aj keď sestrička pacienta zavola, nemusí byť v čakárni, takže bude volanie ešte raz opakovať).</p>
<p>Marienka si na konci leta zamkla bicykel zámkom s číselným kódom. Na jar si však nevedela na číselnú kombináciu spomenúť. Koľko kombinácií bude musieť skúsiť, aby si bicykel odomkla?</p>	<p>Pokiaľ/kým Marienka neuhádne správnu kombináciu, bude musieť skúšať ďalšie kombinácie.</p> <p>Najmenej to môže byť 1x (ak ihneď na prvýkrát kombináciu uhádne).</p> <p>Koľko najviac, to nevieme povedať. Nebude to však viac, ako je všetkých možných kombinácií (ak nebude opakovane skúšať rovnaké kombinácie).</p>
<p>Mamka upiekla buchty a ponúkla ich Kubkovi. Kubko má buchty rád, ale nerád sa prejedá.</p> <p>Koľkokrát si Kubko zoberie a zje buchtu z misky?</p>	<p>Pokiaľ/kým je Kubko hladný, bude si brať a jesť buchty z misky.</p> <p>Najmenej to môže byť 0x (Kubko už predtým nebol hladný).</p> <p>Koľko najviac, to nevieme povedať. Nebude to však</p>

viac, ako je buchiet na miske.

## SKÚMANIE (CCA 7 MIN)

V nasledujúcej úlohe žiaci analyzujú dva bloky kódu. Obidva obsahujú cyklus `while`. Podstatný rozdiel je v tom, že situácia v prvej ukážke predpokladá vykonanie opakovaného kódu aspoň 1x. V druhej ukážke sa opakovaný kód nemusí vykonať ani raz.

**Úloha 2** Otvorte súbor **opakuj.py**. V súbore sú dva bloky príkazov. Bloky postupne odkomentujte a identifikujte v každom bloku nový, pre vás neznámy príkaz. Spúšťajte uvedený kód a popíšte činnosť neznámeho príkazu. V popise výsledku identifikujte vzťah medzi vaším vstupom a výsledkom bloku príkazov.

```
print('Blok 1')
heslo = input('Zadaj heslo: ')
while heslo != 'abrakadabra':
    heslo = input('Zadaj heslo: ')
print('Výborne!')
```

Program si od nás pýta heslo. Heslo musíme zadávať dovtedy, kým ho neuhádneme. Minimálne ho musíme zadať 1x.

Príkaz `while` opakuje výzvu na zadanie hesla. Opakovanie výzvy skončí, ak podmienka behu `heslo != 'abrakadabra'` prestane platiť – heslo sme uhádli. **Heslo musíme zadať aspoň raz.** O toto sa postará výzva pred príkazom `while`.

```
print('Blok 2')
pocet = input('Koľko prvočísiel potrebujete? ')
pocet = int(pocet)
prvocisla = []
cislo = 1
while len(prvocisla) < pocet:
    if je_prvocislo(cislo):
        prvocisla.append(cislo)
        cislo = cislo + 1
print(prvocisla)
```

Program si od nás vypýta počet prvočísiel, ktorý požadujeme. Program postupne testuje čísla, či sú prvočíslami. Ak nájde požadovaný počet, hľadanie prvočísiel skončí.

Príkaz `while` opakuje testovanie čísiel. Po otestovaní čísla, postúpi na testovanie nasledujúceho čísla (zvyší hodnotu testovaného čísla o 1). Testovanie prebieha dovtedy, kým počet nájdených prvočísiel je menší ako počet požadovaných. **Ak nepotrebujeme žiadne prvočíslo, nerealizuje sa ani jeden test.**

## VYSVETLENIE (CCA 7 MIN)

**Úloha 3** Vráťte sa k predchádzajúcej ukážke kódu. Čím je zabezpečené, že príkazy sa nebudú opakovať do nekonečna?

Riešenie:

Cyklus `while` v tele opakuje príkazy, ktorých vykonávanie má vplyv na platnosť podmienky behu (načítaný

vstup môže byť niekedy rovný reťazcu `abracadabra`, počet prvočísel stále pribúda, takže raz dosiahne požadovaný počet). Tým je zabezpečené, že podmienka behu cyklu raz prestane platiť.

V tejto časti by mali žiaci vysvetliť svoje zistenie. Ich vysvetlenie by malo smerovať k tomu, že v situáciách, kedy potrebujeme opakovane vykonávať nejakú postupnosť príkazov a presný počet nedokážeme určiť, môžeme použiť príkaz cyklu `while` s podmienkou behu. Podmienka je logický výraz. Pokiaľ je podmienka platná, príkazy v tele cyklu sa budú opakovať. Aby nedošlo k nekonečnému opakovaniu, musia mať príkazy v tele cyklu vplyv na platnosť podmienky behu cyklu.

Odporúčame, aby učiteľ na záver formálne uviedol nasledujúce schémy:

```
while podmienka:
    postupnost_prikazov
```

Ak sa `postupnost_prikazov` nemusí vykonať **ani raz**.

```
postupnost_prikazov
while podmienka:
    postupnost_prikazov
```

Ak sa `postupnost_prikazov` musí vykonať **aspoň raz**.

## ROZPRACOVANIE (CCA 16 MIN)

V nasledujúcej úlohe by mali žiaci implementovať Euklidov algoritmus pre nájdenie najväčšieho spoločného deliteľa dvoch prirodzených čísel. Je možné, že sa s týmto algoritmom ešte nestretli a algoritmus je navyše zapísaný matematickou notáciou. Aby žiaci ľahšie „objavili“ algoritmus z programátorského pohľadu, zaradili sme časť úlohy a). Pri riešení tejto časti by žiaci mali algoritmus realizovať priamo na konkrétnom prípade. Tento prístup je jednou zo stratégií riešenia problémov – vyrieš konkrétny prípad.

**Úloha 4** V matematike pri úprave zlomkov na základný tvar hľadáme spoločného deliteľa čitateľa aj menovateľa. Jedným z postupov ako nájsť spoločného deliteľa, dokonca toho najväčšieho, je Euklidov algoritmus. V matematike ho zapíšeme nasledovne:

podľa  $NSD(x, y) = x, \Leftrightarrow x = y$

pokynov  $NSD(x - y, y) \Leftrightarrow x > y$

učiteľa.  $NSD(x, y - x) \Leftrightarrow x < y$

- Zvoľte si dve prirodzené čísla  $a$  a  $o$  verte, či podľa vyššie uvedeného postupu nájdete ich najväčšieho spoločného deliteľa. Ako by ste slovami popísali uvedený algoritmus.
- Vytvorte funkciu `nsd()`, ktorá pre dve zadané prirodzené čísla vráti ich najväčšieho spoločného deliteľa.
- Upravte funkciu `nsd()` tak, aby v prípade, že zadané čísla nie sú prirodzené, funkcia vyhodila výnimku.

Riešenie uložte do súboru **euklides.py**.

Riešenie a):

Zvoľme napr. čísla  $x = 18$  a  $y = 24$ . Podľa algoritmu platí:  $NSD(18, 24) = NSD(18, 6) = NSD(12, 6) = NSD(6, 6) = 6$ . Číslo 6 je najväčším spoločným deliteľom čísel 18 a 24. Tento algoritmus môžeme slovne popísať napr. takto: Kým sú čísla rôzne, od väčšieho odpočítavaj menšie. Ak sa rovnajú, ich najväčším spoločným deliteľom je jedno z nich.

Riešenie b):

```
def nsd(x, y):  
    while x != y:  
        if x > y:  
            x = x - y  
        else:  
            y = y - x  
    return x
```

Riešenie c):

```
def je_prirodzene(cislo):  
    if cislo != int(cislo):  
        return False  
    if cislo <= 0:  
        return False  
    return True  
  
def nsd(x, y): # Riešenie C  
    if not (je_prirodzene(x) and je_prirodzene(y)):  
        raise ValueError('Zadané čísla nie sú prirodzené!')  
    while x != y:  
        if x > y:  
            x = x - y  
        else:  
            y = y - x  
    return x
```

V riešení c) potrebujeme otestovať, či sú obidve čísla prirodzené. Aby sme nepísali 2x rovnaký kód, je výhodnejšie vytvoriť si na testovanie funkciu.

Nasledujúca úloha je zameraná na vylepšenie riešenia. Žiaci by mali identifikovať „slabé“ miesto uvedeného algoritmu, navrhnúť a implementovať efektívnejšiu verziu algoritmu.

Žiaci by mali porovnať čas behu oboch verzií algoritmu (obidve funkcie). Tu sa neočakáva nejaké presné meranie. Ak si žiaci zvolia dostatočne veľký rozdiel čísel (viď. napr. dvojica uvedená v riešení), rozdiel v čase realizácie oboch funkcií bude zjavný. Samozrejme záleží to od výkonnosti počítača. Odporúčame, aby učiteľ žiakov pri výbere testovacej dvojice usmernil.

**Úloha 5** Preskúmajte Euklidov algoritmus a identifikujte situácie, v ktorých tento postup nie je veľmi efektívny. Vedeli by ste sa k výsledku, ktorý dosiahnete postupným odpočítavaním menšieho čísla od väčšieho, dopracovať aj rýchlejšie? Ak áno, implementujte efektívnejší postup do funkcie `nsd_vylepsene()`. Riešenie uložte do súboru **euklides.py**.

Riešte podľa pokynov učiteľa Porovnajte čas výpočtu pôvodným a vylepšeným spôsobom. Spustite každú z funkcií pre rovnakú dvojicu čísel a porovnajte čas behu.

Riešenie:

Algoritmus nie je veľmi efektívny, ak je rozdiel čísel  $x$  a  $y$  príliš veľký, napr. `NSD(1000000000, 6)`. Postupným odpočítavaním menšieho čísla od väčšieho získame zvyšok po delení väčšieho čísla menším.



$$1000000000 - 6 - 6 - 6 \dots - 6 = 4 = 1000000000 \% 6$$

Postupné odpočítavanie môžeme nahradiť zvyškom po delení.

```
def nsd_vylepsene(x, y):
    while x != 0 != y:
        if x > y:
            x = x % y
        else:
            y = y % x
    return x + y
```

V pôvodnom algoritme sme výpočet ukončili, ak sa čísla  $x$  a  $y$  rovnali. V prípade vylepšeného algoritmu to bude inak. Pre dve čísla, z ktorých je jedno násobkom druhého (napr. 24, 6), sa väčšie z nich nahradí číslom 0 ( $24 \% 6 = 0$ ). Výpočet budeme opakuvať, pokiaľ budú čísla rôzne od 0. Ak jedno z čísiel bude 0, druhé číslo (alebo ich súčet) vyhlásime za najväčšieho spoločného deliteľa.

V nasledujúcej úlohe žiaci simulujú hru „Hádaj číslo“. Ak žiaci uvedenú hru nepoznajú, odporúčame, aby si ju vo dvojiciach zahrli. Žiak, ktorý číslo vymýšľa, by si mal uvedomiť, aké kroky počas realizácie hry vykonáva.

Časť b) nasledujúcej úlohy je propedeutikou k binárnemu vyhľadávaniu hodnoty v usporiadanej postupnosti hodnôt. Táto časť úlohy je zameraná aj na kritické myslenie. Je možné, že žiaci budú hádať myslené číslo opakovane na menej ako 5 pokusov. Žiaci by mali argumentovať, prečo nimi preferovaný maximálny počet hádaní je postačujúci, resp. nachádzať kontra príklady, kedy uvedený počet nie je dostačujúci.

**Úloha 6** Hru „Hádaj číslo!“ asi pozná každý z nás. Jeden z dvojice si vymyslí číslo a druhý sa ho pokúša na čo najmenší počet pokusov uhádnuť. Hádajúci pri hádaní dostáva odpovede „viac“, „menej“ alebo „uhádol si“.

Časť b)  
riešte  
podľa  
pokynov  
učiteľa

- Vytvorte program **hadaj\_cislo.py**, kde úlohu toho, kto si vymyslí číslo, bude simulovať počítač. Používateľ bude ten, ktorý sa bude pokúšať, počítačom myslené číslo uhádnuť. Poznámka: Je rozumné určiť interval pre vymyslené číslo.
- Upravte program tak, aby vymyslené číslo bolo z intervalu 1..16 a hádajúci mal maximálne 4 pokusy. Podarí sa vám na maximálne 4 pokusy číslo vždy uhádnuť? Zdôvodnite!

Riešenie a):

```
import random

myslene_cislo = random.randrange(1, 17)
print('Myslím si číslo od 1 do 16. Uhádneš ho?')
cislo = input('Zadaj svoj tip: ')
cislo = int(cislo)
while cislo != myslene_cislo:
    if cislo > myslene_cislo:
        print('menej')
    else:
        print('viac')
    cislo = input('Zadaj svoj tip: ')
    cislo = int(cislo)
print(f'Uhádol si. Myslel som si číslo {myslene_cislo}.')
```

Riešenie b):

```
import random

myslene_cislo = random.randrange(1, 17)
print('Myslím si číslo od 1 do 16. Uhádneš ho?')
cislo = input('Zadaj svoj tip: ')
cislo = int(cislo)
pocet_pokusov = 1
while cislo != myslene_cislo and pocet_pokusov < 4:
    if cislo > myslene_cislo:
        print('menej')
    else:
        print('viac')
    cislo = input('Zadaj svoj tip: ')
    cislo = int(cislo)
    pocet_pokusov = pocet_pokusov + 1
if cislo == myslene_cislo:
    print(f'Uhádol si. Myslel som si číslo {myslene_cislo}.')
else:
    print(f'Neuhádol si. Myslel som si číslo {myslene_cislo}.')
```

Pri hádaní čísla by sme mali zvoliť nejakú premyslenú stratégiu. Hádať čísla v poradí 1, 2, 3 atď. sa síce dá, ale nie je to najšikovnejší postup. Po každom hádaní vylúčime len jedno číslo.

Použijeme stratégiu delenie intervalu na polovicu. Nájdime číslo v strede intervalu a tipujme toto číslo. Ak číslo v strede nie je hľadaným číslom, vylúčime aspoň polovicu čísiel z intervalu (pri prvom hádaní 8 čísiel). V ďalšom kole tipovania uvažujeme už iba nový interval. Takto postupne pokračujeme až kým číslo neuhádneme alebo nám neostane interval len s jedným číslom, ktoré musí byť to myslené.

Aj keď je tento postup šikovnejší ako ten predchádzajúci, 4 pokusy nám vždy stačiť nebudú. Na začiatku máme 16 čísiel.

Po prvom hádaní nám ostane najviac 8 čísiel.

Po druhom hádaní nám ostanú najviac 4 čísla.

Po treťom hádaní nám ostanú najviac 2 čísla.

Po štvrtom hádaní nám ostane najviac 1 číslo. Vieme s istotou, aké je myslené číslo, ale 4 pokusy sme si už vyčerpali.

V niektorých prípadoch sa nám to podarí aj na menej pokusov. V najhoršom prípade však potrebujeme aspoň 5 pokusov.

## VYHODNOTENIE (CCA 5 MIN)

Následne požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal, a nie klasifikácia známku. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

*Sebahodnotiaci test*

1.	<p>Čo bude výstupom nasledujúceho programu? Vyberte jednu z uvedených možností a) až f).</p> <pre>odpocet = 3  while odpocet &gt; 1:     print(odpocet)     odpocet = odpocet - 1 print('Štart')</pre> <table><tr><td>a)</td><td>b)</td><td>c)</td><td>d)</td><td>e)</td><td>f)</td></tr><tr><td>3</td><td><b>3</b></td><td>3</td><td>3</td><td>3</td><td>3</td></tr><tr><td>2</td><td><b>2</b></td><td>Štart</td><td>2</td><td>2</td><td>Štart</td></tr><tr><td></td><td><b>Štart</b></td><td>2</td><td>1</td><td>1</td><td>2</td></tr><tr><td></td><td></td><td>Štart</td><td></td><td>Štart</td><td>Štart</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>1</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>Štart</td></tr></table>	a)	b)	c)	d)	e)	f)	3	<b>3</b>	3	3	3	3	2	<b>2</b>	Štart	2	2	Štart		<b>Štart</b>	2	1	1	2			Štart		Štart	Štart						1						Štart
a)	b)	c)	d)	e)	f)																																						
3	<b>3</b>	3	3	3	3																																						
2	<b>2</b>	Štart	2	2	Štart																																						
	<b>Štart</b>	2	1	1	2																																						
		Štart		Štart	Štart																																						
					1																																						
					Štart																																						
2.	<p>Doplňte prázdne miesta v nasledujúcom programe tak, aby výstupom bolo 5 písmen X.</p> <pre>pocitadlo = <input type="text" value="2"/> print('X') print('X') while pocitadlo != 5:     print('X')     pocitadlo = pocitadlo + <input type="text" value="1"/></pre>																																										

V prvej úlohe sledujeme pochopenie konštrukcie cyklu `while`. Žiaci by si mali uvedomiť, ako/kedy sa vyhodnocuje podmienka behu cyklu a ktoré príkazy do cyklu patria a ktoré už nie.

V druhej úlohe očakávame, že si žiaci uvedomia, ktoré príkazy sa vykonávajú pred cyklom a ktoré v cykle. Uvedené hodnoty riešenia nie sú jediné možné. Dedukciou zistíme, že posledná hodnota počítadla musí byť 5 a pre túto hodnotu sa cyklus už nevykoná. Cyklus sa musí zopakovať 3x. Stačí teda zvolenú pripočítavanú hodnotu 3x odpočítať od 5 a dostaneme štartovaciu hodnotu.

## 21 VNORENÉ RIADIACE ŠTRUKTÚRY

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>pomocou cyklov,</li> <li>pomocou vetvenia.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>odchytať a generovať výnimky,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus a podmienený príkaz pri riešení problémov,</li> <li>používať zoznamy a reťazce pri riešení problémov.</li> </ul>	
<b>Ciele</b>	
Žiakom osvojované vedomosti a zručnosti	Žiakom rozvíjané spôsobilosti
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia,</li> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>rozpoznávať opakujúce sa vzory,</li> <li>rozpoznávať, aká časť algoritmu sa má vykonať pred, počas aj po skončení cyklu,</li> <li>riešiť problémy, v ktorých treba výsledok získať akumulovaním čiastkových výsledkov v rámci cyklu,</li> <li>riešiť problémy, ktoré vyžadujú neznámy počet opakovaní,</li> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia,</li> <li>stanoviť hranice a podmienky vykonávania cyklov.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie a podmienky, kedy treba použiť vetvenie,</li> <li>rozpoznávať, aká časť algoritmu sa má vykonať pred, v rámci a po skončení vetvenia,</li> <li>riešiť problémy, ktoré vyžadujú vetvenie so zloženými podmienkami (s logickými spojkami),</li> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> Programovací jazyk <b>Python</b> :	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG5) logicky zdôvodniť rozdelenie algoritmu na menšie časti (vytváranie podmienok pre viaceré vetvenia v riešení úloh).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (algoritmy riešiace uvedené problémy),</li> <li>(ALG4) vytvárať vlastné algoritmy, ktoré pracujú s množinou dát (analýza dát v zoznamoch, porovnávanie textov).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK2) hierarchická dekompozícia – hierarchicky rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (vytvoriť diagram, zobrazíť hierarchiu aj podproblémy rozdeliť na menšie podproblémy najmä v časti analýzy riešených problémov).</li> </ul> Vyhodnotenie: <ul style="list-style-type: none"> <li>(VYH3) posúdiť kvalitu/správnosť objektu na základe definovaných kritérií (napr. vyhodnotiť korektnosť záznamu z pretekov, kvalitu prepisu textu).</li> </ul>

- riešiť algoritmické problémy pomocou cyklov, podmienených príkazov, generovania výnimiek, odchyťovania výnimiek a i ich kombinácií.

### **Riešený didaktický problém**

Pri implementácii riešenia problému často vytvoríme kód, v ktorom sú vzájomne, vo viacerých úrovniach, vnorené cykly a podmienené príkazy. Takýto kód môže byť ťažšie čitateľný, a preto sa týmto situáciám v metodikách snažíme vyhnúť (napr. tak, že vnorenú časť presunieme do samostatnej funkcie). V niektorých prípadoch sa tento prístup nedá použiť, lebo by sme vytvorili veľmi veľa malých funkcií, čím by sme čitateľnosť kódu naopak zhoršili. V tejto metodike sa zameriame práve na tieto prípady. Aby sme v riešení (vo výslednom kóde) redukovali množstvo chýb, kladieme dôraz na dôslednú analýzu problému. Vďaka nej sa výsledné riešenie často zjednoduší a sprehladní.

### **Dominantné vyučovacie metódy a formy**

- Problémová metóda,
- Frontálna a individuálna forma.

### **Príprava učiteľa a pomôcky**

Pre učiteľa:

- **ucitel/programovanie\_v\_pythone.pdf**  
metodika vyučovania,
- **ucitel/pracovny\_zosit\_riesene\_ulohy.docx**  
pracovný zošit a riešenia úloh,
- **ucitel/pracovne\_subory\_riesenia/21/**  
riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu,

Pre žiaka:

- **ziak/pracovny\_zosit.docx**  
pracovný zošit,
- **ziak/pracovne\_subory/21/**  
pracovný priestor pre žiaka.

Použitie digitálnych nástrojov: NUTNÉ

### **Diagnostika splnenia vzdelávacích cieľov**

Spätná väzba získavaná počas riešenia úloh, v závere formou diskusie.

## Úvod

V tejto metodike sa zameriavame na riešenie úloh, v ktorých sa využívajú vnorené riadiace štruktúry (podmienené príkazy a cykly). Keďže zostavenie takýchto kombinácií štruktúr nie je jednoduché, žiaci by mali venovať dostatok pozornosti analýze úloh a až na základe výsledkov analýzy navrhovať riešenie. Pri analýze môžu žiaci využívať tabuľky, nákresy, schémy, prípadne vyriešiť niekoľko konkrétnych prípadov. Pri vybraných úlohách uvádzame niektoré z možných analýz, prípadne ukážky riešení, ktoré sú dôsledkom nedostatočnej analýzy problému. V tejto metodike môžu žiaci vytvárať rôzne riešenia tej istej úlohy, pričom každé z nich môže byť správne.

Žiaci by mali vedieť používať dátové štruktúry (reťazec a zoznam) a riadiace štruktúry (cykly, podmienený príkaz).

Táto metodika nie je založená na bádateľskej metóde, jej cieľom je systematizácia a hlbšie precvičenie vedomostí a zručností žiakov. Je preto vhodné, ak na predchádzajúcej hodine žiakov požiadame, aby si opätovne pozorne prečítali Vedomosti v kocke z predchádzajúcich pracovných listov.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh a miesto primárne určené na analýzu úloh.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny:

- **Úvod (5 minút)** – krátka diskusia o riadiacich štruktúrach.
- **Precvičovanie, samostatná práca (30 minút)** – riešenie úloh z pracovného listu (úlohy 1 až 6).
- **Zhrnutie (5 minút)** – diskusia.

### ÚVOD (CCA 5 MIN)

Hodinu začneme krátkou diskusiou o riadiacich štruktúrach, s ktorými sa žiaci doteraz pri programovaní stretli.

**Úloha 1** Zopakujme si, ktoré riadiace štruktúry už poznáme.

Riešenie:

cyklus for	príkaz if	cyklus while	výnimky
<pre>for i in range(10):     prikazy</pre>	<pre>if podmienka:     prikazy</pre>	<pre>while podmienka:     prikazy</pre>	<pre>try:     prikaz except chyba1:     prikaz 1 except chyba2:     prikaz2 ...</pre>
<pre>for i in text:</pre>	<pre>if podmienka:     prikazy1</pre>		

prikazy	else: prikazy2	else: prikazn
for i in zoznam: prikazy	if podmienka1: prikazy1 elif podmienka2: prikazy2 elif podmienka3: prikazy2 .... else: prikazy	raise chyba

## PRECVIČOVANIE, SAMOSTATNÁ PRÁCA (CCA 30 MIN)

Úlohy 2 a 3 sú zamerané na prepis podmienok, ktoré sú definované v zadaní. V úlohe 2 sú podmienky prehľadne vyjadrené v tabuľke. V úlohe 3 je potrebné podmienky extrahovať z textu.

**Úloha 2** Na čerpacej stanici majú uvedený nasledovný systém zliav:

odber v litroch	ZĽAVA	
	zákaznícka karta – NIE	zákaznícka karta – ÁNO
< 20	0 %	1 %
< 40	1 %	2 %
>= 40	2 %	4 %

Vytvorte funkciu `platba()`, ktorá pre zadaný objem natankovaného paliva, jednotkovú cenu paliva a informáciu, či zákazník má zákaznícku kartu vráti sumu, ktorú má zákazník zaplatiť. Riešenie uložte do súboru **cerpacia\_stanica.py**

Riešenie:

```
def platba(objem_paliva, zakaznicka_karta, cena_l):
    if objem_paliva < 20:
        if zakaznicka_karta:
            zlava = 0.01
        else:
            zlava = 0
    elif objem_paliva < 40:
        if zakaznicka_karta:
            zlava = 0.02
        else:
            zlava = 0.01
    else:
        if zakaznicka_karta:
            zlava = 0.04
        else:
            zlava = 0.02
    cena = objem_paliva * cena_l * (1 - zlava)
    return cena
```

Pri riešení predchádzajúcej úlohy môžu mať žiaci tendenciu vetviť riešenie problému nasledovným spôsobom:

```
def platba(objem_paliva, zakaznicka_karta, cena_l):
    if objem_paliva < 20 and zakaznicka_karta:
        zlava = 0.01
    elif objem_paliva < 20 and not zakaznicka_karta:
        zlava = 0
    elif objem_paliva < 40 and zakaznicka_karta:
        zlava = 0.02
    elif objem_paliva < 40 and not zakaznicka_karta:
        zlava = 0.01
    elif objem_paliva >= 40 and zakaznicka_karta:
        zlava = 0.04
    elif objem_paliva >= 40 and not zakaznicka_karta:
        zlava = 0.02
    cena = objem_paliva * cena_l * (1 - zlava)
    return cena
```

alebo ešte menej efektívnym spôsobom:

```
def platba(objem_paliva, zakaznicka_karta, cena_l):
    if objem_paliva < 20 and zakaznicka_karta:
        zlava = 0.01
    if objem_paliva < 20 and not zakaznicka_karta:
        zlava = 0
    if 20 <= objem_paliva < 40 and zakaznicka_karta:
        zlava = 0.02
    if 20 <= objem_paliva < 40 and not zakaznicka_karta:
        zlava = 0.01
    if objem_paliva >= 40 and zakaznicka_karta:
        zlava = 0.04
    if objem_paliva >= 40 and not zakaznicka_karta:
        zlava = 0.02
    cena = objem_paliva * cena_l * (1 - zlava)
    return cena
```

Aj keď sú obidve riešenia správne, tento prístup neodporúčame. V riešeniach opakovane testujeme rovnaké podmienky. V prípade, ak by sme takýto prístup použili v riešení problému, kde sa vyhodnocuje viac jednoduchých podmienok, bude výsledný zápis značne neprehľadný.

**Úloha 3** Obeh Zeme okolo Slnka netrvá celý počet dní. Presne je to 365 dní, 5 hodín, 48 minút a 45,4 sekundy. Aby sme kalendárny rok zosúladiť s obehom Zeme okolo Slnka, musíme jeho dĺžku pravidelne korigovať. Táto korekcia spočíva v zavedení priestupných rokov, ktoré majú o jeden deň (29. február) viac, ako ostatné roky. Priestupné sú tie roky, ktoré sú deliteľné 4. Ak je rok zároveň deliteľný 100, tak priestupný nie je. Ak by však bol deliteľný aj 400, priestupný bude. Vytvorte funkciu `je_priestupny_rok()`, ktorá pre zadaný rok zistí a vráti, či je priestupný alebo nie. Riešenie uložte do súboru **priestupne\_roky.py**.

Riešenie:

```
def je_priestupny_rok(rok):
    if rok % 4 == 0:
        if rok % 100 == 0:
            if rok % 400 == 0:
                return True
            else:
```



```

        return False
    else:
        return True
    else:
        return False

```

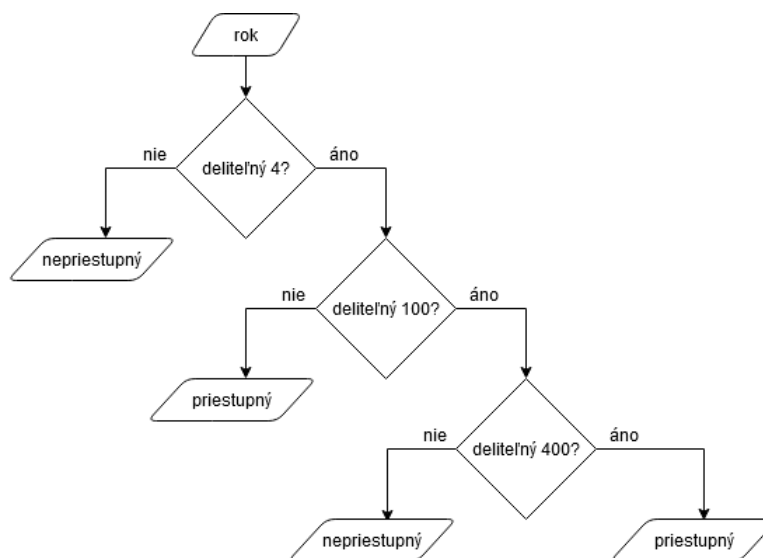
V riešení úlohy môžu mať žiaci tendenciu zostaviť podmienku pre priestupný rok „priamym“ prepísaním slovného vyjadrenia podmienky zo zadanie úlohy:

```

def je_priestupny_rok1(rok):
    if rok % 4 == 0 and rok % 100 == 0 and rok % 400 == 0 or rok % 4 == 0 and rok % 100 != 0:
        return True
    else:
        return False

```

Aj keď je vyššie uvedené riešenie správne, prečítanie a pochopenie takto zapísanej podmienky nie je jednoduché. Odporúčame, aby učiteľ smeroval žiakov k hlbšej analýze problému, na základe ktorej je možné podmienky zostaviť efektívnejšie. Pri analýze môžeme využiť jednoduchý diagram, napr.:



V nasledujúcich úlohách odporúčame, aby žiaci problémy vopred analyzovali a až na základe analýzy zostavili program.

- Úloha 4** V súťaži v strojopise sa porovnáva originálny text s jeho prepisom. Ak súťažiaci navzájom vymenil dva po sebe idúce znaky, dostane jeden mínusový bod. Ak súťažiaci namiesto originálneho znaku napísal nejaký iný znak, dostáva dva mínusové body.
- Časť b) riešte podľa pokynov učiteľa.
- Vytvorte funkciu `vyhodnot()`, ktorá porovná originálny text s jeho prepisom a vráti počet mínusových bodov. Ak porovnávané texty nemajú rovnakú dĺžku, nech funkcia vyhodí zmysluplnú výnimku. Riešenie uložte do súboru **strojopis.py**.
  - Upravte funkciu `vyhodnot()` tak, aby dokázala porovnať aj rôzne dlhé texty. Texty porovnávajte od začiatku a chýbajúce znaky považujte za chybné prepísané znaky.

Riešenie a):

```
def vyhodnot(original, prepis): # Riešenie A
    if len(original) != len(prepis):
        raise ValueError('Porovnávané texty majú rôzne dĺžky')
    pocet_bodov = 0
    idx = 0
    while idx < len(original):
        if original[idx] != prepis[idx]:
            if idx + 1 < len(original) and \
                original[idx] == prepis[idx + 1] and \
                original[idx + 1] == prepis[idx]:
                pocet_bodov = pocet_bodov + 1
                idx = idx + 2
            else:
                pocet_bodov = pocet_bodov + 2
                idx = idx + 1
        else:
            idx = idx + 1
    return pocet_bodov
```

Riešenie b):

```
def vyhodnot(original, prepis): # Riešenie B
    pocet_bodov = 0
    idx = 0
    porovnavana_dlzka = min(len(original), len(prepis))
    while idx < porovnavana_dlzka:
        if original[idx] != prepis[idx]:
            if idx + 1 < len(original) and \
                original[idx] == prepis[idx + 1] and \
                original[idx + 1] == prepis[idx]:
                pocet_bodov = pocet_bodov + 1
                idx = idx + 2
            else:
                pocet_bodov = pocet_bodov + 2
                idx = idx + 1
        else:
            idx = idx + 1
    rozdiel_dlzok = abs(len(original) - len(prepis))
    pocet_bodov = pocet_bodov + 2 * rozdiel_dlzok
    return pocet_bodov
```

V riešení primárne identifikujeme situácie, v ktorých sú znaky na rovnakých miestach v origináli a v prepise rôzne. Najskôr zisťujeme zmenu poradia dvoch znakov. Toto však nemôže nastať v prípade, ak aktuálny znak je posledným znakom textu. Túto situáciu preto vylúčime. Ak nenastala zmena poradia, ide o chybný prepis daného znaku.

V riešení b) porovnáваме len začiatky časti textov, ktoré sú rovnako dlhé. Nepredpokladáme, že prepísaný text je kratší alebo dlhší. Chýbajúce alebo nadbytočné znaky – rozdiel dĺžok textov vyhodnotíme ako chybné prepísané znaky.

**Úloha 5** V laboratóriu skúmajú zaujímavý kmeň buniek. Zistili, že ak ich umiestnia do kruhu, populácia buniek sa každú hodinu zmení podľa nasledovných pravidiel:

- ak bunka má dvoch susedov, do ďalšej generácie neprežije a umiera na nedostatok potravy,
- ak bunka má jedného suseda, do ďalšej generácie bunka prežije,
- ak bunka nemá žiadneho suseda, do ďalšej generácie neprežije a umiera na osamotenie,
- ak prázdne políčko má dvoch susedov, v ďalšej generácii sa na tomto mieste objaví živá bunka,
- v ostatných prípadoch sa situácia na danom mieste nezmení.

- Vytvorte funkciu `dalsia_generacia()`, ktorá zmení zadanú generáciu buniek na ďalšiu generáciu.
- Vytvorte funkciu `zobraz_generacie()`, ktorá pre zadaný počet hodín a počiatočnú generáciu buniek postupne zobrazí, ako sa populácia buniek vyvíjala.

Pomôcka: Premyslite si, ako vhodne reprezentovať stav populácie buniek.

Riešenie uložte do súboru **zivot.py**.

Riešenie:

```
def pocet_susedov(idx, generacia):
    dlzka = len(generacia)
    pocet = generacia[(idx - 1) % dlzka] + generacia[(idx + 1) % dlzka]
    return pocet

def dalsia_generacia(generacia: list): # Riešenie A
    nova_generacia = [0] * len(generacia)
    for i in range(len(generacia)):
        if generacia[i] == 1: # bunka žije
            if pocet_susedov(i, generacia) == 1:
                nova_generacia[i] = 1
        else: # bunka nežije
            if pocet_susedov(i, generacia) == 2:
                nova_generacia[i] = 1
    generacia.clear() # záznam pôvodnej generácie zmažeme
    generacia.extend(nova_generacia) # aktualizujeme generáciu na novú

def zobraz_generacie(hodiny, generacia): # Riešenie B
    print(generacia)
    for i in range(hodiny):
        dalsia_generacia(generacia)
        print(generacia)
```

Pri riešení predchádzajúcej úlohy musíme:

- 1 navrhnuť vhodné kódovanie buniek v populácii,
- 2 implementovať pravidlá pre vytvorenie ďalšej generácie.

Generácia buniek žije v cyklickej štruktúre. Takúto štruktúru môžeme implementovať pomocou zoznamu tak, že dopyt za posledný prvok presmerujeme na prvý a naopak, dopyt pred prvý prvok presmerujeme za posledný. Využijeme operáciu zvyšok po delení. Živé bunky môžeme kódovať hodnotou `1` a prázdne miesta hodnotou `0`.

Dôsledná analýza úlohy na začiatku nám následne výrazní uľahčí jej riešenie. Pri implementácii pravidiel odporúčame, aby si žiaci znázornili všetky situácie, ktoré môžu nastať. Napr.: (sledovaná bunka je zvýraznená červeným obrysom)

stará generácia	→	nová generácia	stav
○○○	→	○○○	bez zmeny
○●○	→	○●○	bez zmeny
○●○	→	○○○	zomiera
○●●	→	○●●	prežíva
●○○	→	●○○	bez zmeny
●○●	→	●●●	narodí sa
●●○	→	●●○	prežíva
●●●	→	●○●	zomiera

Ak si všimneme stav sledovanej pozície v novej generácii, vidíme, že len v troch prípadoch sa na sledovanej pozícii objaví živá bunka. Novú generáciu preto môžeme vytvoriť na začiatku prázdnu a len v týchto troch prípadoch na určené pozície umiestniť živé bunky. Počet situácií, ktoré by sme potrebovali otestovať sa týmto značne zníži.

Posledná vec na ktorú musíme dať pozor je, že novú generáciu musíme generovať do nového zoznamu. Inak by sme si priebežne prepisovali pôvodnú generáciu. Časť zoznamu by bola ešte reprezentáciou starej generácie a časť už reprezentáciou novej generácie. Takto by sme nevedeli vyhodnotiť počet susedov konkrétnej bunky v starej generácii.

Pre zistenie počtu susedov sme vytvorili samostatnú funkciu `pocet_susedov()`. Vďaka tomu sa funkcia `dalsia_generacia()` sprehľadnila.

## ZHRNUTIE (CCA 5 MINÚT)

V tejto časti stručne zopakujeme obsah vyučovania, postupy a metódy, ktoré sme využili pri riešení úloh. Zdôraznime žiakom potrebu analýzy úloh ešte pred samotnou implementáciou riešenia. Odporúčame, aby si žiaci do pracovného listu k úlohe 1 doplnili informácie, ktoré na začiatku hodiny nepostrehli.

## 22 OPAKOVANIE IV. + DIDAKTICKÝ TEST

Tematický celok / Téma	Stupeň školy / Odporúčaný ročník / Rozsah
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>pomocou cyklov,</li> <li>pomocou vetvenia.</li> </ul>	SŠ / 2. ročník / 2 vyučovacie hodiny
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>odchytať a generovať výnimky,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus a podmienený príkaz pri riešení problémov,</li> <li>používať zoznamy a reťazce pri riešení problémov.</li> </ul>	
<b>Ciele</b>	
Žiakom osvojované vedomosti a zručnosti	Žiakom rozvíjané spôsobilosti
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia,</li> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Pomocou cyklov:</b> <ul style="list-style-type: none"> <li>rozpoznávať opakujúce sa vzory,</li> <li>rozpoznávať, aká časť algoritmu sa má vykonať pred, počas aj po skončení cyklu,</li> <li>riešiť problémy, v ktorých treba výsledok získať akumulovaním čiastkových výsledkov v rámci cyklu,</li> <li>riešiť problémy, ktoré vyžadujú neznámy počet opakovaní,</li> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia,</li> <li>stanoviť hranice a podmienky vykonávania cyklov.</li> </ul> <b>Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie a podmienky, kedy treba použiť vetvenie,</li> <li>rozpoznávať, aká časť algoritmu sa má vykonať pred, v rámci a po skončení vetvenia,</li> <li>riešiť problémy, ktoré vyžadujú vetvenie so zloženými podmienkami (s logickými spojkami),</li> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> Programovací jazyk <b>Python</b> :	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG5) logicky zdôvodniť rozdelenie algoritmu na menšie časti (vytváranie podmienok pre viaceré vetvenia v riešení úloh).</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (algoritmy riešiace uvedené problémy),</li> <li>(ALG4) vytvárať vlastné algoritmy, ktoré pracujú s množinou dát (analýza dát v zoznamoch, porovnávanie textov).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK2) hierarchická dekompozícia – hierarchicky rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (vytvoriť diagram, zobrazíť hierarchiu aj podproblémy rozdeliť na menšie podproblémy najmä v časti analýzy riešených problémov).</li> </ul> Hľadanie vzorov: <ul style="list-style-type: none"> <li>(VZO5) použiť vzory z jedného problému na druhý problém (riešenie izomorfných úloh).</li> </ul>

- riešiť algoritmické problémy pomocou cyklov, podmienených príkazov, vnorených riadiacich štruktúr, dátového typu zoznam, generovania výnimiek, odchyťovania výnimiek a i ich kombinácií.

### **Riešený didaktický problém**

Nové príkazy alebo riadiace štruktúry sa často žiakom predstavujú na „jednorazových“, pre tento účel definovaných úlohách. Žiaci sa tak možno naučia technické programátorské zručnosti, ale nemajú skúsenosť s ich reálnym využitím. V tejto metodike sme preto zvolili jeden kontext a v rámci neho žiaci „trénujú“ použitie programátorských konceptov. Zadania úloh sú formulované jazykom reálneho zadávateľa. Úlohou žiakov je analyzovať požiadavky úlohy a správne určiť a použiť nástroje programovacieho jazyka. Vďaka sile výrazových prostriedkov jazyka Python ide často o jeden, prípadne kombinácie malého počtu príkazov.

### **Dominantné vyučovacie metódy a formy**

- Problémové vyučovanie,
- frontálna a individuálna forma.

### **Príprava učiteľa a pomôcky**

Pre učiteľa:

- **ucitel/programovanie\_v\_pythone.pdf**  
metodika vyučovania,
- **ucitel/pracovny\_zosit\_riesene\_ulohy.docx**  
pracovný zošit a riešenia úloh,
- **ucitel/pracovne\_subory\_riesenia/22/**  
riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu,
- **ucitel/testy/test4/**  
didaktický test, javová analýza testu, tabuľka pre vyhodnotenie testu.

Pre žiaka:

- **ziak/pracovny\_zosit.docx**  
pracovný zošit,
- **ziak/pracovne\_subory/22/**  
pracovné súbory pre žiaka.

Použitie digitálnych nástrojov: NUTNÉ

### **Diagnostika splnenia vzdelávacích cieľov**

Výsledky žiackych riešení úloh z pracovného listu, výsledky testu IV.

## Úvod

Toto je 22. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Cieľom metodiky je precvičenie a upevnenie poznatkov získaných v rámci metodík 17 až 21. Úlohy sú zamerané na spracovanie zoznamov pomocou cyklov, podmienených príkazov a vnorených riadiacich štruktúr.

Po absolvovaní tejto metodiky by mal učiteľ zaradiť hodinu určenú na preverenie žiackych vedomostí formou testu, ktorého príklad (spolu s riešením a javovou analýzou) je prílohou tejto metodiky.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny:

- **Predstavenie cieľov vyučovania (3 minúty)** – oboznámenie žiakov s vyučovacími cieľmi hodiny.
- **Precvičovanie, práca žiakov (32 minút)** – riešenie úloh z pracovného listu (úlohy 1 až 7).
- **Zhrnutie (5 minút)** – diskusia o problematických častiach riešení úloh.

### PREDSTAVENIE CIEĽOV VYUČOVANIA (CCA 5 MIN)

Na úvod predstavíme žiakom cieľ hodiny – precvičenie a systematizáciu poznatkov z predchádzajúcich metodík. Pri riešení úloh budú žiaci pracovať so zoznamami, s cyklami, podmienenými príkazmi, s vnorenými riadiacimi štruktúrami a s výnimkami.

### PRECVIČOVANIE, SAMOSTATNÁ PRÁCA ŽIAKOV (CCA 32 MIN)

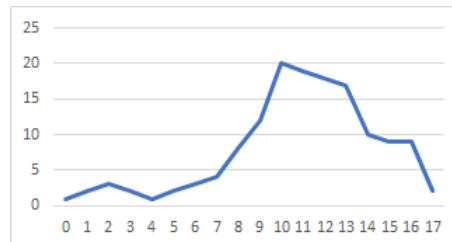
Nasledujúce úlohy sa týkajú rovnakého kontextu – spracovaniu záznamov z výškomera zaznamenaných počas turistického výletu. Každá úloha je zameraná na vyhodnotenie niektorého z parametrov výletu. Z programátorského pohľadu, žiaci vytvárajú funkcie, ktoré zisťujú niektoré vlastnosti zoznamov alebo ich prvkov. Aby úlohy nepôsobili umelo a samoúčelne ale dostali reálny zmysel a využitie, údaje v zoznamoch predstavujú záznamy výšok.

Žiaci by mali primárne pracovať samostatne. Vzájomnú komunikáciu medzi žiakmi k riešeniu úloh umožníme. Odporúčame učiteľovi priebežne sledovať prácu žiakov a ak uzná za vhodné, tak žiakov vhodnými otázkami smerovať k správne riešeniu.

#### Úvodný text

*Tento text je spoločný pre všetky nasledujúce úlohy.*

*Turisti si pomocou výškomera priebežne zaznamenávajú nadmorskú výšku miest, ktorými prechádzajú počas turistických výletov. Používajú výškomer, ktorý v pravidelných jednominútových intervaloch zaznamenáva nadmorskú výšku. Zaznamenávanie spustia pri štarte a zastavia ho, keď prídu na koniec trasy. V ročenke potom pri prejdenej trasách uvádzajú aj ich základné charakteristiky.*



Spracovanie záznamov do ročenky by si radi čo najviac uľahčili. Hodili by sa im programy alebo funkcie, ktoré by dokázali záznamy výškomera vyhodnocovať.

Riešenia nasledujúcich úloh ukladajte do súboru **turistika.py**.

Úloha 1 je zameraná na zisťovanie základných charakteristík zoznamov. Pri zisťovaní dĺžky trvania trasy je potrebné zohľadniť počet úsekov medzi meraniami a nie počet meraní (medzi dvoma meraniami je časový rozdiel 1 minúta).

**Úloha 1** Vytvorte funkcie:

- a) `trvanie_trasy()`, ktorá pre zadaný záznam trasy vráti trvanie trasy v minútach,
- b) `najvyssi_bod()`, ktorá pre zadaný záznam trasy vráti najvyšší bod (v metroch) trasy.

Riešenie:

```
def trvanie_trasy(zaznam):  
    return len(zaznam) - 1  
  
def najvyssi_bod(zaznam):  
    return max(zaznam)
```

Úloha 2 je zameraná na prechod zoznamom a akumulovanie vybraných čiastkových výsledkov (súčet čiastkových stúpaní, resp. klesaní). Časť b) je izomorfná úloha k úlohe a). Jej zaradenie ponechávame na učiteľovi.

**Úloha 2** Vytvorte funkcie:

- Časť b) riešte podľa pokynov učiteľa
- a) `celkove_stupanie()`, ktorá pre zadaný záznam trasy vráti jej celkové stúpanie (súčet všetkých stúpaní na trase),
  - b) `celkove_klesanie()`, ktorá pre zadaný záznam trasy vráti jej celkové klesanie (súčet všetkých klesaní na trase).

Riešenie:

```
def celkove_stupanie(zaznam):  
    stupanie = 0  
    for idx in range(len(zaznam) - 1):  
        rozdiel = zaznam[idx + 1] - zaznam[idx]  
        if rozdiel > 0:  
            stupanie = stupanie + rozdiel  
    return stupanie  
  
def celkove_klesanie(zaznam):  
    klesanie = 0  
    for idx in range(len(zaznam) - 1):  
        rozdiel = zaznam[idx] - zaznam[idx + 1]  
        if rozdiel > 0:  
            klesanie = klesanie + rozdiel  
    return klesanie
```



Úloha 3 je zameraná na prechod zoznamom a nájdenie maxima – trvanie najdlhšieho súvislého stúpania. Funkciu `max()` žiaci využili v úlohe 1. V tomto prípade hľadáme najväčšiu časť zoznamu s uvedenou vlastnosťou. Na trase môže byť stúpajúcich úsekov viac. Je preto potrebné si pri prehľadávaní priebežne uchovávať doposiaľ najdlhší čas súvislého stúpania a v prípade potreby ho aktualizovať. Časť b) je izomorfná úloha k úlohe a). Jej zaradenie ponechávame na učiteľovi.

### Úloha 3 Vytvorte funkcie:

- Časť b) riešte podľa pokynov učiteľa
- `najdlhsie_stupanie()`, ktorá pre zadaný záznam trasy vráti trvanie úseku, v ktorom sa najdlhšie stúpalo,
  - `najdlhsie_klesanie()`, ktorá pre zadaný záznam trasy vráti trvanie úseku, v ktorom sa najdlhšie klesalo.

Riešenie:

```
def najdlhsie_stupanie(zaznam):
    naj_cas_stupania = 0
    akt_cas_stupania = 0
    for idx in range(len(zaznam) - 1):
        if zaznam[idx] < zaznam[idx + 1]:
            akt_cas_stupania = akt_cas_stupania + 1
            if akt_cas_stupania > naj_cas_stupania:
                naj_cas_stupania = akt_cas_stupania
        else:
            akt_cas_stupania = 0
    return naj_cas_stupania

def najdlhsie_klesanie(zaznam):
    naj_cas_klesania = 0
    akt_cas_klesania = 0
    for idx in range(len(zaznam) - 1):
        if zaznam[idx] > zaznam[idx + 1]:
            akt_cas_klesania = akt_cas_klesania + 1
            if akt_cas_klesania > naj_cas_klesania:
                naj_cas_klesania = akt_cas_klesania
        else:
            akt_cas_klesania = 0
    return naj_cas_klesania
```

Úloha 4 je zameraná na vytváranie nových zoznamov z prvkoch iných štruktúr. Odporúčame smerovať žiakov tak, aby úlohu riešili využitím generátorovej notácie zoznamu. Žiaci by si mali vopred premyslieť, ako počiatočnú chybu výškomera reprezentovať – čo znamená kladná a čo záporná hodnota. V autorskom riešení kladná hodnota znamená, že výškomer meria viac než je skutočnosť. Pri korekcii teda chybu (`odchylka_pri_starte`) odpočítavame.

**Úloha 4** Občas sa stane, že turista pri štarte zabudne kalibrovať výškomer. Všetky záznamy výškomera sú potom posunuté smerom hore alebo smerom dole podľa toho, aká bola počiatočná odchýlka výškomera.

- Vytvorte funkciu `posun_vysky()`, ktorá pre zadaný záznam trasy a odchýlku výškomera pri štarte vráti upravený záznam.

Riešenie:

```
def posun_vysky(zaznam, odchylka_pri_starte):
    return [vyska - odchylka_pri_starte for vyska in zaznam]
```

Nasledujúca úloha je zameraná na precvičenie príkazu cyklu s podmienkou. Prechádzame postupne zoznamom od začiatku, kým nenarazíme na klesanie. Analogicky aj z druhej strany zoznamu. Prechádzame zoznamom od konca a hľadáme miesto, kde trasa začne stúpať. Ak obidve nájdené miesta sú na rovnakej pozícii, trasu označíme za vrcholovku.

Zadaniu vyhovujú aj trasy typu „len smerom hore“, „len smerom dolu“ a „len po rovine“. Formulácii typu „trasa nikde neklesá“ vyhovuje nie len konštantný priebeh, ale aj nulovú dĺžku časti zoznamu. Ak by žiaci uvažovali tak, že musí byť nejaké stúpanie zo začiatku a musí byť nejaké klesanie ku koncu, môžeme akceptovať aj tento prístup.

**Úloha 5** Niektoré trasy sú typu „vrcholovka“. Priebeh týchto trás je charakteristický tým, že od začiatku trasy až na vrchol trasa nikde neklesá a z vrcholu až na koniec trasa nikde nestúpa.

- Vytvorte funkciu `je_vrcholovka()`, ktorá pre zadaný záznam trasy vráti informáciu, či trasa je typu „vrcholovka“.

Riešenie:

```
def je_vrcholovka(zaznam):
    idx = 0
    while idx < len(zaznam) - 1 and zaznam[idx] <= zaznam[idx + 1]:
        idx = idx + 1
    koniec_prveho_stupania = idx

    idx = len(zaznam) - 1
    while idx > koniec_prveho_stupania and zaznam[idx] <= zaznam[idx - 1]:
        idx = idx - 1
    zaciatok_posledneho_klesania = idx

    return koniec_prveho_stupania == zaciatok_posledneho_klesania
```

Úloha 6 je zameraná na spájanie zoznamov. Podľa zadania „do prvého zadaného záznamu pridá údaje“. Neočakávame teda nový zoznam, v ktorom sú dva zadané zoznamy zlúčené. Riešenie podľa zadania však vyžaduje, aby si žiaci uvedomili, že zoznam je meniteľná štruktúra. Nech k zoznamu pristupujeme z rôznych miest (v hlavnom programe cez jedno pomenovanie a vo funkcii cez meno `povodny`, stále je to jeden a ten istý zoznam). Zmena na jednom mieste sa teda prejaví aj na inom mieste. Napriek vyššie uvedenému odporúčame, aby učiteľ zvážil a akceptoval aj riešenie, kde funkcia vráti nový zoznam.

Pri spájaní záznamov výškomera by si žiaci mali uvedomiť, že koniec prvého a začiatok druhého zoznamu predstavujú ten istý bod trasy. Prvý bod druhého zoznamu môžeme vynechať. V riešení b) sa

predpokladá korekcia údajov v druhom zázname tak, aby spoločné body mali rovnakú výšku. Predpokladáme, že žiaci využijú už definovanú funkciu `posun_vysky()`.

**Úloha 6** V niektorých prípadoch si turisti prácu rozdelia a každý prejde a zaznamená len časť trasy. Posledný zaznamenaný údaj prvého turistu predstavuje ten istý bod trasy ako prvý zaznamenaný údaj druhého turistu. Pred vyhodnotením musia záznamy spojiť.

Časť b)  
riešte  
podľa  
pokynov  
učiteľa

- Vytvorte funkciu `pripoj_zaznam()`, ktorá do prvého zadaného záznamu pridá údaje z druhého záznamu.
- Upravte funkciu `pripoj_zaznam()` tak, aby pridávané výšky z druhého záznamu boli upravené (posunuté) tak, aby prvá výška v druhom zázname bola rovnaká, ako posledná výška v prvom zázname.

Riešenie a):

```
def pripoj_zaznam(povodny, pridavany): # Riešenie A
    povodny.extend(pridavany[1:])
```

Riešenie b):

```
def pripoj_zaznam(povodny, pridavany): # Riešenie B
    zmena = pridavany[0] - povodny[-1]
    pridavany_uprava = posun_vysky(pridavany[1:], zmena)
    povodny.extend(pridavany_uprava)
```

Úloha 7 je zameraná na vytváranie nových zoznamov z iných štruktúr. V tomto prípade by sa dala využiť generátorová notácia, ale riešenie by bolo pomerne neprehľadné, pretože na vytvorenie prvku nového zoznamu, potrebujeme spracovať tri prvky pôvodného zoznamu.

**Úloha 7** V ročenke sa pri popise trasy uvádza aj jej profil. Výškomer je citlivý prístroj a niektoré faktory prostredia spôsobia, že zaznamenaná výška sa mierne líši od skutočnosti a výsledný profil je trochu „roztrásený“.

- Vytvorte funkciu `vyhlad_zaznam()`, ktorá vráti vyhladený záznam trasy. Záznam trasy vyhladí tak, že každú hodnotu nahradí priemerom troch hodnôt v jej okolí (hodnotu na pozícii `idx` nahradí priemerom hodnôt na pozíciách `idx-1`, `idx` a `idx+1`. Prvý, resp. posledný bod trasy nahradí priemerom prvých, resp. posledných dvoch hodnôt.
- Upravte funkciu `vyhlad_zaznam()` tak, aby v prípade, že sa záznam nedá pre malý počet údajov vyhodnotiť, funkcia vyhodila zmysluplnú výnimku.

Riešenie:

```
def vyhlad_zaznam(zaznam): # Riešenie A
    if len(zaznam) < 2: # Riešenie B
        raise ValueError('Záznam sa nedá vyhladiť. Malý počet údajov.')
    upraveny = []
    upraveny.append(sum(zaznam[:2]) / 2)
    for idx in range(1, len(zaznam) - 1):
        upraveny.append(sum(zaznam[idx - 1:idx + 2]) / 3)
    upraveny.append(sum(zaznam[-2:]) / 2)
    return upraveny
```

**ZHRNUTIE (CCA 3 MIN)**

Vyzveme žiakov, aby sa vyjadrili, či mali pri samostatnej práci nejaké problémy s úlohami, čo bolo pre nich ťažké, resp. ktoré z ponúknutých programátorských problémov zvládli bez väčších komplikácií. Odporúčame, aby učiteľ stručne zhrnul kľúčové prvky, ku ktorým precvičovanie smerovalo.

## 23 GRAFICKÉ POUŽÍVATEĽSKÉ ROZHRAKIE – TLAČIDLÁ, TEXTOVÉ POLIA, POPISY

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>pomocou premenných,</li> <li>pomocou nástrojov na interakciu.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>odchytať a generovať výnimky,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus a podmienený príkaz pri riešení problémov.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie.</li> </ul> <b>Pomocou premenných:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých si treba zapamätať a neskôr použiť zapamätané hodnoty vo výrazoch.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba zobraziť výstup, realizovať akciu,</li> <li>zapisovať algoritmus, ktorý reaguje na vstup.</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>vytvárať programy využívajúce grafické používateľské rozhranie pomocou modulu <b>tkinter</b> (hlavne pomocou grafických prvkov <b>Button, Entry, Label, messagebox</b>, špeciálnych premenných modulu <b>tkinter</b> typu <b>StringVar</b> a manažéra rozmiestnenia obsahu <b>grid</b>).</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa jednoduchých programov.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (vytváranie grafického rozhrania),</li> <li>(ALG6) dotvárať nekompletné algoritmy (doplniť chýbajúci kód v programe).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia – lineárne rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (rozdelenie programu na menšie časti/funkcie pre akcie tlačidiel).</li> </ul>
<b>Riešený didaktický problém</b>	
<p>Prirodzenou súčasťou kurzu programovania je tvorba programov využívajúcich grafické používateľské rozhranie, ktoré je pre používateľov atraktívnejšie ako programy s konzolovými vstupmi a výstupmi. V našom prípade, sme túto tému zaradili až na záver kurzu programovania, aby sme žiakom umožnili v predchádzajúcich témach sústrediť sa na osvojenie a precvičenie základných riadiacich konštrukcií, jednoduchých dátových typov a typu zoznam. Tvorbu programov s grafickým používateľským rozhraním pokladáme za prirodzené vyvrcholenie kurzu programovania, kde žiaci preukážu schopnosť vytvoriť graficky atraktívny program, ktorý</p>	

zároveň bude robustný voči chybným vstupom. Pri programovaní používame modul **tkinter**, z ktorého vyberáme veľmi obmedzený počet prvkov grafického rozhrania a ich metód, aby sme žiakov nezaťažovali zbytočným množstvom učiva, ktoré nepotrebujú počas tejto vyučovacej hodiny. Pri programovaní s modulom **tkinter** majú niekedy žiaci problém so syntaxou (napr. nesprávne použité veľké/malé písmená v názve metódy či vlastnosti, chýbajúce zátvorky v názve metódy alebo zátvorky navyše v parametri command, atď.) či s dôsledným ošetrovaním robustnosti programu.

<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• individuálna a skupinová forma práce žiakov.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit a riešenia úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/23/</b> riešené pracovné úlohy a tabuľka pre zápis výsledkov žiackych riešení úloh z pracovného zošitu.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/23/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.	

## Úvod

Toto je 23. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. V metodikách 2 až 6 sme sa venovali tvorbe programov, ktoré vykresľovali obrázky pomocou korytnačej grafiky. V ostatných predchádzajúcich metodikách sme sa venovali výpočtom s číslami či textami. V tejto a zvyšných štyroch metodikách sa žiaci naučia vytvárať programy využívajúce grafické používateľské rozhranie pomocou modulu **tkinter**.

Žiaci majú k dispozícii pracovný list, ktorý obsahuje zadania úloh, miesto na žiacke riešenie a miesto pre poznámky. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Poslednou časťou je sebahodnotiaci test.

## PRIEBEH VÝUČBY

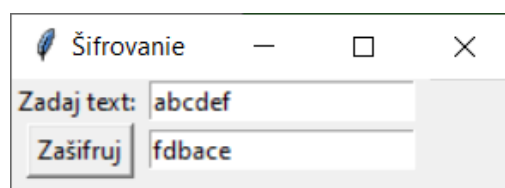
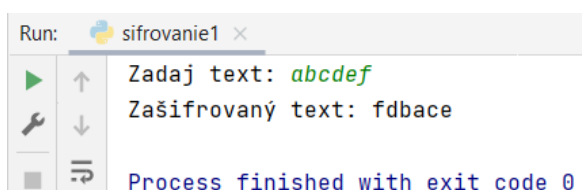
Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (3 minúty)** – rozhovor so žiakmi – porovnanie dvoch prístupov pri riešení výpočtovej úlohy – konzola vs. grafické používateľské rozhranie (úloha 1)
- **Skúmanie (12 minút)** – skúmanie prvkov grafického rozhrania, ich rozmiestnenia na obrazovke, prepojenia vstupov a výstupov pomocou špeciálnych premenných (úloha 2 a 3)
- **Vysvetlenie (10 minút)** – zhrnutie a vysvetlenie zistení z časti Skúmanie problematiky tvorby programov s grafickým používateľským rozhraním (úloha 4)
- **Rozpracovanie (10 minút)** – precvičenie vytvorenia grafického programu pomocou modulu tkinter (úloha 5)
- **Vyhodnotenie (5 minút)** – kontrola žiackych prác, vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 3 MIN)

V **úlohe 1** prepojíme staré (výpočty v konzole) a nové učivo (výpočty v grafickom používateľskom rozhraní) a formulujeme cieľ hodiny – vytvoriť programy s grafickým používateľským rozhraním.

**Úloha 1** Otvorte programy **sifrovanie1.py** a **sifrovanie2.py** a spustite ich s rovnakými vstupnými dátami.



- Uvedte, v ktorom z týchto programov sa Vám lepšie pracovalo a zdôvodnite aj prečo.
- Uvedte, ktoré prvky grafického rozhrania sú použité v programe **sifrovanie2.py**. Ktoré z nich slúžia ako

vstupy resp. výstupy?

Riešenie:

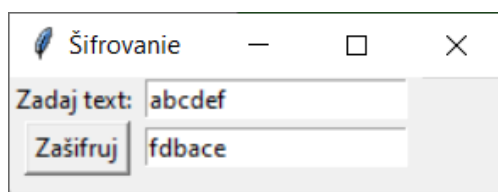
- a) Druhý program je prehľadnejší a môžeme zadať rôzne vstupy bez opätovného spustenia programu.  
b) Prvky grafického rozhrania sú: okno, popis, tlačidlo, textové políčka (jedno je vstup, druhé výstup).

## SKÚMANIE (CCA 12 MIN)

V nasledujúcich úlohách žiaci analyzujú programové kódy s rôznymi prvkami grafického rozhrania (angl. widgets) a zisťujú ich význam a diskutujú ich použitie.

Po skúmaní v **úlohe 2** by mali žiaci získať predstavu o najpoužívanejších prvkoch grafického rozhrania (**Label**, **Entry**, **Button**) a o ich umiestňovaní do mriežky pomocou manažéra grafického obsahu **grid**. Pri riešení našich úloh vystačíme s týmto manažérom, ostatné dva manažéry **pack** a **place** žiakom neuvádzame, aby sme ich nezaťažovali zbytočným učivom. Uvedieme ich, len ak sa žiaci na nich spýtajú.

**Úloha 2** Otvorte program **sifrovanie2.py**. Spustite ho a preskúmajte jednotlivé prvky grafického rozhrania.



Do mriežky s grafickým rozhraním doplňte na mieste červených bodiek použité prvky grafického rozhrania a tiež čísla riadkov (angl. rows) a stĺpcov (angl. columns), kde sú umiestnené (číslovanie začíname od 0).

	stĺpec ...	stĺpec ...
riadok ...	...	...
riadok ...	...	...

Do tabuľky doplňte význam a použitie uvedených prvkov grafického rozhrania.

Grafický prvok	Význam a použitie grafického prvku	Grafický prvok	Význam a použitie grafického prvku
<b>Label</b>		<b>Entry</b>	
<b>Button</b>		<b>Entry</b>	



Riešenie:

	stĺpec 0	stĺpec 1
riadok 0	Label	Entry
riadok 1	Button	Entry

Grafický prvok	Význam a použitie grafického prvku	Grafický prvok	Význam a použitie grafického prvku
Label	Textový popis (k vedľajšiemu prvku)	Entry	Textové políčko na vstup dát
Button	Tlačidlo na spúšťanie akcii (funkcií)	Entry	Textové políčko na výstup dát

Ak to dovoľí čas, žiaci by mohli zistiť zaujímavú vlastnosť tohto šifrovania, že pri opakovanom zašifrovaní už zašifrovaného textu dostaneme pôvodný otvorený text.

**Úloha 3** Otvorte súbor **sifrovanie2.py**. Spustite súbor a preskúmajte ako je v ňom realizovaný výpočet.

```
import tkinter

def vypocet():
    ''' Vráti zašifrovaný text zo zadaného otvoreného textu

    :param text_otvoreny: otvorený text
    :type text_otvoreny: StringVar
    '''
    vstup = text_otvoreny.get()
    if len(vstup) % 2 == 0:
        vystup = vstup[-1::-2] + vstup[0::2]
    else:
        vystup = vstup[-2::-2] + vstup[0::2]
    text_sifrovany.set(vystup)

okno = tkinter.Tk()
okno.title('Šifrovanie')

# riadok 0: Label, Entry
tkinter.Label(okno, text='Zadaj text: ').grid(row=0, column=0)

text_otvoreny = tkinter.StringVar()
text_otvoreny.set('abcdef')
tkinter.Entry(okno, textvariable=text_otvoreny).grid(row=0, column=1)

# riadok 1: Button, Entry
tkinter.Button(okno, text='Zašifruj', command=vypocet).grid(row=1,
column=0)

text_sifrovany = tkinter.StringVar()
```

```
text_sifrovany.set('?')
tkinter.Entry(okno, textvariable=text_sifrovany).grid(row=1, column=1)

okno.mainloop()
```

a) Uvedte názvy premenných, ktoré boli vytvorené ako špeciálne premenné modulu tkinter typu **StringVar**:

.....

b) Uvedte význam metód

premenná.set(): .....

premenná.get(): .....

c) Uvedte význam špeciálnych premenných modulu tkinter typu **StringVar**:

.....

Riešenie:

a) text\_otvoreny, text\_sifrovany.

b) premenna.set() sa používa na nastavenie hodnoty premennej,  
premenna.get() sa používa na načítanie (zistenie) hodnoty premennej.

c) pomocou špeciálnych premenných modulu tkinter typu StringVar vieme načítať, spracovať či vypísať informácie zadané v grafických prvkoch typu Entry.

## VYSVETLENIE (CCA 10 MIN)

Prostredníctvom nasledovných otázok prediskutujeme so žiakmi čo zistili v úlohách 1 až 3.

1. Aké sú pozitíva a negatíva grafických programov?  
(Možná odpoveď: Oproti konzolovým grafické programy vyzerajú profesionálnejšie, len treba pri ich tvorbe napísať viac riadkov programového kódu.)
2. Aké prvky grafického prostredia poznáte?  
(Možná odpoveď: Button, Entry, Label.)
3. Ako sa rozmiestnili v okne jednotlivé grafické prvky?  
(Možná odpoveď: Použili sme metódu grid() na umiestnenie grafických prvkov do riadkov a stĺpcov pomyslenej mriežky.)
4. Pri ktorých prvkoch grafického rozhrania používame špeciálne premenné modulu tkinter typu StringVar? Na čo slúžia?  
(Možná odpoveď: Pri grafických prvkoch Entry, ktoré slúžia ako vstupné resp. výstupné textové políčka, používame špeciálne premenné modulu tkinter typu StringVar na prepojenie funkcií programu s grafickým rozhraním.)

Nasledovným spoločným upravovaním konzolového programu na grafický program na výpočet BMI vysvetlíme princíp tvorby grafických programov využívajúcich modul tkinter.

**Úloha 4** Otvorte program na výpočet BMI **bmi\_kategorie1.py** a upravte ho z konzolovej verzie na grafickú verziu a výsledný programový kód uložte do súboru **bmi\_kategorie2.py**.

Riešenie:

Z pôvodného konzolového programu ponecháme funkcie **pocitaj\_bmi()** a **kategoria\_bmi()**. Navrhujeme grafické rozhranie programu,

	stĺpec 0	stĺpec 1
riadok 0	Label (Zadaj hmotnosť)	Entry (StringVar hmotnosť)
riadok 1	Label (Zadaj výšku)	Entry (StringVar vyska)
riadok 2	Button (Výpočet, zobraz_kategoriu_bmi)	
riadok 3	Label (Kategória BMI)	Entry (StringVar bmi_kat)
riadok 4	Button (Koniec, okno.destroy)	

ktoré implementujeme programovým kódom:

```
okno = tkinter.Tk()
okno.title('Výpočet kategórie BMI')

tkinter.Label(okno, text='Zadaj hmotnosť v kg:').grid(row=0, column=0)
hmotnost = tkinter.StringVar()
tkinter.Entry(okno, textvariable=hmotnost).grid(row=0, column=1)

tkinter.Label(okno, text='Zadaj výšku v m:').grid(row=1, column=0)
vyska = tkinter.StringVar()
tkinter.Entry(okno, textvariable=vyska).grid(row=1, column=1)

tkinter.Button(okno, text='Výpočet',
command=zobraz_kategoriu_bmi).grid(row=2, column=0)

tkinter.Label(okno, text='Kategória BMI:').grid(row=3, column=0)
bmi_kat = tkinter.StringVar()
tkinter.Entry(okno, textvariable=bmi_kat).grid(row=3, column=1)

tkinter.Button(okno, text='Koniec', command=okno.destroy).grid(row=4,
column=0)

okno.mainloop()
```

Napokon pôvodný kód hlavnej časti konzolovej programu

```
hmotnost = input('Zadaj hmotnosť v kg: ')
vyska = input('Zadaj výšku v m: ')
try:
    kategoria = kategoria_bmi(hmotnost, vyska)
except ValueError as chyba:
    print(chyba)
else:
    print('Kategória BMI je:', kategoria)
```

nahradíme dvomi funkciami `zobraz_kategoriu_bmi()` a `zobraz_chybu()`:

```
def zobraz_kategoriu_bmi():
    try:
        kategoria = kategoria_bmi(hmotnost.get(), vyska.get())
    except ValueError as chyba:
        zobraz_chybu(chyba)
    else:
        bmi_kat.set(kategoria)

def zobraz_chybu(chyba):
```

```
messagebox.showerror('Chyba', chyba)
```

Vo funkcii **zobraz\_kategoriu\_bmi()** oproti hlavnému programu namiesto globálnych premenných **hmotnost** a **hodnota** používame špeciálne premenné modulu tkinter typu **StringVar** **hmotnost** a **hodnota**, ktorých hodnoty načítame pomocou metódy **get()**. Namiesto vypísania výsledku výpočtu do konzoly použijeme zobrazenie **StringVar** premennej **bmi\_kat** v textovom poli. Na vypísanie chybovej hlášky namiesto výpisu do konzoly použijeme prvok grafického rozhrania **messagebox** – samostatné okno vyvolané funkciou **zobraz\_chybu()**. Aby sme mohli s grafickým prvkom **messagebox** pracovať, potrebujeme na začiatok programu doplniť importovanie tohto grafického prvku:

```
from tkinter import messagebox
```

Stručne zhrnieme, čo sme sa teraz naučili.

Pri tvorbe grafického programu typu vstup-spracovanie-výstup nám stačí použiť základné tri prvky grafického rozhrania **popis** (angl. **Label**) na doprovodný text k vstupným a výstupným **políčkam** (angl. **Entry**) a **tlačidlo** (angl. **Button**) na spustenie výpočtu. Uvedené prvky grafického rozhrania umiestnime na obrazovke programu pomocou manažéra grafického obsahu **grid** do jednotlivých políček mriežky. Na prepojenie prvkov grafického rozhrania (v našom prípade vstupných a výstupných políček) s funkciami pre výpočet použijeme špeciálne premenné modulu tkinter typu **StringVar**, ktorých hodnoty načítavame resp. zapisujeme pomocou metód **get()** resp. **set()**. Robustnosť programu (jeho odolnosť voči zadaniu chybných vstupov) zabezpečíme generovaním a zachytávaním výnimiek. Na rozdiel od konzolového programu pri grafickom programe chybu nevypisujeme do konzoly, ale ju zobrazíme do nového okna pomocou grafického prvku **messagebox**, ktorý potrebujeme zvlášť importovať.

## ROZPRACOVANIE (CCA 10 MIN)

Pomocou úlohy 5 si žiaci môžu precvičiť programovanie výpočtovo orientovaných grafických programov využívajúcich prvky grafického rozhrania **Label**, **Entry**, **Button** a **messagebox**.

**Úloha 5** Vytvorte program na overenie korektnosti zadaného rodného čísla. Prínajmenšom ošetríte zadanie numerických znakov, dĺžku vstupu a deliteľnosť rodného čísla číslom 11. Prípadne môžete ošetriť aj korektnosť dátumov a pohlavia, ktoré je uvedené v úlohe 3 v kapitole 16. Výsledný program uložte do súboru **overenie\_rc.py**.

Riešenie:

	stĺpec 0	stĺpec 1
riadok 0	Label (Zadaj rodné číslo)	Entry (StringVar rodne_cislo)
riadok 1	Button (Over rodné číslo, vypocet)	Entry (StringVar odpoved)

```
import tkinter
from tkinter import messagebox
```

```
def over_cislo(cislo):
```

```

''' Overí platnosť rodného čísla

:param cislo: rodné číslo
:type cislo: str
:return: výsledok
:rtype: str
'''

if not cislo.isdigit():
    raise ValueError('Na vstupe boli zadane aj nenumericke znaky')
if len(cislo) not in [9, 10]:
    raise ValueError('Rodné číslo musí mať 9 alebo 10 číslic')
if int(cislo) % 11 != 0:
    raise ValueError('Rodné číslo musí byť deliteľné 11')
else:
    return 'OK'

def vypocet():
    vstup = rodne_cislo.get()
    vystup = 'Chyba'
    try:
        vystup = over_cislo(vstup)
    except ValueError as chyba:
        zobraz_chybu(chyba)
    odpoved.set(vystup)

def zobraz_chybu(chyba):
    messagebox.showerror('Chyba', chyba)

okno = tkinter.Tk()
okno.title('Overenie rodného čísla')

# riadok 0: Label, Entry
tkinter.Label(okno, text='Zadaj rodné číslo bez lomky: ').grid(row=0,
column=0)

rodne_cislo = tkinter.StringVar()
rodne_cislo.set('')
tkinter.Entry(okno, textvariable=rodne_cislo).grid(row=0, column=1)

# riadok 1: Button, Entry
tkinter.Button(okno, text='Over rodné číslo', command=vypocet).grid(row=1,
column=0)

odpoved = tkinter.StringVar()
odpoved.set('?')
tkinter.Entry(okno, textvariable=odpoved).grid(row=1, column=1)

okno.mainloop()

```

## VYHODNOTENIE (CCA 5 MIN)

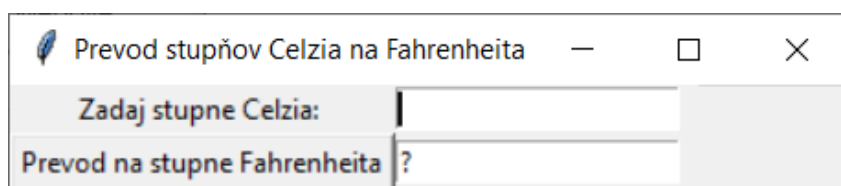
Na konci hodiny požiadame žiakov, aby vypracovali sebahodnotiaci test, ktorého je primárne nástrojom formatívneho hodnotenia výsledkov učenia sa žiakov. Následne žiakom poskytneme spätnú

väzbu, aby vedeli, ktoré ich odpovede sú správne a ktoré nesprávne. Problematické odpovede môžeme s nimi prediskutovať.

### Sebahodnotiaci test

1. Doplňte niektoré z uvedených šiestich textov na označené miesta **1 2 3** v programe na výpočet prevodu stupňov Celzia na stupne Fahrenheita.

- `stupne_Celzia.get()`
- `stupne_Celzia.set(vystup)`
- `stupne_Fahrenheita()`
- `stupne_Fahrenheita.set(vystup)`
- `vypocet()`
- `vypocet`



```
def vypocet():
    vstup = 1
    try:
        vystup = 32 + 1.8 * float(vstup)
    except ValueError:
        zobraz_chybu('Nezadali ste číselný vstup')
    2

tkinter.Button(okno, text='Prevod na stupne Fahrenheita',
command=3).grid(row=1, column=0)

stupne_Fahrenheita = tkinter.StringVar()
stupne_Fahrenheita.set('?')
tkinter.Entry(okno, textvariable=stupne_Fahrenheita).grid(row=1, column=1)
```

Riešenie:

- 1** `stupne_Celzia.get()`
- 2** `stupne_Fahrenheita.set(vystup)`
- 3** `vypocet`

V tejto úlohe sledujeme, či žiaci vedia aplikovať metódy `set()` a `get()` premenných modulu `tkinter` a pri nastavení grafického prvku tlačidlo použiť meno funkcie na výpočet (t. j. bez nasledujúcich okrúhlych zátvoriek), nie jej volanie.

## 24 GRAFICKÉ POUŽÍVATEĽSKÉ ROZHRAŇIE – PLÁTNO

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>pomocou premenných,</li> <li>pomocou nástrojov na interakciu.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>odchytať a generovať výnimky,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus a podmienený príkaz pri riešení problémov.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie.</li> </ul> <b>Pomocou premenných:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých si treba zapamätať a neskôr použiť zapamätané hodnoty vo výrazoch.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba zobraziť výstup, realizovať akciu,</li> <li>zapisovať algoritmus, ktorý reaguje na vstup.</li> </ul> <b>Programovací jazyk Python:</b> <ul style="list-style-type: none"> <li>vytvárať programy využívajúce grafické používateľské rozhranie pomocou modulu <b>tkinter</b> (hlavne pomocou grafických prvkov <b>Canvas</b> (vykresľovanie úsečky, obdĺžnika, elipsy, polygónu, textu, obrázka), <b>OptionMenu</b>) a dátový typ súbor.</li> </ul>	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG2) využitím logických zdôvodnení predpokladať správanie sa jednoduchých programov.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém (vytváranie grafického rozhrania),</li> <li>(ALG6) dotvárať nekompletné algoritmy (doplniť chýbajúci kód v programe).</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK1) lineárna dekompozícia – lineárne rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (rozdelenie programu na menšie časti/funkcie pre akcie tlačidiel).</li> </ul>
<b>Riešenie didaktický problém</b>	
<p>Prirodzenou súčasťou kurzu programovania je tvorba programov využívajúcich grafické používateľské rozhranie, ktoré je pre používateľov oveľa prívetivejšie ako konzolové vstupy a výstupy. V našom prípade, sme túto tému zaradili na záver kurzu programovania, aby sme žiakom umožnili v predchádzajúcich témach sa sústrediť na základné riadiace konštrukcie, jednoduché dátové typy a typ zoznam, a nezaťažovali ich touto nadstavbovou témou. Pomocou modulu <b>tkinter</b>, konkrétne pomocou grafického prvku <b>Canvas</b> a jeho metód umožníme žiakom riešiť grafické problémy v karteziánskej grafike. Namiesto tréningu širokej škály vykresľovacích metód grafického prvku <b>Canvas</b>, sme sa zamerali na spracovanie a vizualizáciu dát uložených v zoznamoch (vo vnútornej pamäti aj na disku). Tým chceme viac reflektovať realitu, keď máme okolo seba</p>	

dostupné rôzne typy otvorených dát zo senzorov z blízkyh aj vzdialených vstavených zariadení, sociálnych sietí, vedeckých výskumov, štatistických úradov atď.

<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• Bádateľská metóda (model 5E),</li> <li>• individuálna a skupinová forma práce žiakov.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit s riešeniami úloh,</li> <li>• <b>ucitel/pracovne_subory_riesenia/24/</b> riešenia úloh.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/24/</b> pracovné súbory pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu, sebahodnotiaci test.	



## Úvod

Toto je 24. metodika zo série 27 metodík, ktoré sú určené pre základný kurz programovania. V nej sa žiaci naučia vytvárať grafické programy využívajúce grafické prvky **Canvas**, **PhotoImage**, **OptionMenu** modulu **tkinter** a základy práce so súbormi. Po tejto metodike nasledujú posledné tri metodiky zamerané na projektové vyučovanie, v rámci ktorého žiaci zosumarizujú svoje doterajšie programovacie vedomosti a zručnosti.

Žiaci majú k dispozícii pracovný list. Odporúčame, aby učiteľ žiakom pri každej fáze vyučovania uviedol zoznam úloh z pracovného listu, ktoré budú aktuálne riešiť. Po sebahodnotiacom teste by mali žiaci za domácu úlohu premyslieť námet projektu, ktorý budú vytvárať na posledných hodinách.

## PRIEBEH VÝUČBY

Osnova vyučovacej hodiny (podľa modelu 5E):

- **Zapojenie (4 minúty)** – vykreslenie elementárneho obrázka v karteziánskej súradnicovej sústave + diskusia so žiakmi k súradnicovému systému v module **tkinter** (úloha 1)
- **Skúmanie (12 minút)** – skúmanie nového príkazu (úloha 2 až 4)
- **Vysvetlenie (10 minút)** – zhrnutie a vysvetlenie zistení z časti Skúmanie problematiky tvorby programov s grafickým používateľským rozhraním
- **Rozpracovanie (9 minút)** – samostatné programovanie náročnejších úloh (úlohy 5 až 6)
- **Vyhodnotenie (5 minút)** – vyriešenie sebahodnotiaceho testu, diskusia o odpovediach.

## ZAPOJENIE (CCA 4 MIN)

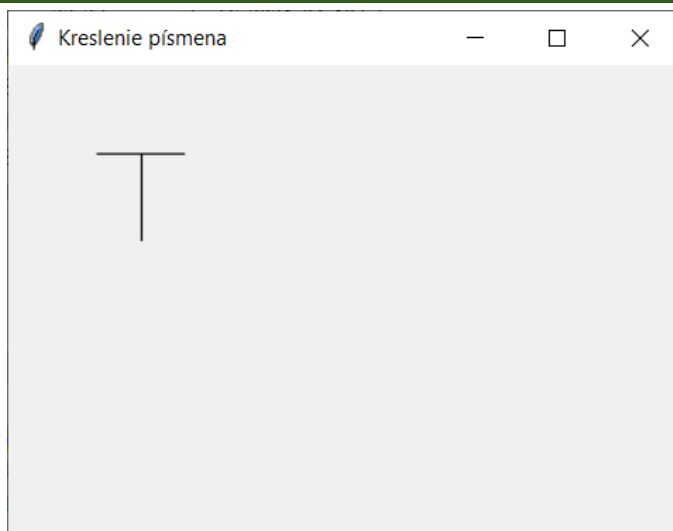
V **úlohe 1** predstavíme vykresľovanie elementárnych obrázkov pomocou modulu **tkinter** a odlišnosti jeho súradnicového systému oproti systému používanom na hodinách matematiky.

**Úloha 1** Otvorte a spustite súbor **pismo.py**.

```
import tkinter

def pismo():
    platno.create_line(50, 50, 100, 50)
    platno.create_line(75, 50, 75, 100)

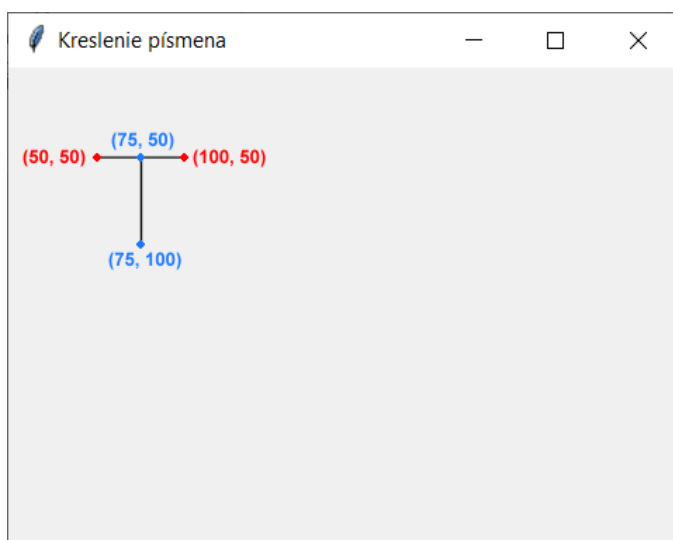
okno = tkinter.Tk()
okno.title('Kreslenie písmena')
platno = tkinter.Canvas(okno)
platno.grid()
pismo()
okno.mainloop()
```



- Do obrázku doplňte súradnice koncových bodov úsečiek vykreslených pomocou funkcie **pismo()**.
- Upravte súradnice jednej z úsečiek vo funkcii **pismo()** tak, aby vykreslila písmeno L.
- V čom sa líši tento súradnicový systém a bežne používaný karteziansky súradnicový systém?

Riešenie:

- Koncové body vodorovnej úsečky majú súradnice (50, 50) a (100, 50) a body zvislej úsečky (75, 50) a (75, 100).



- Upravíme napríklad len súradnice prvej úsečky na (75, 100) a (100, 100).
- Súradnicový systém v module **tkinter** je odlišný od zaužívaného systému v matematike, t. j. počiatok súradnicového systému O bod (0,0) je v ľavom hornom rohu plátna s **osou x** nasmerovanou **dopravo** a **osou y** nasmerovanou **nadol**.

Poznámka:

Pri vykresľovaní na plátno treba brať do úvahy, že pri štandardnom nastavení sa okraj okna prekrýva s okrajom plátna v šírke dvoch bodov, čo spôsobí, že nevidíme na plátno niektoré body napr. (0,0), (1,1). Situáciu vyriešime pomocou nastavenia parametra **highlightthickness** pri inicializácii grafického prvku **Canvas**:

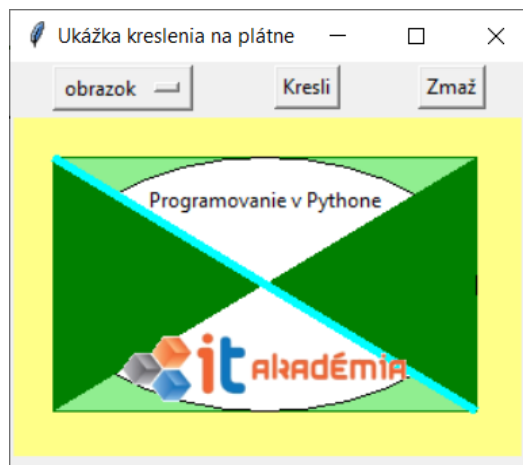
```
platno = tkinter.Canvas(okno, highlightthickness=0)
```

## SKÚMANIE (CCA 12 MIN)

V nasledujúcich dvoch úlohách žiaci analyzujú programové kódy s rôznymi prvkami grafického rozhrania (angl. widgets) a programové kódy so základmi práce so súbormi.

**Úloha 2** umožní žiakom preskúmať význam a použitie nových prvkov grafického rozhrania – **Canvas**, **PhotoImage** a **OptionMenu** v grafickom programe na vykresľovanie rôznych základných útvarov (obdĺžnik, elipsa, úsečka, polygón, text, obrázok) pomocou metód prvku **Canvas**.

**Úloha 2** Otvorte súbor **platno\_ukazka.py**. Spustite ho a preskúmajte jednotlivé grafické prvky a ich metódy.



```

1  import tkinter
2
3
4  def kresli():
5      if vyber.get() == 'obdĺžnik':
6          platno.create_rectangle(25, 25, 275, 175, fill='lightgreen',
outline='green')
7      elif vyber.get() == 'elipsa':
8          platno.create_oval(25, 25, 275, 175, fill='white')
9      elif vyber.get() == 'čiara':
10         platno.create_line(25, 25, 275, 175, fill='cyan', width=5)
11         elif vyber.get() == 'lomená čiara':
12             platno.create_polygon(25, 25, 275, 175, 275, 25, 25, 175,
fill='green')
13         elif vyber.get() == 'text':
14             platno.create_text(150, 50, text='Programovanie v Pythone',
fill='black')
15         elif vyber.get() == 'obrázok':
16             platno.create_image(150, 150, anchor='center', image=obrazok)
17
18
19  def zmaz():
20      platno.delete('all')
21
22
23  okno = tkinter.Tk()
24  okno.title('Ukážka kreslenia na plátne')
25  platno = tkinter.Canvas(okno, width=300, height=200, background='#FF8')
26  obrazok = tkinter.PhotoImage(file='ita_logo.gif')
27

```

```

28 # riadok 0: Label, Button, Button
29 zoznam = ['obdĺžnik', 'elipsa', 'čiara', 'lomená čiara', 'text', 'obrázok']
30 vyber = tkinter.StringVar()
31 vyber.set(zoznam[0])
32 tkinter.OptionMenu(okno, vyber, *zoznam).grid(row=0, column=0)
33
34 tkinter.Button(okno, text='Kresli', command=kresli).grid(row=0, column=1)
35
36 tkinter.Button(okno, text='Zmaž', command=zmaž).grid(row=0, column=2)
37
38 # riadok 2: Canvas
39 platno.grid(row=1, column=0, columnspan=3)
40
41 okno.mainloop()

```

- a) Do tabuľky doplňte význam uvedených grafických metód použitých na plátne (v riadkoch 6, 8, 10, 12, 14, 16).

Metóda	Význam metódy
<code>create_rectangle()</code>	
<code>create_oval()</code>	
<code>create_line()</code>	
<code>create_polygon()</code>	
<code>create_text()</code>	
<code>create_image()</code>	

- b) Uveďte, v ktorom stĺpci/stĺpcoch je umiestnené plátno? Ako dosiahneme, že sa natiahne na všetky tri stĺpce? Ktorý parameter na to slúži?
- .....
- c) Uveďte na čo slúži prvok grafického rozhrania **OptionMenu**:
- .....
- d) Uveďte na čo slúži prvok grafického rozhrania **PhotoImage**:
- .....

Riešenie:

- a) Vykreslenie obdĺžnika, elipsy, úsečky, uzavretej lomenej čiary, textu, obrázku zo súboru.
- b) V programe na riadku 39 je plátno umiestnené v mriežke v riadku 1 a stĺpci 0 s natiahnutím na 3 stĺpce pomocou nastavenia parametra **columnspan** na hodnotu 3.
- c) Prvok **OptionMenu** slúži na výber jednej z uvedených možností rozbaľovacej ponuky.
- d) Prvok **PhotoImage** slúži na prácu s obrázkom uloženom v grafickom súbore (vo formáte gif, png, a i.)

**Úloha 3** umožní žiakom preskúmať vykresľovanie čiar s dĺžkami zadanými v zozname pomocou metódy **create\_line()** grafického prvku Canvas. Okrem zadania zoznamu priamo v súbore, necháme žiakom, aby si vyskúšali načítanie zoznamu dát zo súboru pomocou vytvorenej funkcie **nacitaj\_data()**. Túto funkciu žiakom poskytneme primárne ako čiernu skrinku, ktorá pre zadaný vstup (meno súboru) vráti požadovaný výstup (zoznam dát zo súboru).

**Úloha 3** Otvorte a spustite súbor *stalaktity.py*.

```
1 import tkinter
2
3
4 def nacitaj_data(meno_suboru):
5     ''' Načítanie dát zo súboru so zadaným menom do zoznamu
6
7     :param meno_suboru: zadané meno súboru s dátami
8     :type meno_suboru: str
9     :return: zoznam riadkov s dátami
10    :rtype: list of str
11    '''
12    with open(meno_suboru, 'r', encoding='utf-8') as f:
13        data = f.readlines()
14    data = [polozka.strip() for polozka in data]
15    return data
16
17
18 def kresli(zoznam):
19     for x in range(len(zoznam)):
20         platno.create_line(x + 5, 5, x + 5, zoznam[x])
21
22
23 okno = tkinter.Tk()
24 okno.title('Vykreslenie stalaktitov')
25 platno = tkinter.Canvas(okno, width=300, height=300, background='yellow')
26 platno.grid()
27
28 zoznam = [50, 40, 90, 70, 20, 60, 80, 10, 30, 100]
29 # zoznam = nacitaj_data('stalaktity1.txt')
30 # zoznam = nacitaj_data('stalaktity2.txt')
31 # zoznam = nacitaj_data('stalaktity3.txt')
32 # zoznam = nacitaj_data('stalaktity4.txt')
33 kresli(zoznam)
34
35 okno.mainloop()
```

- Akú zmenu vo funkcii **kresli()** treba urobiť, aby sme stalaktity nezobrazovali zhora nadol, ale ako úsečky zľava doprava?
- Ako sa zmení správanie programu, ak zakomentujeme riadok 28 a odstránime znak komentára z riadku 29 (resp. 30, 31 alebo 32)?
- Čo robí funkcia **nacitaj\_data()**?
- Ktorý zo spôsobov považujete za lepší? Spracovanie hodnôt zoznamu uvedeného v programe, alebo v súbore mimo programu?

Riešenie:

- Stačí v programe v riadku 28 v metóde **create\_line()** prehodiť prvý s druhým a tretí so štvrtým parametrom.
- Namiesto hodnôt uvedených v programe v zozname sa spracujú hodnoty uložené v súbore **stalaktity1.txt** (resp. **stalaktity2.txt**, **stalaktity3.txt** alebo **stalaktity4.txt**).
- Funkcia načíta dáta zo súboru so zadaným menom a vráti ich ako zoznam dát.
- Druhý spôsob je lepší, lebo oddelíme program od spracovávaným dát. A bez zmeny v programe môžeme spracovať rôzne viaceré dátové súbory.

## VYSVETLENIE (CCA 10 MIN)

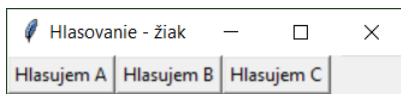
Prostredníctvom nasledovných otázok prediskutujeme so žiakmi čo zistili v **úlohách 1 až 3**.

1. Ako je orientovaný súradnicový systém na plátne modulu tkinter? Kde sa nachádza bod (0, 0)?  
(Možná odpoveď: V súradnicovom systéme plátna modulu tkinter je vodorovná os *x* orientovaná zľava doprava a zvislá **os y** *zhora nadol*.)
2. Ktoré ďalšie nové prvky grafického prostredia poznáte? Uveďte aj na čo slúžia.  
(Možná odpoveď: **Canvas** – kresliace plátno, **PhotoImage** – obrázok zo súboru, ktorý sa dá zobraziť na plátne, **OptionMenu** – rozbaľovacia ponuka umožňujúca výber jednej z uvedených možností.)
3. Ktoré metódy grafického prvku Canvas poznáte? Uveďte, ktoré útvary vieme nimi vykresliť na plátne.  
(Možná odpoveď: **create\_rectangle()** – obdĺžnik, **create\_oval()** – elipsu, **create\_line()** – úsečku, **create\_polygon()** – uzavretú lomenú čiary, **create\_text()** – text, **create\_image()** – obrázok zo súboru.)
4. Kedy je vhodné spracovať hodnoty zoznamu uvedené v programe a kedy, keď sú hodnoty uvedené v súbore?  
(Možná odpoveď: Pri jednoduchých programoch s krátkym zoznamom dát môžeme tieto hodnoty zoznamu uviesť priamo v programe. Vo všeobecnosti ale odporúčame oddeliť dáta od programu a mať ich uložené v súbore, čo umožní spracovať rôzne dáta bez zásahu programového kódu.)

Nasledovným spoločným riešením **úlohy 4** ukážeme a prediskutujeme princíp tvorby grafického programu zameraného na záznam dát do súboru, výpočet a vizualizáciu výsledkov na

**Úloha 4** Vytvorte hlasovací systém pozostávajúci z dvoch programov. V jednom programe budú žiaci hlasovať pomocou troch tlačidiel a v druhom bude učiteľ zobrazovať aktuálny stav hlasovania.

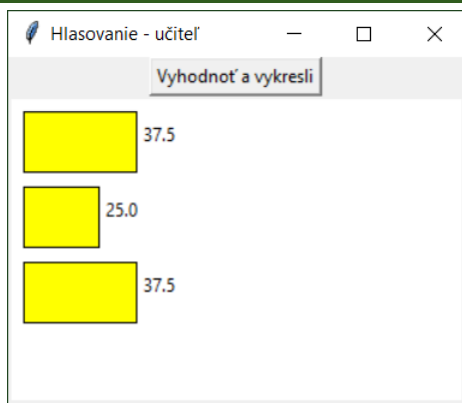
Otvorte a spustite súbor **hlasovanie\_ziak.py**. Urobte niekoľko výberov kliknutím na tlačidlá a medzitým kontrolujte obsah súboru **hlasovanie.txt**.



Uveďte, čo ste zistili z tohto experimentovania. Ako vyzerá súbor **hlasovanie.txt**?

Otvorte program **hlasovanie\_ucitel.py** pre učiteľa, ktorý načítava hlasovanie žiakov zo súboru **hlasovanie.txt**. Naprogramujte v ňom funkciu **vyhodnot()**, ktorá vypočíta percentuálne zastúpenie jednotlivých hlasovacích volieb a vykreslí ich ako vodorovné stĺpcové diagramy na plátne spolu s percentuálnymi hodnotami.

plátne.



Riešenie:

S prvým programom žiaci primárne experimentujú. Mali by prísť na to, že po každom kliknutí na niektoré tlačidlo sa zapíše patričná voľba na koniec textového súboru **hlasovanie.txt**, ktorý v každom riadku obsahuje niektorú z volieb z množiny {'A', 'B', 'C'}.

Je na zvážení učiteľa, či použije program **hlasovanie\_ziak.py** len ako čiernu skrinku na experimentovanie, alebo či aj stručne okomentuje správanie programu – funkciu **zapis\_data()** na pridanie hodnoty na koniec súboru **hlasovanie.txt**.

```
def zapis_data(volba):
    with open('hlasovanie.txt', 'a', encoding='utf-8') as f:
        f.write(f'{volba}\n')
```

```
def hlasujem_a():
    zapis_data('A')
```

Vo funkcii **vyhodnot()** bude potrebné:

- načítať hodnoty zo súboru do zoznamu,
- vytvoriť zoznam z početnosťami jednotlivých volieb,
- pre všetky početnosti volieb vypočítať ich percentuálne zastúpenie a vykresliť ich ako vodorovné stĺpcové diagramy na plátne spolu s percentuálnymi hodnotami.

Riešenie funkcie môže vyzeráť nasledovne:

```
def vyhodnot():
    zoznam = nacitaj_data('hlasovanie.txt')
    volby = ['A', 'B', 'C']
    pocetnosti = [zoznam.count(i) for i in volby]
    platno.delete('all')
    for i in range(len(pocetnosti)):
        percento = 100 * pocetnosti[i] / sum(pocetnosti)
        percento = round(percento, 2)
        platno.create_rectangle(10, 10 + 50 * i, 10 + 2 * percento, 50 + 50
* i, fill='yellow')
        platno.create_text(25 + 2 * percento, 25 + 50 * i, text=percento)
```

Ak majú žiaci problém s určovaním súradníc jednotlivých obdĺžnikov, odporúčame použiť obrázok zo zadania, do ktorého žiaci doplnia konkrétne súradnice pre každý obdĺžnik. Následne to zovšeobecnia na súradnice každého obdĺžnika.

## ROZPRACOVANIE (CCA 9 MIN)

**Úloha 5** je zameraná precvičenie kresliacich metód plátna, prácu so súradnicami a ošetrenie chybného správania sa programu. Ďalším rozšírením úlohy zlepšujúcej použiteľnosť programu môže byť akceptovanie aj malých znakov príkazov a ignorovanie medzier, či iných oddeľovacích znakov.

**Úloha 5** Otvorte súbor **robot.py**. Doprogramujte v ňom dve funkcie **kresli\_mriezku()** a **tah()**, aby program vykresľoval animáciu pohybu robota podľa jednopísmenkových príkazov z množiny {H, D, L, P} uvedených v textovom reťazci vo vstupnom poli. Funkcia **kresli\_mriezku()** má vykresliť štvorcovú mriežku, po uzloch ktorej sa bude pohybovať robot. Funkciu **tah()** pre zadané súradnice robota a zadaný príkaz, vráti nové súradnice robota po vykonaní zadaného príkazu. Funkcia by mala ošetriť prípad, ak v reťazci príkazov bol použitý zadaný iný znak ako z množiny {H, D, L, P} a tiež prípad, ak sa robot dostal mimo štvorcovú mriežku.

Riešenie:

```
def kresli_mriezku():
    for a in range(10, 180, 20):
        platno.create_line(a, 10, a, 170)
        platno.create_line(10, a, 170, a)

def tah(x, y, prikaz):
    if prikaz not in 'HDLP':
        raise ValueError('Na vstupe bol zadaný znak rôzny od znakov {H,D,L,P}.')
    elif prikaz == 'H':
        y = y - 20
    elif prikaz == 'D':
        y = y + 20
    elif prikaz == 'L':
        x = x - 20
    elif prikaz == 'P':
        x = x + 20
    if not (10 <= x <= 170 and 10 <= y <= 170):
        raise ValueError('Robot sa po danom príkaze dostal mimo hraniu plochu.')
    return x, y
```

V nadväznosti na nasledovné tri metodiky zamerané na projektové vyučovanie, odporúčame učiteľovi, aby žiakom zadal za domácu úlohu premyslieť námet projektu, ktorý budú vytvárať na posledných hodinách tohto kurzu programovania v Pythone.

**Úloha 6** Premyslite si námet projektu, ktorý budete vytvárať na ďalších vyučovacích hodinách a zloženie vývojového tímu.

## VYHODNOTENIE (CCA 5 MIN)

Následne požiadame žiakov, aby vypracovali sebahodnotiaci test. Odporúčame žiakom vysvetliť a zdôrazniť, že cieľom je zistiť čo a ako si žiak z obsahu hodiny zapamätal, a nie klasifikácia známku. Pre učiteľa a žiaka zvlášť, je cenná pravdivá informácia o úrovni osvojených poznatkov než



umelo vylepšená. Odporúčame žiakom poskytnúť spätnú väzbu ohľadom správnosti odpovedí. Problematické odpovede môžeme so žiakmi prediskutovať, najlepšie na konci vyučovacej hodiny.

*Sebahodnotiaci test*

1.	<p>Doplňte označené miesta ❶ ❷ ❸ v nasledujúcej funkcii tak, aby vykreslila skupinu úsečiek uvedenú na obrázku.</p>  <pre data-bbox="204 593 829 683">def kresli1():     for i in range(0, 101, 10):         platno.create_line(0, ❶, ❷, ❸)</pre> <p>Riešenie:</p> <p>❶ i ❷ i ❸ 0</p>
----	--

Úloha je zameraná na diagnostiku určovania súradníc bodov v súradnicovom systéme, ich zovšeobecnenie pomocou lineárnej funkcie s použitím kresliacej metódy plátna **create\_oval()**.

## 25 KOMPLEXNÝ PROJEKT – VÝBER PROBLÉMU, ANALÝZA A NÁVRH RIEŠENIA PROBLÉMU

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>pomocou nástrojov na interakciu.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus a podmienený príkaz pri riešení problémov,</li> <li>odchytávať a generovať výnimky,</li> <li>používať jednoduché dátové typy, textové reťazce, zoznamy a jednoduché algoritmy pri riešení problémov zameraných na výpočty, modelovanie a vizualizáciu javov a systémov.</li> </ul>	
<b>Ciele</b>	
<i>Žiakom osvojované vedomosti a zručnosti</i>	<i>Žiakom rozvíjané spôsobilosti</i>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>identifikovať problém, ktorý sa bude riešiť algoritmicky,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia,</li> <li>uvažovať o vlastnostiach vykonávateľa (napr. korytnačka, grafické pero, robot, a pod.),</li> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba zobraziť výstup, realizovať akciu.</li> </ul> Analyzovať a diskutovať možné funkcionality programu. Špecifikovať navrhované správanie sa programu – vytvoriť anotáciu projektu.	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG5) logicky zdôvodniť rozdelenie algoritmu/programu/problému/objektu na menšie časti.</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK2) hierarchická dekompozícia – hierarchicky rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (zobraziť hierarchiu, aj podproblémy rozdeliť na menšie podproblémy).</li> </ul> Hľadanie vzorov: <ul style="list-style-type: none"> <li>(VZO4) zovšeobecniť riešenie podobných problémov na celú triedu, zovšeobecniť na základe konkrétnych prípadov.</li> </ul> Abstrakcia: <ul style="list-style-type: none"> <li>(ABS1) určiť, ktoré detaily/prvky/vlastnosti/vzťahy objektov/problémov/procesov sú v danej situácii podstatné a ktoré môžeme zanedbať.</li> </ul> Ďalšie spôsobilosti <ul style="list-style-type: none"> <li>spôsobilosti tímovej práce (vytvoriť pracovný tím, rozdeliť a riadiť prácu v tíme, spolupracovať v tíme).</li> </ul>
<b>Riešený didaktický problém</b>	
Častou pedagogickou praxou je, že kurzy programovania sú ukončené záverečným projektom. Niektorí učitelia (aby stihli vysvetliť a precvičiť čo najviac učiva) venujú tvorbe projektov minimálny čas počas vyučovania	

a ťažisko ich tvorby presúvajú na čas mimo vyučovania. Rovnako sú veľmi časté individuálne žiacke projekty, ktoré sa učiteľom lepšie klasifikujú ako tímové projekty. Niekedy pri nedostatočnom usmernení učiteľa sa stáva, že sú žiacke projekty vzájomne málo rozmanité, či v praxi málo využiteľné.

V aktuálnom ŠVP sú v charakteristike a cieľoch predmetu informatika uvedené viaceré požiadavky, napr. rozvíjanie schopnosti kooperácie a komunikácie žiakov, rozvíjanie ich tvorivosti a zodpovednosti, realizácia medzipredmetových projektov. Aby sme čo najviac naplnili tieto požiadavky, tvorbe programátorského projektu sa venujeme v troch posledných metodikách, v ktorých dávame priestor učiteľom, aby sa mohli venovať jednotlivým etapám tvorby projektu. Uprednostňujeme tímovú tvorbu záverečného komplexného projektu, ktorú pokladáme za veľmi užitočnú životnú skúsenosť pre žiakov, ktorí ju zúročia v ďalšom svojom štúdiu či v budúcom zamestnaní. Doposiaľ v tomto kurze programovania (v predchádzajúcich 24 metodikách) boli žiaci otestovaní a klasifikovaní štyrikrát za svoj individuálny výkon, preto nepokladáme za problematické, ak žiaci v tíme za svoj tímový projekt budú klasifikovaní rovnakou známku, prípadne len binárne uspel/neuspeš. Naším zámerom je, aby na konci kurzu programovania všetci žiaci zažili úspech a pozitívnu skúsenosť z tímovej tvorby programu využiteľného v praxi. Týmto prístupom sledujeme, aby si žiaci z kurzu odniesli presvedčenie, že programovanie môže byť zaujímavé a využiteľné v praxi. Vytvorené žiacke projekty budú súčasťou žiackeho aj učiteľského portfólia.

<i>Dominantné vyučovacie metódy a formy</i>	<i>Príprava učiteľa a pomôcky</i>
<ul style="list-style-type: none"> <li>• Projektové vyučovanie,</li> <li>• skupinová forma práce žiakov.</li> </ul>	<p>Pre učiteľa:</p> <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania.</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit s riešeniami úloh.</li> </ul> <p>Pre žiaka:</p> <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/25/</b> pracovný priestor pre žiaka.</li> </ul> <p>Použitie digitálnych nástrojov: NUTNÉ</p>
<i>Diagnostika splnenia vzdelávacích cieľov</i>	
Výsledky žiackych riešení úloh z pracovného listu, vyplnená anotácia tímového projektu.	

## Úvod

Uvedená metodika je v poradí 25. metodikou zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Posledné tri metodiky nášho kurzu programovania sú zamerané na projektové vyučovanie, v rámci ktorého žiaci využijú svoje doterajšie programovacie vedomosti a zručnosti a vytvoria v praxi využiteľný softvérový produkt. V nich sa budeme postupne venovať jednotlivým etapám tvorby komplexného projektu:

- Zostavenie tímu. Výber témy projektu. Analýza a návrh riešenia problému. Tvorba anotácie projektu. (1 hodina)
- Implementácia riešenia problému. (1 hodina)
- Finalizácia projektu. Sebahodnotenie výsledného projektu. Prezentácia projektu s diskusiou. Hodnotenie projektu učiteľom a spolužiakmi. (2 hodiny)

Žiaci majú k dispozícii šablónu anotácie projektu, sebahodnotiacu rubriku projektu a ďalšie nástroje formatívneho hodnotenia počas vývoja tímového projektu.

## PRIEBEH VÝUČBY

Vyučovacia hodina pozostáva z nasledovných častí:

- **Zhrnutie doterajších vedomostí a zručností z programovania (cca 6 minút)**
- **Diskusia k typom projektov a návrhom konkrétnych projektov (cca 8 minút)**
- **Etapy tvorby projektu (cca 1 minúta)**
- **Požiadavky na komplexný projekt, (seba)hodnotiaca rubrika (cca 5 minút)**
- **Samostatná práca v tímoch na tvorbe anotácie projektu (cca 20 minút)**

## ZHRNUTIE DOTERAJŠÍCH VEDOMOSTÍ A ZRUČNOSTÍ Z PROGRAMOVANIA (CCA 6 MIN)

Na tvorbu projektu môžeme metaforicky pozerať ako na stavbu domu. Jednou z dôležitých záležitostí je prediskutovať, aký stavebný materiál môžeme použiť na jeho stavbu. Pri programátorských projektoch sú „stavebnými kameňmi“ programátorské vedomosti a zručnosti žiakov.

**Úloha 1** *Prediskutujte v dvojiciach a zhrňte do tabuľky, čo ste sa naučili v predchádzajúcich hodinách programovania v Pythone.*

Typy premenných	Príkazy (len typovo)	Algoritmy	Oblasti problémov

*Riešenie:*

*Po diskusiách v dvojiciach by sa mali výsledky zosumarizovať frontálne v celej triede.*

*Alternatívou tabuľky s prvkami učiva môže byť pojmová mapa.*

*Pri riešení tejto úlohy môžeme využiť niektoré z online kolaboratívnych nástrojov, napr. (<https://padlet.com/>, <https://mind42.com/>, <https://miro.com/>, <https://docs.google.com/>)*

Typy premenných	Príkazy (len typovo)	Algoritmy	Oblasti problémov
int, float (bool) str list	funkcie s parametrami a výsledkom cyklus (for, while) vetvenie (if, elif, else) výnimky (odchytávanie, generovanie) modulu turtle modulu tkinter modulu math modulu random	s reťazcami so zoznamami	Výpočtové – hľadanie prvočísel ...  Kresliace – vykresľovanie zložených obrázkov ...  Práca s textami – šifrovanie textov ...

## DISKUSIA K TYPOM PROJEKTOV A NÁVRHOM KONKRÉTNÝCH PROJEKTOV (CCA 8 MIN)

Podobne ako z kvalitného stavebného materiálu sa dá postaviť nefunkčný dom, aj pri programovaní ovládanie základných prostriedkov programovacieho jazyka a znalosť základných algoritmov nezaručuje, že vytvoríme v praxi využiteľný program. Pri tvorbe v praxi použiteľného projektu je vhodné zrealizovať brainstorming a následne prediskutovať možné typy užitočných programov. Tieto námety budú výborným východiskom pre výber témy vlastných komplexných projektov. Pre lepšiu úspešnosť riešenia tejto úlohy žiaci dostali na konci predchádzajúcej hodiny za domácu úlohu premyslieť námet projektu, ktorý budú vytvárať na tejto a ďalších hodinách kurzu programovania v Pythone

**Úloha 2** Uvedte aké praktické problémy by sa mohli riešiť pomocou programovania?

.....

.....

.....

**Riešenie:**

Pri tvorbe návrhov programov na riešenie problémov si predstavte **pre koho** chcete vytvoriť program (seba či spolužiakov, žiakov nižších ročníkov, rodičov, ľudí so zdravotným znevýhodnením, športovcov atď.) Aký **účel** bude plniť tento program (edukačná pomôcka, hra, pomocný nástroj atď)? S akým **typom dát** bude pracovať Váš program, bude sa v ňom viac počítať, pracovať s textami, či kresliť obrázky?

Pre inšpiráciu uvádzame pre učiteľov niekoľko príkladov námetov komplexných projektov:

- **Nástroje na spracovanie a vizualizáciu dát:**
  - Vizualizácia a výpočet hodnôt zaznamenaných z dataloggera (napr. z krokomera, cyklopočítača, senzora osvetlenia).
  - Spracovanie údajov zo zdroja otvorených dát (priebehov hodnôt v čase, napr. z covid-19 pandémie, kurzov mien, teplôt, geografických údajov).
  - Zostavovanie projektových tímov z dvojíc/trojíc žiakov na základe sociometrických údajov (kto s kým chce/nechce spolupracovať).
- **Edukačné pomôcky:**
  - Zobrazenie grafu funkcie zadanej matematickým predpisom resp. dátovým súborom.
  - Riešenie trojuholníka (vstup:  $a, b, c$ ; výstup:  $\alpha, \beta, \gamma, P, O, R, r$ ) – výpočet aj zobrazenie.
  - Oprava diktátu – na vstupe súbory s textami učiteľa, žiaka<sub>1</sub>, žiaka<sub>2</sub> .... žiaka<sub>N</sub> – na výstupe súbor s opraveným diktátom (s prípadnými vylepšeniami napr. ignorovanie viacnásobných medzier, písmeno navyše/menej, či prehodené písmená).
  - Slepá mapa – hľadanie pozície/polohy mesta/časti ľudského tela.
  - Generovanie úloh a testovanie násobenia či sčítania čísel/číselných prevodov.
- **Hry:**
  - Hra hádaj prirodzené číslo so zadaného intervalu s pomocným výpisom aktuálnych hraníc hľadaného čísla.
  - Strategická hra s počítačom napr. hra NIMM (odoberanie 1-4 zápaliek z hromady 21 zápaliek).
- **Simulátory javov, zariadení:**
  - Robotický vysávač (naplánovanie trasy upratovania vysávača pri zadaných prekážkach v miestnosti so zadanými rozmermi).
  - Inteligentná domácnosť – simulácia chodu viacerých spotrebičov s možnosťou ich ovládania (naprogramovaným pravidlom, resp. okamžitým priamym zásahom).
- **Umenie:**
  - Upravovanie a generovanie textov piesní (napr. Sedí mucha na stene – zmena ľubovoľnej hlásky).
  - Generátor príbehov (výhovoriek) využívajúci pripravené dáta a zohľadňujúci vlastné pravidlá.
  - Grafický editor s vlastnými (aj netradičnými) grafickými funkciami – generátor zaujímavých obrázkov.

## ETAPY TVORBY PROJEKTU (CCA 1 MIN)

Pri tvorbe projektu budeme postupovať v nasledovných etapách:

- **Zostavenie tímu. Výber témy projektu. Analýza a návrh riešenia problému** (jeho možné vstupy a výstupy). **Tvorba anotácie projektu.** (1 hodina)
- **Implementácia** riešenia problému. (1 hodina)
- **Finalizácia projektu** (testovanie funkčnosti programu, kompletizácia dokumentačných reťazcov funkcií, ošetrovanie chýb). **Sebahodnotenie výsledného projektu** (jeho úroveň, zoznam jeho možných vylepšení, úroveň svojej práce a práce ostatných členov tímu). (1 hodina)
- **Prezentácia projektov s diskusiou. Hodnotenie projektov učiteľom a spolužiakmi.** Založenie projektov do žiackeho (aj učiteľského) portfólia. (1 hodina)

## POŽIADAVKY NA KOMPLEXNÝ PROJEKT. (SEBA)HODNOTIACA RUBRIKA (CCA 5 MIN)

Vzhľadom na rôznorodosť typov projektov, veľkosť tímov, výkonnostné rozdiely študentov navrhujeme vopred zverejniť žiakom hodnotiacu rubriku projektu, ktorú môžu použiť aj na sebahodnotenie projektu v priebehu aj na konci jeho tvorby. Aj keď dôraz kladieme na využiteľnosť projektu v praxi, ošetrovanie chýb počas výpočtu a dôsledné okomentovanie programového kódu, výsledná klasifikácia projektu bude závislá od súčtu bodov dosiahnutých za jednotlivé položky rubriky. Prevod dosiahnutého skóre na známky necháme na samotných učiteľov (napr. hodnotenie výborný pri dosiahnutí hodnoty aspoň 35 bodov). Pri binárnom hodnotení uspel/neuspel by hranicou úspešnosti mohlo byť dosiahnutie určitej hranice, napr. 20 bodov.

Mená autorov projektu:			Trieda:	Dátum:	
Úroveň Položka	Úroveň 1 (1 bod)	Úroveň 2 (2 body)	Úroveň 3 (3 body)	Úroveň 4 (4 body)	Spolu
<b>1 Náročnosť riešenia algoritmického problému, pri riešení žiak použil</b>	nanajvýš jednoduchý sekvenčný algoritmus	algoritmus obsahujúci cykly alebo algoritmus obsahujúci vetvenie	algoritmus obsahujúci cykly a vetvenie	algoritmus obsahujúci vzájomné vnorené cykly a vetvenie	
<b>2 Dátové štruktúry použité pri riešení algoritmického problému</b>	žiadne alebo len jednoduché typy int alebo float alebo boolean	štruktúrovaný typ str	štruktúrovaný typ list	kombinácia viacerých štruktúrovaných typov	
<b>3 Ošetrovanie chybových stavov</b>	testujú sa nanajvýš požiadavky na vstup	testujú sa operácie s dátami	testujú sa požiadavky na vstup a operácie s dátami		
<b>4 Reakcia na chybové stavy</b>	v prípade problematického stavu nie je používateľ informovaný alebo je informovaný len systémovou hláškou	v prípade problematického stavu je používateľ informovaný relevantnou hláškou o problematickom stave (napr. aj výpisom do konzoly)	v prípade problematického stavu je používateľ informovaný relevantnou hláškou o problematickom stave prostredníctvom grafického rozhrania		
<b>5 Vstup dát</b>	program nepoužíva vstup alebo len vstup z konzoly	vstupné textové okienko/tlačidlo grafickej aplikácie, myš v grafickej ploche	vstupné dáta zo súboru	kombinácia vstupu zo súboru s ďalším interaktívnym vstupom	
<b>6 Výstup dát</b>	program nepoužíva výstup alebo len výstup do konzoly	výstupné textové okienko/ výstup do grafickej plochy	výstupné dáta do súboru	kombinácia výstupu do súboru v kombinácii s ďalším typom výstupu	
<b>7 Dekompozícia</b>	žiadna alebo len jedna	vhodne navrhnutá	správne dekomponovaný	správne dekomponovaný	



<b>problému</b>	funkcia riešiaci celý problém	funkcia s parametrom alebo funkcia s návratovou hodnotou	problém, vhodne navrhnutá funkcie s parametrami a s návratovými hodnotami	problém, jednotlivé časti problému sa riešia v samostatných funkciách, funkcie sa vzájomne volajú a odovzdávajú si výstupné hodnoty	
<b>8 Kód programu</b>	identifikátory sú nejasné, z ich názvu nie je možné dedukovať ich význam	identifikátory (názvy premenných a funkcií) sú popisné, z ich názvu je možné dedukovať význam ich obsahu, resp. použitia	funkcie obsahujú dokumentačné reťazce popisujúce výsledok funkcie, vstupné a výstupné hodnoty	dokumentačné reťazce sú úplné, vstupy a výstupy sú definované podľa štandardu	
<b>9 Používateľské rozhranie</b>	grafické rozhranie nie je k dispozícii alebo je neprehľadné	grafické rozhranie je prehľadné	je zrejmé, ktoré polia sú vstupné a ktoré výstupné, obsah polí je popísaný (napr. pomocou Label)	výstupné polia sú chránené voči používateľským vstupom	
<b>10 Prezentácia projektu žiakom</b>	žiak vie nanajvýš „prerozprávať“ časti kódu	žiak vie popísať logiku algoritmu a identifikovať jednotlivé časti programu (napr. táto funkcia robí to a to)	žiak vie zdôvodniť použitý algoritmus, postup, žiak vie vysvetliť jednotlivé časti programu	žiak vie popísať hraničné (extrémne, krajné) prípady a ako na ne v programe reaguje, príp. v ktorých situáciách program nebude pracovať správne	
<b>11 Využitie projektu v praxi</b>	použiteľný v praxi po veľkých úpravách	použiteľný v praxi po menších úpravách	použiteľný v praxi bez potrebných úprav		
<b>Spolu</b>					

**SAMOSTATNÁ PRÁCA V TÍMOCH NA TVORBE ANOTÁCIE PROJEKTU (CCA 20 MIN)**

V tejto etape by mali žiaci vo svojich tímoch pre vybranú tému projektu premyslieť predstavu o projekte a zostaviť k nemu anotáciu. Každý tím by mal mať svoj spoločný zdieľaný priestor, do ktorého by mal mať prístup aj učiteľ. Každý tímový projekt treba ponímať ako zákazku s termínom jej dodania.

**Úloha 3** Zostavte dvoj až trojčlenné projektové tímy. Uvedte **mená** a **e-maily** členov tímu a doplňte dohodnutého **hlavného zodpovedného člena tímu**:

.....

.....

.....

Prediskutujte a vyberte **tému** vášho tímového projektu:

.....

Napište stručnú **anotáciu** – akého typu je projekt, pre koho je určený, aký je jeho význam/úžitok pre prax, čo konkrétne bude program robiť:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Uložte si túto anotáciu projektu a pošlite ju spolupracovníkom v tíme a tiež svojmu učiteľovi.

## 26 KOMPLEXNÝ PROJEKT – IMPLEMENTÁCIA RIEŠENIA PROBLÉMU

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
Algoritmické riešenie problémov: <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupností príkazov,</li> <li>pomocou nástrojov na interakciu,</li> <li>pomocou premenných,</li> <li>pomocou cyklov,</li> <li>pomocou vetvenia,</li> <li>interpretácia zápisu riešenia,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 1 vyučovací hodina
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus a podmienený príkaz pri riešení problémov,</li> <li>odchytať a generovať výnimky,</li> <li>používať jednoduché dátové typy, textové reťazce, zoznamy a jednoduché algoritmy pri riešení problémov zameraných na výpočty, modelovanie a vizualizáciu javov a systémov.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Analýza problému:</b> <ul style="list-style-type: none"> <li>identifikovať vstupné informácie zo zadania úlohy,</li> <li>popisovať očakávané výstupy, výsledky, akcie,</li> <li>identifikovať problém, ktorý sa bude riešiť algoritmicky,</li> <li>formulovať a neformálne (prirodzeným jazykom) vyjadriť ideu riešenia,</li> <li>uvažovať o vlastnostiach vykonávateľa (napr. korytnačka, grafické pero, robot, a pod.),</li> <li>plánovať riešenie úlohy ako postupnosť príkazov vetvenia a opakovania.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>rozpoznávať situácie, kedy treba získať vstup,</li> <li>identifikovať vlastnosti vstupnej informácie (obmedzenia, rozsah, formát),</li> <li>rozpoznávať situácie, kedy treba zobrazíť výstup, realizovať akciu,</li> <li>zapisovať algoritmus, ktorý reaguje na vstup.</li> </ul> Spresniť špecifikáciu problému.	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG5) logicky zdôvodniť rozdelenie algoritmu/programu/problému/objektu na menšie časti.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém,</li> <li>(ALG5) využívať existujúce algoritmy v návrhu vlastných algoritmov.</li> </ul> Dekompozícia: <ul style="list-style-type: none"> <li>(DEK2) hierarchická dekompozícia – hierarchicky rozdeliť problémy na menšie časti tak, aby sa dali využiť pre dosiahnutie cieľa (zobraziť hierarchiu, aj podproblémy rozdeliť na menšie podproblémy, rozdeliť prácu pre členov hierarchického tímu ktorí delegujú prácu na podriadených).</li> </ul> Abstrakcia: <ul style="list-style-type: none"> <li>(ABS1) určiť, ktoré detaily/prvky/vlastnosti/vzťahy objektov/problémov/procesov sú v danej situácii podstatné a ktoré môžeme</li> </ul>

Dekomponovať problém na podproblémy. Navrhnuť funkcionality programu a maketu grafického používateľského rozhrania. Implementovať riešenia vybraných funkcionalít v programovacom jazyku Python.	zanedbať.  Ďalšie spôsobilosti <ul style="list-style-type: none"> <li>• spôsobilosti tímovej práce (riadiť prácu v tíme, spolupracovať v tíme).</li> </ul>
<b>Riešený didaktický problém</b>	
Pri realizácii projektového vyučovania nie je vhodné, ak učiteľ veľmi zasahuje do rozhodovania sa žiakov v jednotlivých etapách tvorby programu. Rovnako nie je vhodné, ak nechá žiakov pracovať na projekte bez patričných usmernení a odporúčaní. Potom sa stáva, že žiaci neurobia dôslednú špecifikáciu problému a hneď sa pustia do vytvárania programového kódu, ktorý neustále zbytočne prerábajú. V predloženej metodike poskytujeme učiteľovi a žiakom rámcové informácie pre jednotlivé etapy tvorby projektu, obzvlášť pre spresnenie špecifikácie problému, jeho dekompozíciu na podproblémy, navrhnutie funkcionalít programu a grafického používateľského rozhrania.	
<b>Dominantné vyučovacie metódy a formy</b>	<b>Príprava učiteľa a pomôcky</b>
<ul style="list-style-type: none"> <li>• Projektové vyučovanie,</li> <li>• skupinová forma práce žiakov.</li> </ul>	Pre učiteľa: <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania.</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit s riešeniami úloh.</li> </ul> Pre žiaka: <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/26/</b> pracovný priestor pre žiaka.</li> </ul> Použitie digitálnych nástrojov: NUTNÉ
<b>Diagnostika splnenia vzdelávacích cieľov</b>	
Výsledky žiackych riešení úloh z pracovného listu, vyplnená maketa s prvkami grafického používateľského rozhrania.	

## Úvod

Uvedená metodika je v poradí 26. metodikou zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Na predchádzajúcej hodine žiaci zostavili projektové tímy a vytvorili tému projektu a jeho anotáciu. Na tejto hodine žiaci spresnia špecifikáciu problému, navrhnu funkcionality jednotlivých podproblémov a návrh grafického používateľského rozhrania, začnú programovať funkcionality jednotlivých podproblémov. Až po ich naprogramovaní odporúčame, aby žiaci programovali grafické používateľské rozhranie. Žiaci pracujú samostatne v tímoch, učiteľ len monitoruje a usmerňuje prácu projektových tímov. Podľa záujmu žiakov im vie poskytnúť odbornú radu napr. k novým prvkom grafického používateľského rozhrania (Scale, Text, Checkbutton, Radiobutton, atď.)

## PRIEBEH VÝUČBY

Vyučovacia hodina pozostáva z nasledovných častí:

- **Spresnenie špecifikácie problému a návrh funkcionalít programu (cca 15 minút)**
- **Návrh grafického používateľského rozhrania (cca 5 minút)**
- **Implementácia riešenia problému (cca 20 minút)**

### SPRESNENIE ŠPECIFIKÁCIE PROBLÉMU A NÁVRH FUNKCIONALÍT PROGRAMU (CCA 15 MIN)

V **úlohe 1** žiaci urobia detailnejšiu analýzu problému, určia vstupy, výstupy a vzťahy medzi nimi, rozdelia problém na podproblémy a pre každý podproblém navrhnu jeho funkcionality.

**Úloha 1** Formulujte problém, ktorého riešenie chcete naprogramovať:

.....

Urobte detailnejšiu analýzu problému. Určte, čo sú vstupy a výstupy:

Vstupy

.....

Výstupy .....

Uvedte vzťahy medzi vstupmi a výstupmi:

.....

.....

Rozdeľte problém do podproblémov. Pre každý podproblém navrhnite jeho funkcionality:

.....

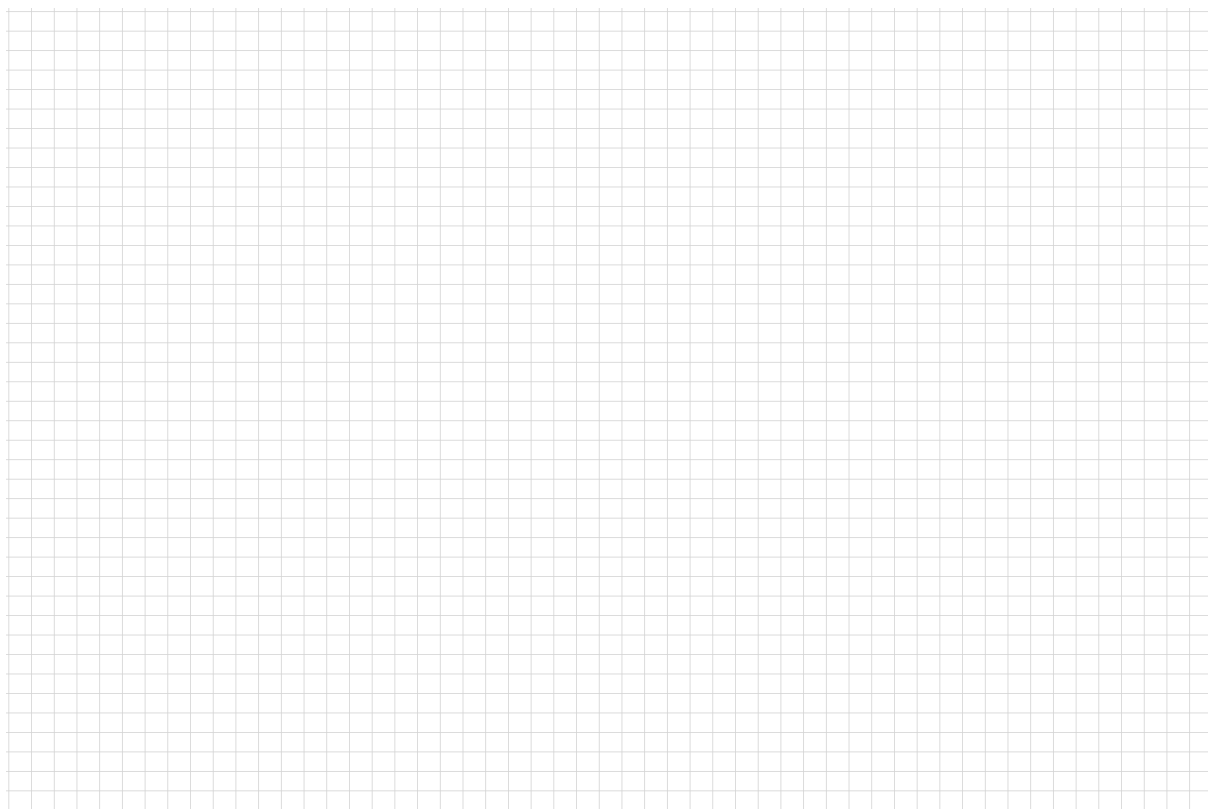
.....

.....

## NÁVRH GRAFICKÉHO POUŽÍVATEĽSKÉHO ROZHRAINIA (CCA 5 MIN)

V nadväznosti na predchádzajúcu úlohu majú žiaci v **úlohe 2** navrhnuť grafické používateľské rozhranie ako maketu s prvkami grafického používateľského rozhrania umiestnenými v mriežke so stručným opisom ich použitia. Učiteľ vie usmerniť žiakov, aby uvažovali, ktoré grafické prvky budú slúžiť pre vstup, medzivýsledky a ktoré pre výstup, a tiež, ktoré prvky sú na nastavenie parametrov a ktoré na vykonanie vybranej funkcionality.

**Úloha 2** Navrhnite grafické používateľské rozhranie – vytvorte maketu s prvkami grafického používateľského rozhrania umiestnenými v mriežke so stručným opisom ich použitia.



## IMPLEMENTÁCIA RIEŠENIA PROBLÉMU (CCA 20 MIN)

V tejto etape realizácie projektu (v **úlohe 3**) žiaci začínajú implementovať v jazyku Python funkcie pre riešenie jednotlivých podproblémov. Pri tvorbe funkcií odporúčame, aby žiaci využili svoje návrhy funkcionalít (z úlohy 1) v dokumentačných reťazcoch a až následne dotvárali programový kód funkcií.

**Úloha 3** *Pre jednotlivé podproblémy navrhnete funkcie, ktoré ich budú riešiť. Pri každej funkcii určte vstupy, výstupy a vzťah medzi nimi. Odporúčame začať tvorbu funkcie napísaním dokumentačného reťazca a až následne dotvoriť programový kód funkcie.*

Poznámka: Odporúčame učiteľom pripomínať žiakom, že každý člen tímu je dôležitý a zodpovedajúci za svoj diel práce. Rovnako by mali žiaci prípadné zmeny špecifikácie problému konzultovať s učiteľom ako zadávateľom zákazky, aby sa o nich nedozvedel až pri záverečnej prezentácii projektu.

## 27 KOMPLEXNÝ PROJEKT – FINALIZÁCIA PROJEKTU + PREZENTÁCIA A DISKUSIA

<i>Tematický celok / Téma</i>	<i>Stupeň školy / Odporúčaný ročník / Rozsah</i>
<b>Algoritmické riešenie problémov:</b> <ul style="list-style-type: none"> <li>analýza problému,</li> <li>jazyk na zápis riešenia,</li> <li>pomocou postupností príkazov,</li> <li>pomocou nástrojov na interakciu,</li> <li>pomocou premenných,</li> <li>pomocou cyklov,</li> <li>pomocou vetvenia,</li> <li>interpretácia zápisu riešenia,</li> <li>hľadanie a opravovanie chýb.</li> </ul>	SŠ / 2. ročník / 2 vyučovacie hodiny
<b>Požiadavky na vstupné vedomosti a zručnosti</b>	
<ul style="list-style-type: none"> <li>vytvárať a vyhodnocovať aritmetické výrazy s premennou,</li> <li>vytvárať a používať vlastné funkcie s parametrami a s návratovou hodnotou,</li> <li>používať cyklus a podmienený príkaz pri riešení problémov,</li> <li>odchytať a generovať výnimky,</li> <li>používať jednoduché dátové typy, textové reťazce, zoznamy a jednoduché algoritmy pri riešení problémov zameraných na výpočty, modelovanie a vizualizáciu javov a systémov.</li> </ul>	
<b>Ciele</b>	
<b>Žiakom osvojované vedomosti a zručnosti</b>	<b>Žiakom rozvíjané spôsobilosti</b>
<b>Pomocou postupnosti príkazov:</b> <ul style="list-style-type: none"> <li>aplikovať pravidlá, konštrukcie jazyka pre zostavenie postupnosti príkazov.</li> </ul> <b>Pomocou nástrojov na interakciu:</b> <ul style="list-style-type: none"> <li>zapisovať algoritmus, ktorý reaguje na vstup.</li> </ul> <b>Pomocou premenných:</b> <ul style="list-style-type: none"> <li>zovšeobecniť riešenie tak, aby fungovalo nielen s konštantami.</li> </ul> <b>Pomocou cyklov. Pomocou vetvenia:</b> <ul style="list-style-type: none"> <li>riešiť problémy, v ktorých sa kombinujú cykly a vetvenia.</li> </ul> <b>Interpretácia zápisu riešenia:</b> <ul style="list-style-type: none"> <li>uvažovať o rôznych riešeniach, navrhovať vylepšenie.</li> </ul> <b>Hľadanie a opravovanie chýb:</b> <ul style="list-style-type: none"> <li>hľadať chybu vo vlastnom, nesprávne pracujúcom programe a opraviť ju,</li> <li>zisťovať, pre aké vstupy, v ktorých prípadoch, situáciách program zle pracuje,</li> <li>posudzovať a overovať správnosť riešenia (svojho aj cudzieho).</li> </ul> Implementovať riešenia všetkých funkcionalít (podproblémov) v programovacom jazyku Python.	Koncepty informatického myslenia  Logika: <ul style="list-style-type: none"> <li>(LOG5) logicky zdôvodniť rozdelenie algoritmu/programu/problému/objektu na menšie časti,</li> <li>(LOG6) logicky zdôvodniť zmenu algoritmu/programu.</li> </ul> Algoritmy: <ul style="list-style-type: none"> <li>(ALG3) vytvárať vlastné algoritmy riešiace problém,</li> <li>(ALG7) vylepšovať existujúce algoritmy (zlepšenie efektívnosti, rozšírenie algoritmu na väčšiu množinu vstupov, rozšírenie funkcionality).</li> </ul> Vyhodnotenie: <ul style="list-style-type: none"> <li>(VYH3) posúdiť kvalitu/správnosť/efektívnosť/vhodnosť postupu na základe vybraných/definovaných kritérií (napr. posúdiť efektívnosť algoritmov, správnosť dekompozície, presnosť a úplnosť algoritmu/programu, testovať program).</li> </ul>





Skompletizovať celý projekt – pripraviť vhodné testovacie dáta, pomocou výnimiek ošetriť chybné správanie programu, otestovať funkčnosť jednotlivých častí programu a napokon celého programu.	Ďalšie spôsobilosti <ul style="list-style-type: none"> <li>• spôsobilosti tímovej práce (riadiť prácu v tíme, spolupracovať v tíme),</li> <li>• prezentovať,</li> <li>• argumentovať,</li> <li>• monitorovať a regulovať vlastné učenie sa.</li> </ul>
<b>Riešený didaktický problém</b>	
<p>Niektorí učitelia na konci projektového vyučovania zozbierajú žiacke projekty, ktoré následne ohodnotia a každému autorskému tímu poskytnú spätnú väzbu. Pri takomto vyučovaní žiaci nemajú predstavu o tom ako vyzerajú a čo riešia projekty ostatných spolužiakov, nemajú možnosť prediskutovať funkcionality projektov spolužiakov, ani reflektovať priebeh a výsledky svojho projektu.</p> <p>V našich metodikách odporúčame vyhradiť aspoň 1 vyučovaciu hodinu na prezentáciu žiackych (tímových) komplexných projektov s diskusiou a záverečnou sebareflexiou žiakov k priebehu a výsledkom svojho (tímového) komplexného projektu. Prezentácie projektov s diskusiami by mali byť príjemnou oslavou žiackej kreativity a výsledné programy odrazom ich vedomostí a zručností z programovania. Vyplnené sebahodnotiace rubriky a sebahodnotiace dotazníky by mali rozvíjať metakogníciu žiakov a učiteľovi poskytnúť dôležité informácie pre vyhodnotenie celého základného kurzu programovania v jazyku Python. Prezentácie (tímových) komplexných projektov s diskusiami umožnia žiakom rozvíjať ich komunikačné spôsobilosti (vyjadrovanie v odbornom jazyku, argumentovanie), ich kritické myslenie (uvedomením si silných a slabých stránok svojho projektu aj projektov spolužiakov), časovú disciplínu atď.</p>	
<b>Dominantné vyučovacie metódy a formy</b>	<b>Príprava učiteľa a pomôcky</b>
<ul style="list-style-type: none"> <li>• Projektové vyučovanie,</li> <li>• skupinová forma práce žiakov.</li> </ul>	Pre učiteľa: <ul style="list-style-type: none"> <li>• <b>ucitel/programovanie_v_pythone.pdf</b> metodika vyučovania,</li> <li>• <b>ucitel/pracovny_zosit_riesene_ulohy.docx</b> pracovný zošit s riešeniami úloh.</li> </ul> Pre žiaka: <ul style="list-style-type: none"> <li>• <b>ziak/pracovny_zosit.docx</b> pracovný zošit,</li> <li>• <b>ziak/pracovne_subory/27/</b> pracovný priestor pre žiaka.</li> </ul> Použitie digitálnych nástrojov: NUTNÉ
<b>Diagnostika splnenia vzdelávacích cieľov</b>	
Prezentácia komplexného projektu s diskusiou, sebahodnotiaca rubrika, sebahodnotiaci dotazník.	

## Úvod

Uvedená metodika je poslednou zo série 27 metodík, ktoré sú určené pre základný kurz programovania. Na predchádzajúcej hodine žiaci začali implementovať riešenia jednotlivých podproblémov. Na prvej hodine tejto metodiky žiaci dokončia implementáciu všetkých podproblémov, ktoré otestujú na pripravených testovacích dátach a ošetrí odolnosť programu voči chybám pomocou výnimiek. Za domácu úlohu si žiaci pripraví stručnú prezentáciu k vytvorenému projektu a vyplní sebahodnotiacu rubriku. Na druhej hodine žiaci prezentujú svoj (tímový) komplexný projekt a diskutujú so spolužiakmi a učiteľom pozitíva projektu a odporúčania k jeho vylepšeniu či rozšíreniu. Na konci hodiny žiaci vyplní sebahodnotiaci dotazník.

## PRIEBEH VÝUČBY

Prvá hodina:

- **Kompletizácia programových kódov projektu (cca 40 minút)**
- **Inštrukcie k prezentácii komplexných projektov (cca 5 minút)**

Druhá hodina:

- **Prezentácie (tímových) komplexných projektov s diskusiou (cca 40 minút)**
- **Sebahodnotenie práce žiakov pomocou dotazníka (cca 5 minút)**

## KOMPLETIZÁCIA PROGRAMOVÝCH KÓDOV PROJEKTU (CCA 40 MIN)

Na prvej hodine žiaci pokračujú v etape realizácie projektu, ktorá sa začala na predchádzajúcej hodine. Postupne implementujú v jazyku Python riešenia všetkých stanovených podproblémov. Pri tvorbe jednotlivých funkcií odporúčame učiteľom usmerňovať žiakov, aby začali s písaním dokumentačných reťazcov, ktoré popisujú funkcionality funkcie s jej vstupmi a výstupmi. Pri tvorbe funkcie by mali žiaci zvažovať možné kritické prípady, kedy funkcia môže dať nesprávne výsledky alebo sa skončí s behovou chybou. Pomocou výnimiek ošetrujú tieto kritické prípady. Tvorbu funkcie by mali žiaci ukončiť jej testovaním na rôznych dátach. Učiteľ v tejto etape monitoruje a usmerňuje prácu jednotlivých projektových tímov. Podľa potreby a záujmu žiakov im vie poskytnúť odbornú radu.

## INŠTRUKCIE K PREZENTÁCII KOMPLEXNÝCH PROJEKTOV (CCA 5 MINÚT)

Učiteľ inštruuje žiakov k priebehu záverečnej hodiny programovania v Pythone zameranej na prezentáciu (tímových) komplexných projektov. Prezentácia každého projektu by mala obsahovať stručné informácie o jej zameraní (komu je určená, jej význam pre cieľovú skupinu) a hlavne ukážku jej jednotlivých funkcionalít. Po prezentácii nasleduje diskusia učiteľa a spolužiakov k pozitívam prezentovaného projektu a prípadných námetov na jej vylepšenie či rozšírenie. Pri počte maximálne 6 projektov a časovej dotácii 42 minút na každý projekt pripadne 7 minút, z toho cca 5 minút na prezentáciu a 2 minúty na diskusiu projektu. Je na rozhodnutí samotného učiteľa, do akej miery

bude akceptovať tieto odporúčania, alebo posilní tieto časové dotácie (znížením počtu projektov, či doplnením o ďalšiu vyučovaciu hodinu).

Pri prezentácii svojho projektu by mali žiaci uviesť anotáciu projektu, ukázať použitie programu, zhodnotiť, čo sa podarilo dosiahnuť a čo nie a prečo. Aj keď čas 5 minút na prezentovanie trojhodinového projektu sa zdá byť veľmi krátkym, núti to žiakov sa vyjadrovať stručne k podstate veci a štruktúrovane, bez zbytočných balastov. Po prezentácii žiackeho projektu je dôležité zhodnotiť pozitíva a slabé stránky projektu podľa poznámok k žiakom vyplnenej sebahodnotiacej rubriky a prípadne stručne prediskutovať so žiakmi možné vylepšenia projektu.

Odporúčame, aby žiaci deň-dva pred hodinou s prezentáciami projektov elektronicky odovzdali učiteľovi:

- finálnu verziu programu,
- prípadnú prezentáciu či videozáznam funkcionality programu,
- finálnu verziu anotácie projektu,
- vyplnenú sebahodnotiacu rubriku projektu.

Pre úspešnú realizáciu prezentácie všetkých projektov je dôležité usmerniť žiakov, aby si premysleli a nacvičili prezentáciu projektu tak, aby sa zmestili do učiteľom určeného času prezentácie. Rovnako je dôležité, aby sa na poslednej hodine do prezentácie a diskusie zapájali všetci členovia tímu.

Prezentácie (tímových) komplexných projektov na poslednej hodine odporúčame realizovať s patričnou dôležitosťou ako obhajobu zákazky pred investormi spolu s pozvanými hosťami (napr. triednym učiteľom, rodičmi, vedením školy atď.) Je vhodné nechať žiakom výber formy ich prezentácie.

Mená autorov projektu:			Trieda:	Dátum:	
Úroveň Položka	Úroveň 1 (1 bod)	Úroveň 2 (2 body)	Úroveň 3 (3 body)	Úroveň 4 (4 body)	Spolu
<b>1 Náročnosť riešenia algoritmického problému, pri riešení žiak použil</b>	nanajvyš jednoduchý sekvenčný algoritmus	algoritmus obsahujúci cykly alebo algoritmus obsahujúci vetvenie	algoritmus obsahujúci cykly a vetvenie	algoritmus obsahujúci vzájomné vnorené cykly a vetvenie	
<b>2 Dátové štruktúry použité pri riešení algoritmického problému</b>	žiadne alebo len jednoduché typy int alebo float alebo boolean	štruktúrovaný typ str	štruktúrovaný typ list	kombinácia viacerých štruktúrovaných typov	
<b>3 Ošetrovanie chybových stavov</b>	testujú sa nanajvyš požiadavky na vstup	testujú sa operácie s dátami	testujú sa požiadavky na vstup a operácie s dátami		
<b>4 Reakcia na chybové stavy</b>	v prípade problematického stavu nie je používateľ informovaný alebo je informovaný len systémovou hláškou	v prípade problematického stavu je používateľ informovaný relevantnou hláškou o problematickom stave (napr. aj výpisom do konzoly)	v prípade problematického stavu je používateľ informovaný relevantnou hláškou o problematickom stave prostredníctvom grafického rozhrania		
<b>5 Vstup dát</b>	program nepoužíva vstup alebo len vstup z konzoly	vstupné textové okienko/tlačidlo grafickej aplikácie, myš v grafickej ploche	vstupné dáta zo súboru	kombinácia vstupu zo súboru s ďalším interaktívnym vstupom	
<b>6 Výstup dát</b>	program nepoužíva výstup alebo len výstup do konzoly	výstupné textové okienko/ výstup do grafickej plochy	výstupné dáta do súboru	kombinácia výstupu do súboru v kombinácii s ďalším typom výstupu	
<b>7 Dekompozícia</b>	žiadna alebo len jedna	vhodne navrhnutá	správne dekomponovaný	správne dekomponovaný	

<b>problému</b>	funkcia riešiaci celý problém	funkcia s parametrom alebo funkcia s návratovou hodnotou	problém, vhodne navrhnutá funkcie s parametrami a s návratovými hodnotami	problém, jednotlivé časti problému sa riešia v samostatných funkciách, funkcie sa vzájomne volajú a odovzdávajú si výstupné hodnoty	
<b>8 Kód programu</b>	identifikátory sú nejasné, z ich názvu nie je možné dedukovať ich význam	identifikátory (názvy premenných a funkcií) sú popisné, z ich názvu je možné dedukovať význam ich obsahu, resp. použitia	funkcie obsahujú dokumentačné reťazce popisujúce výsledok funkcie, vstupné a výstupné hodnoty	dokumentačné reťazce sú úplné, vstupy a výstupy sú definované podľa štandardu	
<b>9 Používateľské rozhranie</b>	grafické rozhranie nie je k dispozícii alebo je neprehľadné	grafické rozhranie je prehľadné	je zrejmé, ktoré polia sú vstupné a ktoré výstupné, obsah polí je popísaný (napr. pomocou Label)	výstupné polia sú chránené voči používateľským vstupom	
<b>10 Prezentácia projektu žiakom</b>	žiak vie nanajvýš „prerozprávať“ časti kódu	žiak vie popísať logiku algoritmu a identifikovať jednotlivé časti programu (napr. táto funkcia robí to a to)	žiak vie zdôvodniť použitý algoritmus, postup, žiak vie vysvetliť jednotlivé časti programu	žiak vie popísať hraničné (extrémne, krajné) prípady a ako na ne v programe reaguje, príp. v ktorých situáciách program nebude pracovať správne	
<b>11 Využitie projektu v praxi</b>	použiteľný v praxi po veľkých úpravách	použiteľný v praxi po menších úpravách	použiteľný v praxi bez potrebných úprav		
<b>Spolu</b>					

## PREZENTÁCIE (TÍMOVÝCH) KOMPLEXNÝCH PROJEKTOV S DISKUSIOU (CCA 42 MIN)

Pre hladký priebeh prezentácii je dôležité, aby učiteľ pripravil projekciu, uložil na prezentačný počítač všetky súbory potrebné na prezentáciu projektov (finálne verzie programov, prípadne prezentácie či videozáznamy), určil poradie prezentovaných projektov.

Počas prezentovania projektov učiteľ sleduje dodržiavanie časovej disciplíny, v diskusii uvádza objektívne zistenia, ale tiež sa snaží vyzdvihnúť pozitívne aspekty projektu a celkovo prispieť k pozitívnej atmosfére tejto vyučovacej hodiny. Do žiakmi vypracovanej hodnotiacej rubriky zapisuje svoje poznámky a prípadne upravuje žiakmi vybrané možnosti pri jednotlivých položkách. Sleduje tiež a zaznamenáva aktivitu jednotlivých členov tímu počas prezentácie a diskusie.

## SEBAHODNOTENIE PRÁCE ŽIAKOV POMOCOU DOTAZNÍKA (CCA 3 MIN)

Na konci hodiny žiaci vyplnia sebahodnotiacu kartu, v ktorej uvedú:

- ďalšie funkcionality, o ktoré by doplnil či vylepšili ich komplexný projekt,
- mieru spokojnosti s tímovou prácou a výsledkom projektu
- svoje postoje k programovaniu (aspekty jeho náročnosti, zaujímavosti, dôležitosti)
- informáciu či naprogramovali niečo užitočné (hru, pomôcku pre nejaký školský predmet)

Hlavným účelom tohto dotazníka je získať aj subjektívne postojové informácie od žiakov pre celkové vyhodnotenie kurzu programovania. Takto učiteľ, zistí mieru spokojnosti žiakov s tímovou prácou a aký majú žiaci vzťah k programovaniu. Tieto informácie spolu s výsledkami 4 priebežných testov a prezentácie komplexných projektov by mali dať učiteľovi dostatočnú spätnú väzbu, aby na jej základe vedel skvalitniť svoju výučbu programovania.

Uved'te na koľko percent hodnotíte **celkovú funkčnosť Vášho programu** vzhľadom k anotácii: ..... %

Uved'te **ďalšie funkcionality programu**, o ktoré by sa dal **doplniť** a **vylepšiť** Váš komplexný projekt:

.....

.....

.....

.....

Uved'te, za ktoré záležitosti pri tvorbe Vášho komplexného projektu ste boli Vy **osobne zodpovedný/á**:

.....

Do akej miery ste **spokojný/á** s **tímovou prácou** a **výsledkom projektu**? Vyberte vhodnú odpoveď pre každú položku:

S/So	som	veľmi spokojný/á	skôr spokojný/á	neviem	skôr nespokojný/á	veľmi nespokojný/á
svojou prácou na projekte						
prácou ostatných členov tímu						
výsledným programom						

Uved'te, čo ste sa pri tvorbe projektu **nové naučili**:

.....

Aké sú Vaše postoje k **programovaniu**? Vyberte vhodnú odpoveď pre každú položku:

Programovanie je	určite áno	skôr áno	neviem	skôr nie	určite nie
náročné					
zaujímavé					
dôležité					

Okrem tohto komplexného projektu ste naprogramovali niečo užitočné **aj mimo vyučovania** programovania, napr. hru, pomôcku pre nejaký školský predmet?      áno – nie

Ak áno, stručne uved'te o **aké softvérové aplikácie** išlo:

.....

---

## **CELKOVÉ HODNOTENIE PROJEKTOV A ICH ZALOŽENIE DO PORTFÓLIA**

Výber spôsobu hodnotenia komplexných projektov nechávame na samotnom učiteľovi. Pri klasifikovaní projektu známku učiteľ primárne využije informácie zo žiackej sebahodnotiacej rubriky projektu, ktoré prípadne skoriguje. Rovnako aj pri binárnom hodnotení – uspel/neuspel učiteľ využije sebahodnotiacu rubriku, kde o (ne)úspešnosti projektu rozhodne podľa nastavenej prahovej hodnoty bodov.

Výsledné komplexné projekty žiakov spolu s hodnotením učiteľa odporúčame uložiť do elektronického portfólia žiakov aj do učiteľského portfólia samotného učiteľa.



## Bibliografia

**BCS Barefoot. 2019.** Computational Thinking Concepts and Approaches. [Online] 2019. <https://www.barefootcomputing.org/concept-approaches/computational-thinking-concepts-and-approaches>.

**Bender, William N. 2012.** *Project-based learning: differentiating instruction for the 21st century*. s.l. : Corwin, 2012. 978-4129-9790-4.

**Čapek, Robert. 2015.** *Moderní didaktika? Lexikon vyukových a hodnotících metod*. Praha : Grada Publishing, a.s., 2015. 978-80-247-3450-7.

**Štátny pedagogický ústav. 2020.** Inovovaný ŠVP pre gymnázia so štvorročným a päťročným vzdelávacím programom. *Štátny pedagogický ústav*. [Online] 08. 09 2020. [https://www.statpedu.sk/files/articles/dokumenty/inovovany-statny-vzdelavaci-program/informatika\\_g\\_4\\_5\\_r.pdf](https://www.statpedu.sk/files/articles/dokumenty/inovovany-statny-vzdelavaci-program/informatika_g_4_5_r.pdf).

**Turek, Ivan. 2008.** *Didaktika*. Bratislava : Iura Edition, 2008. 978-80-8078-198-9.

## PRÍLOHA

Umiestnenie prílohy: <https://registracia.itakademia.sk/admin/theme/download/zim-python.zip>

Obsah prílohy:

- priečinok **ucitel**
  - súbor **pracovny\_zosit\_riesene\_ulohy.docx** – pracovný zošit s pracovnými listami, ktoré obsahujú zadania a riešenia úloh pre jednotlivé kapitoly,
  - priečinok **pracovne\_subory\_riesenia/** – obsahujúci podpriečinky **01** až **27** obsahujúce pythonovské zdrojové kódy riešení úloh, excelovské tabuľky pre vyhodnotenie výsledkov vyučovania, prípadne prezentácie k výučbe,
  - priečinok **testy/** – obsahujúci priečinky **test1**, **test2**, **test3**, **test4** so zadaniami a riešeniami didaktických testov s prípadnými pracovnými súbormi.
- priečinok **ziak**
  - súbor **pracovny\_zosit.docx** – pracovný zošit pre žiakov s pracovnými listami, ktoré obsahujú zadania úloh pre jednotlivé kapitoly,
  - priečinok **pracovne\_subory/** – obsahujúci podpriečinky **01** až **27** obsahujúce pythonovské zdrojové kódy pracovných súborov prípadne referenčné materiály či inštruktážne listy.