MASARYKOVA UNIVERZITA V BRNĚ

FAKULTA INFORMATIKY

Výuka programování v jazyce Python pro střední školy

DIPLOMOVÁ PRÁCE

Mgr. Hana Horáčková

Brno, 2014

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracovala samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používala nebo z nich čerpala, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

.....
Hana Horáčková

Poděkování

Je mi milou povinností poděkovat vedoucímu diplomové práce panu RNDr. Jaroslavu Pelikánovi, Ph.D. za nevšední ochotu, s kterou se ujal vedení mé práce a zvlášťě za neocenitelné rady, které mi poskytl při jejím zpracování.

Hana Horáčková, diplomantka

Shrnutí

Tato práce obsahuje zhodnocení jazyka Python z hlediska jeho vhodnosti pro výuku programování na střední škole. Následně je rozebrán jazyk Python podle vyučovaných oblastí programování. Poslední částí jsou praktické příklady doplněné o metodické pokyny pro učitele.

Klíčová slova

Python, výuka programování, programovací jazyk pro výuku, příklady pro výuku programování, střední školy.

Obsah

1. Úvod

Téma jsem si zvolila, protože věřím, že jazyk Python je vhodnou volbou pro výuku žáků, kteří se s programováním zatím nesetkali, nebo jen v jednodušší formě na základní škole (např. Baltík, Scratch). Sama bych při výuce algoritmizace volila právě jazyk Python.

Tato diplomová práce si klade za cíl jednak podrobnější zhodnocení programovacího jazyka Python pro výuku programování na středních školách, a také poskytnutí příkladů vhodných pro výuku zájemcům převážně z řad učitelů.

Při výuce programování na středních školách jsou používány různé programovací jazyky, každý z nich má své výhody i nevýhody. Zhodnocení různých programovacích jazyků pro potřeby výuky na středních školách se věnuje např. ve své diplomové práci Lukáš Rýdlo [2]. Python je zde uveden jako jeden z jazyků vhodných pro výuku začátečníků. Na jeho práci navazuje tato diplomová práce podrobnějším rozborem. Základní teoretické poznatky jsou čerpány především z dokumentace Pythonu 3.3.5 [43] a z knihy Marka Summerfielda Python 3: výukový kurz [40]. Hlavními didaktickými zdroji byl zvláště e-learningový kurz Learn to Program: The Fundamentals [17].

Diplomová práce je rozdělena do tří stěžejních částí. V první části je zhodnocena volba jazyka Python pro výuku programování z různých hledisek. Následující část rozpracovává specifika jazyka Python v jednotlivých vyučovaných oblastech. Ve třetí části jsou uvedeny příklady vhodné pro využití při výuce. Tyto příklady jsou doplněné o metodické pokyny, používané pojmy a řešení.

2. Volba jazyka Python pro výuku programování

Při volbě jazyka příhodného pro výuku programování je vhodné mít na zřeteli několik oblastí:

- **aktuálnost a používanost** – Při výběru již málo používaného jazyka žáci nerozumí, proč se mají učit něco zastaralého. *Argument, že základy programování jsou ve všech jazycích v zásadě stejné, je sice v jádru pravdivý, nikoli však účinný.* [21] Pro mnohé žáky není programování jednoduché a přeučení se na jiný jazyk obtížné, prestože jsou základy téměř stejné.
- **dostupnost** – Ideální stav nastává pokud žáci mohou mít platformu použitou pro výuku naistalovanou i na svých počítačích doma. Existuje tak možnost podívat se znovu a v klidu na programy řešené ve škole, připravit se na písemku apod. V neposlední řadě ekonomická situace mnoha škol neumožňuje využití placeného prostředí.
- **čitelnost kódu** – Příliš mnoho znaků použitých „navíc“ při zápisu programu jej činí méně čitelným. Navíc žákům s dyslexií (porucha čtení), dysgrafií (porucha psaní) a dysortografií (porucha pravopisu) činí nepřehledný, obtížně čitelný i zapisovatelný. Ve třídách běžně bývají žáci s diagnostikovanou poruchou učení. Někteří měli tyto problémy na ZŠ, ale na SŠ již nenavštěvují pedagogicko-psychologickou poradnu z důvodu, že se za své problémy stydí, nebo se setkali s negativním přístupem učitelů apod.
- **rozšiřitelnost** – Je pedagogicky vhodné upozorňovat na využití programovacího jazyka v již probraném učivu, či naopak se v budoucnu, v nově probíraných tématech vracet a vysvětlovat použitelnost osvojených znalostí (i částečně). A tím nejen obnovat a upevňovat naučené znalosti, ale i ukazovat na jejich praktickou využitelnost. Na mnohých školách je nabízen rozšiřující předmět nazvaný např.: Seminář z informatiky, kde bývají řešeny i obtížnější témata z programování.
- **literatura** – Žáci by měli mít ke studiu vhodnou literaturu, možno i elektronickou, která jim bude nápomocna při nepochopení učiva ve škole, při absenci, či pro objasnění nejasností. Případně i pro rozšíření znalostí aktivních a nadaných žáků.
- **imperativní strukturované nebo objektově orientované paradigma** – Funkcionální a logické paradigma není příliš vhodné pro výuku na SŠ vzhledem k potřebě matematických znalostí, které jsou na SŠ vyučovány až ve vyšších ročnících nebo mnohdy nemají dostatečný rozsah. [2]
- **využívá to, co budu učit** – Je nutné vybrat takový jazyk, který pokryje všechny části programování, které si škola stanovila pro výuku ve svém školním vzdělávacím programu (dále ŠVP).

Některé z výše uvedených bodů jsou dále podrobněji rozebrány pro jazyk Python.

2.1. Rámcový vzdělávací program a školní vzdělávací program z hlediska výuky algoritmizace

Rámcový vzdělávací program (dále RVP) stanovuje základní vzdělávací úroveň pro všechny absolventy příslušného oboru, kterou musí škola respektovat ve svém ŠVP, vymezuje závazný vzdělávací obsah – očekávané výstupy a učivo. Níže jsou rozepsány tematické celky týkající se programování v RVP oboru Informační technologie, v RVP pro gymnázia a v katalogu požadavků k maturitní zkoušce z informatiky.

Rámcový vzdělávací program oboru 18 – 20 – M/01 Informační technologie

V tomto oboru je zahrnut samostatný obsahový okruh programování a vývoj aplikací, pro který je stanoven minimální počet 8 týdenních, 256 celkových vyučovacích hodin za celou dobu vzdělávání. [18]

PROGRAMOVÁNÍ A VÝVOJ APLIKACÍ

Cílem obsahového okruhu je naučit žáka vytvářet algoritmy a pomocí programovacího jazyka zapsat zdrojový kód programu. Žák porozumí vlastnostem algoritmů a základním pojmům objektově orientovaného programování, dále se naučí používat zápis algoritmu, datové typy, řídící struktury programu, jednoduché objekty a základní příkazy jazyka SQL. Podstatnou část vzdělávání v programování a vývoji aplikací představuje samostatná tvorba jednoduchých aplikací, statických a dynamických WWW stránek. [18]

Výsledky vzdělávání	***Učivo***
Žák: *- zná vlastnosti algoritmu;* *- zanalyzuje úlohu a algoritmizuje ji;* *- zapíše algoritmus vhodným způsobem;*	***1 Algoritmizace*** *- význam, prvky algoritmu*
- použije základní datové typy; *- použije řídící struktury programu;* *- vytvoří jednoduché strukturované programy;*	***2 Strukturované programování*** *- datové typy* *- řídicí struktury*
- rozumí pojmům třída, objekt a zná jejich základní vlastnosti; *- použije jednoduché objekty;*	***3 Úvod do objektového programování*** *- třída, objekt, vlastnosti tříd*
- zná výhody použití jazyka SQL; *- použije základní příkazy jazyka SQL;*	***4 Základy jazyka SQL*** *- základní příkazy (SELECT, UPDATE, INSERT, DELETE)*
- aplikuje zásady tvorby WWW stránek; *- orientuje se ve struktuře HTML stránky;* *- vytvoří webové stránky včetně optimalizace a validace;* *- použije formuláře a skriptovací jazyk.*	***5 Tvorba statických a dynamických webových stránek***

Tab. 2.1.: Výsledky vzdělávání a učivo v RVP Informační technologie [18]

Python je sice možné využít při výuce celé oblasti, ale je vhodné využít a seznámit žáky i s jinými, v dané oblasti rozšířenějšími jazyky.

Rámcový vzdělávací program pro gymnázia

V rámci vzdělávací oblasti Informatika a informační a komunikační technologie je algoritmizace obsažena v tematickém celku zpracování a prezentace informací [7]. Minimální časová dotace celé vzdělávací oblasti Informatika a informační a komunikační technologie jsou 4 týdenní hodiny za celé 4 roky.

ZPRACOVÁNÍ A PREZENTACE INFORMACÍ
Očekávané výstupy
žák

- *zpracovává a prezentuje výsledky své práce s využitím pokročilých funkcí aplikačního softwaru, multimediálních technologií a internetu*
- *aplikuje algoritmický přístup k řešení problémů*

Učivo

- ***publikování*** *– formy dokumentů a jejich struktura, zásady grafické a typografické úpravy dokumentu, estetické zásady publikování*
- ***aplikační software pro práci s informacemi*** *– textové editory, tabulkové kalkulátory, grafické editory, databáze, prezentační software, multimedia, modelování a simulace, export a import dat*
- ***algoritmizace úloh*** *– algoritmus, zápis algoritmu, úvod do programování* [7]

Katalog požadavků zkoušek společné části maturitní zkoušky – Informatika – základní úroveň obtížnosti.

Přestože v současné době není Informatika zahrnuta mezi státní část maturitní zkoušky, je vhodné zde uvést, jakou základní znalost z algoritmizace by měli mít všichni maturanti z informatiky. Při srovnávání požadavků kladených katalogem požadavků CERMATu a výstupy RVP ohodnotila pracovní skupina v rámci Jednoty školských informatiků tematický celek Algoritmizace a základy programování nejlépe ze všech, nikým nebyl označen jako „zcela mimo“. Pracovní skupina se zaměřila jak na RVP gymnázií, tak odborných škol. [11]

ALGORITMIZACE A ZÁKLADY PROGRAMOVÁNÍ
Algoritmizace úlohy, vlastnosti algoritmu
Žák dovede:

- *vysvětlit pojem algoritmus a jeho základní vlastnosti;*
- *algoritmizovat jednoduchou úlohu.* [6]

Z výše uvedeného vyplývá, že jednoduché základy algoritmizace by měly být zahrnuty ve všech maturitních oborech na středních školách, nicméně některé školy zařadily tuto část do volitelného předmětu.

Dále je rozebrána struktura výuky programování na třech gymnáziích, které měly zvěřejněny ŠVP nebo obsahy předmětů na svých webových stránkách. Na Biskupském gymnáziu v Brně není algoritmizace v povinném předmětu Informatika a výpočetní technika vůbec probírána. Tato problematika se objevuje až v jednoletém volitelném předmětu ve 3. ročníku, který vyplňuje roční mezeru v povinné výuce informatiky a ve dvouletém maturitním semináři ve 3. a 4. ročníku. [22] Na gymnáziu Elgartova v Brně se objevuje základní algoritmizace a programování na konci dvouletého povinného předmětu Informační a komunikační technologie. Prohloubení znalostí algoritmizace a seznámení s technikami strukturovaného i objektově orientovaného programování je v rámci volitelného dvouletého semináře Základy programování. [23] S těmito dvěma modely, jak výukou pouze ve volitelném předmětu, tak začleněním základů v povinném kurzu a různých variant rozšíření ve volitelném předmětu, zřejmě patří k nejrozšířenějším možnostem na gymnáziích.

Možná náplň výuky algoritmizace a programování na základě obsahů volitelných informatických předmětů Biskupského gymnázia [22], gymnázia Elgartova [23] a gymnázia Brno – Řečkovice [24]:

- pojem algoritmus a jeho vlastnosti
- proměnné a konstanty
- základní a strukturované datové typy
- větvení a cyklus
- vstupy a výstupy zpracovávaných dat
- funkce
- práce se soubory
- algoritmy řazení a vyhledávání
- rekurze
- dynamické datové struktury (lineárně zřetězený seznam, zásobník, fronta)
- práce s grafikou

Další vyskytovaná rozšíření (vyskytly se pouze v jednom ŠVP):

- vývojové diagramy
- optimalizace
- objektově orientované programování
- binární vyhledávací strom

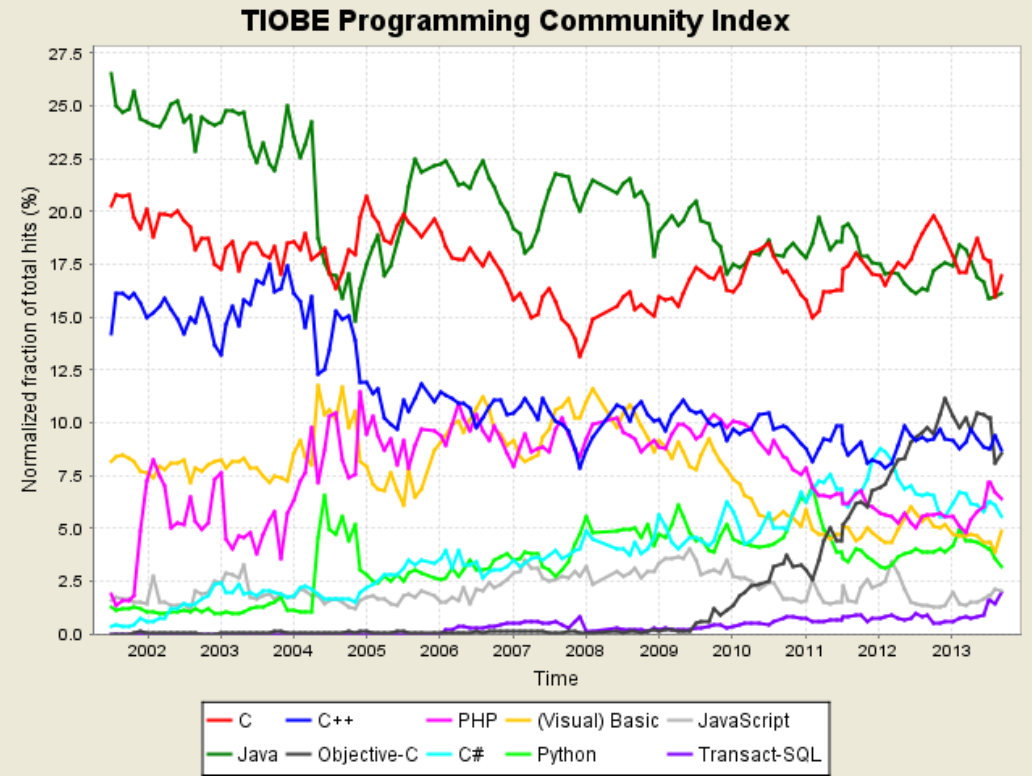
2.2. Aktuálnost a používanost

K posouzení aktuálnosti byl použit TIOBE Programming Community index, jenž je indikátorem popularity programovacích jazyků dle nejpopulárnějších vyhledávačů. V Tab.2.2 jsou silnějším písmem zvýrazněny programovací jazyky převážně používané ve výuce podle výzkumu L. Kotka [20]. Python patří spolu s C, Javou, C++, PHP, C# a Visual Basicem k první desítce.

Pozice září 2013	Pozice září 2012	Programovací jazyk	Ratingy září 2013
1	**1**	**C**	**16,975%**
2	**2**	**Java**	**16,154%**
3	**4**	**C++**	**8,664%**
4	3	Objektive-C	8,561%
5	**6**	**PHP**	**6,430%**
6	**5**	**C#**	**5,564%**
7	**7**	**(Visual) Basic**	**4,837%**
8	**8**	**Python**	**3,169%**
9	11	JavaScript	2,015%
10	14	Transact-SQL	1,997%
11	15	Visual Basic .NET	1,844%
12	9	Perl	1,692%
13	10	Ruby	1,382%
14	**12**	**Delphi/Object Pascal**	**0,897%**
15	**16**	**Pascal**	**0,888%**
16	13	Lisp	0,770%
17	19	PL/SQL	0,676%
18	24	R	0,646%
19	20	MATLAB	0,639%
20	25	COBOL	0,628%

Tab. 2.2 : TIOBE Programming Community index [25]

Z dlouhodobého hlediska vykazují Python spolu s C# v posledních 10 letech mírný růst popularity, zatímco Visual Basic a PHP mírný pokles v posledních 5 letech. Python však v posledních 3 letech stagnuje. Popularita Javy a C++ poklesla, přesto Java patří na čelní pozice spolu se stálým C, hned za nimi je C++ (viz graf 2.1).



Graf 2.1: Dlouhodobý trend TIOBE Programming Community index pro prvních 10 programovacích jazyků [25]

Python se vyskytuje v následujících sektorech: automatizace, řízení, elektrotechnika (0,87 %), telekomunikace (0,56 %), tvorba software na zakázku (2,11 %), software pro informační technologie (4,14 %), ekonomika (5,12 %). Znalost Pythonu požadují české firmy zaměřené na Linux v 15,24 %, nadnárodní firmy zaměřené na Linux v 4,76 % a nadnárodní firmy zaměřené na na embedded zařízení v 5 %. Softwarové a vývojové firmy, které se zabývají programováním vlastních aplikací používají Python v 9,71 %. [26]

Konrétně jej využívá např. Seznam.cz, kde hlavní programovací jazyky jsou C/C++ a Python, NASA pro implementaci systémů pro řízení letu [37], Red Hat ve svých nástrojích pro instalaci Linuxu [37]. Některé společnosti nezveřejňují používání Pythonu, neboť věří, že jim přináší konkurenční výhodu. [37] Django je webový framework napsaný v jazyce Python [15]. V současné době hledá zaměstnance se znalostí Pythonu v České republice: Seznam.cz, Skype, FORTUNA GAME a.s., ING Pojišťovna, Národní knihovna České republiky [28]; v USA: Nokia pro její Cloud computing team, University of Oregon, Panasonic, eBay. [29]

2.3. Dostupnost

Python je vyvíjen jako open source software, je k dispozici zdarma pro různé operační systémy na oficiálních webových stránkách programovacího jazyka Python [8]. Ve většině distribucí Linuxu je již součástí základní instalace. Pro psaní skriptů je možné použít textový editor, např. PSPad, který lze nastavit pro práci při programování v Pythonu [12]. Jednotlivé části programu lze testovat v interaktivním

Pythonu (dále IPython), případně i v některém z online interpretrů, např. Codeacademy Labs [30] (pouze Python 2.7.2), codepad [31], compileonline [32], Ideone [33]. Zkušenější programátoři mouhou využít i integrovaná vývojová prostředí (dále IDE), např. PyDev pro Eclipse, KomodoEdit. [13] Pro středoškolské studenty je vhodné použití IDLE (Python IDE), ve kterém je k dispozici IPython i textový editor. IDLE je skutečné vývojové prostředí, nikoliv uměle vytvořené pro výuku (což je vhodnější z hlediska motivace), přesto uživatelsky přehledné.

2.4. Paradigma

Python je multiparadigmatický jazyk, strukturovaný i objektový s funkcionálními prvky. Volba závisí na zkušenostech a preferencích programátora a vhodnosti přístupu k zadané úloze.

2.5. Čitelnost kódu

Syntaxe v Pythonu je jednoduchá a snadno naučitelná. Stává se, že programování stejné úlohy zabere v Pythonu pětinu času a kódu než v jazyce C. [36] Python byl inspirován jazykem ABC, který nebyl určen pro systémové programování, ale pro výuku začátečníků.

V Pythonu jsou jednoduché programy skutečně jednoduché. Podpuruje funkce a třídy, ale nevynucuje si je. C++ vyžaduje, aby byl program uvnitř funkce a předcházela mu direktiva preprocesoru, v Javě musí být celý kód uvnitř třídy. Blok v Pythonu je určen odsazením, příkaz je ukončen znakem konce řádku, čímž odpadají začátečnické problémy s opomenutím závorek a středníků. Množství kódu, které musí žáci psát, redukuje dynamické typování, kdy není třeba deklarace proměnných. [34] Naproti tomu je nutné žáky více vést k uvědomění si typu proměnných, se kterými pracují, například pomocí docstrings. Díky vynucenému odsazování se Pythonovský kód psaný někým jiným snadněji čte a je srozumitelnější. [38]

2.6. Rozšiřitelnost

Rozšíření je možné pomocí modulů, které jsou obdobou knihoven v jazyce C a Pascal. Moduly, které by mohly být zajímavé či použitelné ve výuce, jsou uvedeny v tabulce 2.3, kde je uvedeno i pro jaké účely je modul vhodný a pro které verze Pythonu je dostupný. Pokud je uvedeno „vždy k dispozici“ je modul součástí základní distribuce, v opačném případě je nutné jej doinstalovat.

Název modulu	Použití	Verze Pythonu
math	matematické funkce	vždy k dispozici
random	generátor náhodných čísel	vždy k dispozici
datetime, time	práce s časem a datem	vždy k dispozici
turtle	želví grafika	vždy k dispozici
winsound	přehrávání zvukových souborů ve formátu WAV	vždy k dispozici pro Windows
pylab (matplotlib)	vykreslení grafů	2.6 – 3.3
Tkiner (tkiner v Pythonu 3)	vytváření oken	2.6 – 3.3
sqlite3	práce s databází	vždy k dispozici v Pythonu 3
MySQLdb	práce s databází	2.4 – 2.7
pygame	tvorba počítačových her	2.4 – 3.2
django	webové stránky	2.6.5 a vyšší

Tab 2.3: Některé moduly jazyka Python

2.7. Studijní literatura

Základní učební text používaný pro středoškolské studenty by měl být psaný česky, mnozí žáci nemají dostatečné znalosti angličtiny. Jako rozšiřující text je možno uvést i anglicky psané texty, aby si žáci uvědomili nutnost jazykových znalostí v této oblasti. Pro některé studenty je nákup učebnic problematický z finančních důvodů. Možnost mít k dispozici učební text dostupný na webových stránkách je vítaný nejen pro ně. U těchto materiálů nemůže nastat situace, kdy si je žák zapomene doma či předčasně prodá. Při doporučení textů pro jinou verzi je nutné upozornit na rozdíly. Přesto s tím mohou mít žáci problémy, proto je vhodnější jako primární studijní literaturu zvolit tu, jenž je psaná pro vyučovanou verzi. Ostatní texty je možné doporučit jako doplňující, zvláště pokud obsahují části, které jsou probírány, ale nevyskytují se v primárně zvoleném textu.

Dále je uveden stručný rozbor některých učebnic a tutoriálů, včetně stručné charakteristiky z hlediska použití pro výuku programování pomocí Pythonu úplných začátečníků na SŠ. Učebnice informatiky ve výčtu zahrnuty nejsou, protože k výkladu algoritmizace využívají jiných programovacích jazyků (vývojové diagramy, C#, Karel apod.).

Učebnice jazyka Python (aneb Létající cirkus)
Autor: Guid van Rossum, Fred L. Drak
Překladatel: Jan Švec
Verze Pythonu: 2.2
Dostupnost: tutorial online i ke stažení http://www.py.cz/UcebniceJazykaPython/

Charakteristika: Poměrně srozumitelný text, ale již zastaralý. Místy je obsah přesahující středoškolskou výuku programování, nicméně použitelný jako rozšíření (např. třídy).

Ponořme se do Pythonu 3
Autor: Mark Pilgrim
Překladatel: Petr Přikryl
Verze Pythonu: 3.3
Dostupnost: Tištěná publikace Edice CZ.NIC, dostupná online i ke stažení v pdf zdarma na http://diveintopython3.py.cz/.
Charakteristika: Tištěná i online dostupná kniha, která však svým širokým záběrem a rychlým nástupem není nejvhodnější pro úplné začátečníky.

Sallyx.org
Autor: Petr Bílek
Verze Pythonu: 2.7 i 3.3
Dostupnost: Elektronický tutorial – http://www.sallyx.org/sally/python/
Charakteristika: Online tutorial, který i srovnává rozdíly mezi zmíněnými verzemi. Výhodou je srozumitelný text pro začátečníky a mírně pokročilé, ve kterém je upozorněno i na možné chyby, které by žáci mohli dělat.

Jak se naučit programovat
Autor: Alan Gauld
Překladatel: Petr Přikryl
Verze Pythonu: 2.3
Dostupnost: http://www.freenetpages.co.uk/hp/alan.gauld/
http://jaksenaucitprogramovat.py.cz/
Charakteristika: Při výkladu jednotlivých částí autor srovnává s jazyky VBScript a JavaScript. Stránky jsou tedy vhodné spíše pro ty, kteří již mají s některým z těchto jazyků zkušenosti. Pro začátečníky by to mohlo být spíše matoucí.

Seriál Python
Autor: Pavel Kosina
Verze Pythonu: 2.3
Dostupnost: http://programujte.com/clanky/54-serial-python/
Charakteristika: Webový seriál článků vysvětlující základy jazyka Python doplněný příklady k procvičení.

Buwralug, Livewires
Autor: Gareth McCaughan
Překladatel: Pavel Kosina
Verze Pythonu: Buwralug – 3.0 (+Tkiner), Livewires – 2.5 (+Livewires)
Dostupnost: http://www.geon.wz.cz/buwralug/index.html/
http://www.geon.wz.cz/livewires/

Charakteristika: Poměrně jednoduché pracovní listy, které provází napsáním různých programů dobře využitelných ve výuce. Ale některé pojmy jsou však pro potřeby výuky příliš zjednodušeny.

Učíme se programovat v jazyce Python 3
Autor: Jeffrey Elkner, Allen B. Downey and Chris Meyers, (M.K. Bradshaw, Peter Wentworth)
Překladatel: Jaroslav Kubias
Verze Pythonu: 3.x (+Tkiner)
Dostupnost: http://howto.py.cz/index.htm/
Charakteristika: Stále aktualizované srozumitelné učební texty doplněné příklady k procvičení. Většina oblastí je využitelná ve výuce na střední škole. Ze všech zde uvedených učebních textů je pro svou aktuálnost a obsahovou využitelnost nejvhodnější.

Python 3 výukový kurz
Autor: Mark Summerfield
Překladatel: Lukáš Krejčí
Verze Pythonu: 3.x
Dostupnost: tištěná publikace Computer Press
Charakteristika: Jediná, v současné době, běžně dostupná kniha o programování v Pythonu v knihkupectvích. Její cena je však pro použití ve školní výuce příliš vysoká (kolem 700 Kč). Vzhledem k ceně i rozsahu, který převyšuje vyučované oblasti, se hodí spíše pro ty, kteří se chtějí programováním v jazyce Python hlouběji zabývat. Pro tyto lidi nabízí mnoho témat poměrně dobře vysvětlených.

Začínáme programovat v jazyce Python
Autor: Daryl Harms, Kennets McDonald [36]
Překladatel: Ivo Fořt, Lubomír Škapa
Verze Pythonu: 1.5.2
Dostupnost: tištěná publikace Computer Press
Charakteristika: Tato kniha je psaná velice srozumitelně a pochopitelně, ale je již nedostupná a zastaralá. V době prodeje stála kolem 500 Kč. Pro výuku na střední škole je příliš obsáhlá, využitelná je přibližně čtvrtina obsahu (při rozšíření o objektové programování a tkiner asi třetina).

Python 1. –11. (seriál)
Autor: Aleš Hakl
Verze Pythonu: 2.3
Dostupnost: http://www.linuxsoft.cz/article_list.php?id_kategory=217/
Charakteristika: Webový seriál článků vysvětlující základy jazyka Python. Některé části nejsou dostatečně vysvětleny, začátečník by nemusel pochopit podstatu popisovaných jevů.

3. Základy jazyka Python z hlediska výuky na střední škole

V této kapitole jsou rozebrány jednotlivé části jazyka Python, které se objevují ve výuce algoritmizace a programování dle kap. 2.1. Nejsou zde zmíněny části pojem algoritmus a jeho vlastnosti a algoritmy řazení a vyhledávání, jejichž teorie se volbou jazyka nezmění. Naopak je text rozšířen o komentáře, dokumentační řetězec a moduly. Mezi moduly jsou uvedeny ty, které se vyskytují v příkladech ve 4. části. Pro výklad byla zvolena novější verze Python 3, z důvodu dlouhodobějšího využití této diplomové práce. V textu se objevují zdrojové kódy, či ukázky z interaktivního režimu psané `neproporcionálním fontem` pro demonstraci vysvětlovaných funkčností, případné komentáře jsou uvozeny symbolem #. *`Kurzíva neproporcionálního fontu`* označuje místa, která mají být nahrazena konkrétní hodnotou.

3.1. Komentáře a dokumentační řetězec

Pro lepší pochopení programu, jak ostatními programátory, tak samotnými tvůrci, kteří se k programu vrátí po určité době, je vhodné jej opatřit vyvětlivkami a poznámkami, tzv. komentáři. Komentář je možné umístit na samostatný řádek nebo na konec řádku. Začíná symbolem # (hash) a končí koncem řádku. Vše, co je napsáno mezi těmito znaky, je Pythonem ignorováno. [42]

Ukázka komentáře (zvýrazněn tučně):

```
def rychlost(metry, sekundy):
    v = metry/sekundy
    return round(v) #zaokrouhleni na cele cislo
```

Pro komentář na více řádků je možné využít víceřádkový řetězec, pomocí ztrojených apostrofů. Víceřádkový komentář, umístěný na začáteku funkce, metody, třídy nebo modulu, Python považuje za dokumentační řetězec (tzv. docstring). Docstring je využit pro nápovědu, například použitím funkce `help`, ale také v tooltipu – informačního okénka, při psaní příkazů v interaktivním režimu [38, 17]

Ukázka dokumentačního řetězce (zvýrazněn tučně):

```
def rychlost(metry, sekundy):
    '''(num,num)-> num

    Zjisti rychlost v m/s po zadani drahy v metrech a casu
    v sekundach.

    >>> rychlost(13000,4)
    3250.0
    >>> rychlost(256,12)
    21.333333333333332
    '''
    v = metry/sekundy
    return v
```

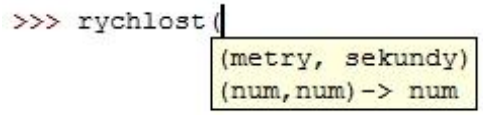
Ukázka využití dokumentačního řetezce funkcí help:

```
>>> help(rychlost)
Help on function rychlost in module __main__:

rychlost(metry, sekundy)
    (num,num)-> num

    Zjisti rychlost v m/s po zadani drahy v metrech a casu
    v sekundach.

        >>> rychlost(13000,4)
        3250.0
        >>> rychlost(256,12)
        21.333333333333332
```



Obr. 3.1 : Ukázka využití dokumentačního řetezce v tooltipu

Víceřádkový řetězec představuje také způsob pro zápis řetězců, kde se vyskytují apostrofy i uvozovky. [38]

3.2. Proměnné a konstanty

V jazyce Python není nutné deklarovat typ proměnné, ani znak pro konec řádku. Proměnné vznikají automaticky při jejich prvním přiřazení. Řádek je ukončen svým koncem. Pokus o volání proměnné, které nikdy nebyla přiřazena žádná hodnota, vyvolá chybové hlášení. Python používá dynamickou práci s typy, kdy lze proměnné opětovně přiřadit jiný objekt, a to i jiného datového typu. V interaktivním interpretu způsobí zápis proměnné vypsání její hodnoty. Proměnné v Pythonu jsou ve skutečnosti odkazy na objekty. Operátor = sváže proměnnou s objektem v paměti. [36, 38, 40]

Syntaxe přiřazení (vytvoření odkazu na objekt): [36]

```
proměnná = hodnota
```

Pravidla pro názvy proměnných [40]:

- první znak musí být písmeno nebo podtržítko
- další znaky mohou být pouze písmena, číslice a podtržítka
- podtržítko nemůže být na začátku i na konci názvu (takto Python označuje metody, u kterých je možné vytvořit si jejich vlastní verze)
- rozlišuje se velikost písmen (x je jiná proměnná než X)
- nemůže mít stejný název jako některé klíčové slovo (viz tab. 3.1)

Konvence pro názvy proměnných [43]:

- víceslovné názvy se oddělují podtržítkem, např. `promenna_x`
- nepoužívat jiné než ASCII znaky

Česká písmena s diakritikou v názvech proměnných jsou možná, ale v některých editorech s nimi mohou mít studenti problémy. Navíc nejsou doporučená konvencí (nepatří mezi ASCII znaky). Stejná pravidla a konvence platí i pro názvy funkcí.

False	*class*	*finally*	*is*	*return*
None	*continue*	*for*	*lambda*	*try*
True	*def*	*from*	*nonlocal*	*while*
and	*del*	*global*	*not*	*with*
as	*elif*	*if*	*or*	*yield*
assert	*else*	*import*	*pass*	
break	*except*	*in*	*raise*	

Tab. 3.1: Klíčová slova [43]

V Pythonu nelze deklarovat konstanty, typicky se deklarují jako běžné proměnné s velkými písmeny. [41]

3.3. Datové typy

V úvodu do základních datových typů je vhodné s žáky pracovat v interpretru Pythonu IDLE.

Pro manipulaci s objekty Python standardně nabízí funkce volané ve tvaru `funkce(`*`argumenty`*`)`, metody volané stylem *`objekt`*`.metoda(`*`argumenty`*`)` a funkce modulů, jejichž syntaxe závisí na způsobu importu (viz kap. 3.9). Metoda je v podstatě funkce volaná pro určitý objekt. [40]

Funkce `type()` zjistí datový typ argumentu:

```
>>> type(True)
<class 'bool'>
>>> type(6)
<class 'int'>
>>> type(1.2)
<class 'float'>
>>> type("Ahoj")
<class 'str'>
```

3.3.1. Čísla (int, float)

K nejpoužívanějším číselným typům na střední škole patří celá a reálná čísla:

`int` celá čísla

`float` reálná čísla – čísla s pohyblivou destinnou čárkou (floating point numbers)

Název float je odvozen od faktu, že není pevně dán počet číslic před a za desetinnou čárkou [44] (např. 0,00025 je i 2,5.10-4).

Mezi celými čísly (integer) a čísly reálnými (floating point) se nerozlišuje deklarací datového typu, ale Python je od sebe poznává podle přítomnosti nebo nepřítomnosti desetinné tečky anebo exponentu uvozeného písmenem e či E: [36, 38]

```
>>> type(2e3)
<class 'float'>
>>> type(1.5)
<class 'float'>
>>> type(4)
<class 'int'>
```

operace [43]

+	sčítání
-	odečítání
*	násobení
**	umocňování (lze i pow(x, y) „x^y“)
/	dělení
//	celočíselné dělení
%	zbytek po celočíselném dělení (modulo)

Dále je možné využít interní funkce, které mohou operovat s čísly a funkce knihovny `math`, které je nutné před použitím importovat. [36]

Jestliže se v jednom výrazu objeví více operátorů, jsou operace vyhodnocovány v určitém pořadí (viz tab. 3.2 a tab 3.9), přičemž jsou respektována pravidla používaná v matematice. Operátory na stejném řádku mají stejnou prioritu a vyhodnocují se zleva doprava kromě umocňování, které se vyhodnocuje zprava do leva. [43] Prioritu je možné upravit pomocí závorek, které mají nejvyšší prioritu a tedy s nimi vyhodnocování začne. Použití závorek je vhodné i u složitějších výrazů, kde kromě požadovaného vyhodnocovaní napomáhá lepší čitelnosti. [36]

Operátor	Popis	Priorita
(), [], {}	vazba nebo n-tice, seznam, slovník, množina	nejvyšší
**	umocňování	
-	aritmetická negace (změna znaménka)	
*, /, //, %	násobení, dělení, zbytek	
+, -	sčítání, odčítání	nejnižší

Tab. 3.2: Priorita aritmetických operátorů [17, 43]

Užitečné interní funkce [43]:

`abs()` absolutní hodnota
`int()` vrátí informaci v argumentu jako typ int
`float()` vrátí informaci v argumentu jako typ float
`round()` zaokrouhlí údaj v argumentu; může být uvedeno další číslo, které určí počet číslic za desetinnou čárkou, na které má být zaokrouhleno
`type()` vrací typ objektu v argumentu
`pow(x, y)` umocňování (x^y)

Při převodu rálného čísla na celé pomocí funkce `int()` dojde k odříznutí desetinné části převáděného čísla.

Kromě výše uvedených numerických typů zvládá Python 3 i komplexních čísla pomocí funkce `complex([real[, imag]])` [43] a zlomky importem modulu `fractions` [38, 43]. Na většině středních škol nejsou komplexní čísla zařazena do standartní výuky matematiky, případně bývají probírána ve volitelných seminářích z matematiky. Z tohoto důvodu není vhodné jejich zavedení ve výuce programování. Zavedení zlomků možné je, ale spíše jako rozšíření probíraného učiva.

3.3.2. Bool

Typ bool nabývá dvou hodnot `True` (pravda) a `False` (nepravda). Pro zápis v Pythonu jsou důležitá počáteční velká písmena, `true` a `false` v něm booleovskými hodnotami nejsou. [38, 42] S `True` lze zacházet jako s číslem 1 a `False` 0. [38] Matematická logika se většinou probírá v prvním ročníku střední školy, je vhodné při výuce propojovat znalosti již získané s novými, což je zahrnuto i v tabulce 3.3.

Boolovské (logické) operátory jsou tři – `and`, `or` a `not`. Podrobněji jsou rozepsány v tabulce 3.3, kde jejich řazení odpovídá prioritě, tedy v prvním řádku je logický operátor s nejvyšší prioritou.

Operátor	Výsledek	Matematický název (matematický symbol)	Čtení (jazykový význam)	Podmínky pravdivosti
`not x`	je-li `x` nepravdivé, vrací True, jinak False	negace (¬)	není pravda, že `x`	je pravda, právě když `x` je nepravdivý
`x and y`	je-li `x` nepravdivé, vrací `x`, jinak `y`	konjunkce (∧)	`x` a `y` (ve významu slučovacím)	je pravda, právě když `x`, `y` jsou oba zároveň pravdivé
`x or y`	je-li `x` nepravdivé, vrací `y`, jinak `x`	disjunkce (∨)	`x` nebo `y` (ve významu nevylučovacím, tj. ve smyslu alespoň jeden)	je pravda, právě když alespoň jeden z `x`, `y` je pravdivý

Tab. 3.3: Boolovské (logické) operátory [39, 43]

Tabulka pravdivostních hodnot v závislosti na pravdivostních hodnotách výrokových proměnných (tab. 3.4) je sestavená na základě podmínek pravdivosti z tab. 3.3.

x	y	not x	x and y	x or y
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

Tab. 3.4: Pravdivostní tabulka [39]

Relační operátory při porovnávání výrazů vrací hodnoty `True` nebo `False`, jejich přehled je uveden v tab. 3.5.

Operace	Význam
<	ostře menší než
<=	menší nebo rovno než
>	ostře větší než
>=	větší nebo rovno než
==	rovná se
!=	nerovná se
is	test identity objektů
is not	negace testu identity objektů
in	test příslušnosti
not in	negace testu příslušnosi

Tab. 3.5: Relační operátory [43]

Operátory `is` a `is not` jsou testy pro identitu objektů, tedy výraz `x is y` je pravdivý pouze pokud `x` a `y` jsou jména pro totožný objekt. [43]

Operátory testu příslušnosti `in` a `not in` jsou podporovány pouze sekvenčními a množinovými typy a slovníkem, kde zjišťují obsah klíče. `x in s` se vyhodnotí na `True` pokud je `x` prvkem `s`, u řetězců pokud je `x` podřetězec `s`. Protože prázdný řetězec je podřetězec jakéhokoliv řetězce, bude `"" in "cokoliv"` vracet `True`.[43]

Od Pythonu 3.0 nelze používat operátory <, >, <= a >= s různými datovými typy (vyhodí výjimku `TypeError`) a znak <> jako „nerovná se“. [45]

Všechny logické operátory mají nižší prioritu než aritmetické operátory. Prioritu je možné upravit pomocí závorek, které i zlepšují čitelnost.

Operátor	Popis	Priorita

`in, not in,`
`is, is not,`
<, <=, >, >=, !=, ==
	testy příslušnosti
	testy identity
	porovnávací operátory
		nejvyšší

`not` *x*	logická negace	
x `and` *y*	logický součin	
x `or` *y*	logický součet	nejnižší

Tab. 3.6: Priorita logických operátorů [37, 43]

Při zpracování logického součinu (`and`) je `y` vyhodnocováno, pouze pokud je `x` vyhodnoceno jako pravda, protože v případě, že je `x False` bude celý výraz `False` a není třeba jej dále kontrolovat. U logického součtu (`or`) je `y` vyhodnocováno, pouze pokud je `x` vyhodnoceno jako nepravda, protože v případě, že `x` je `True` bude celý výraz `True` [17, 37].

Ukázka vyhodnocování logického součinu (gghj je název, který se dosud v programu nevyskytl):

```
>>> False and gghj
False
>>> gghj and False
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    gghj and False
NameError: name 'gghj' is not defined
>>> True and gghj
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    True and gghj
NameError: name 'gghj' is not defined
```

3.3.3 Řetězec (str)

Do řetězce (angl. string) můžeme uložit cokoliv, co můžeme kódovat jako text. [37] V Pythonu 3 jsou všechny řetězce posloupnostmi znaků v Unicode. [38] Řetězce definujeme uzavřením do apostrofů ('...') nebo do uvozovek ("..."), se stejným výsledkem. [38, 43] Trojice apostrofů či uvozovek ("""...""" nebo '''...''') označuje víceřádkový řetězec, případně dokumentační řetězec, pokud se vyskytuje na začátku funkce. [38] V Pythonu není žádný typ pro znak (char), místo něj se používá řetězec s délkou jedna. Řetězce jsou neměnitelné posloupnosti (immutable sequences), není možné v nich měnit jednotlivé znaky. [37]

Operátory [17, 37]:

`'x' in st`	test příslušnosti (je `'x'` obsaženo v `st`?)
`st1 + st2`	spojení řetězců
`k*st`	`k` -opakování `st` (`k` je typu int)
`st*k`	`k` opakování `st` (`k` je typu int)
`st[i]`	kopie `i` -tého znaku (indexováno od 0)
`st[i:(j+1)]`	kopie části řetězce od `i` -tého po `j` -tý znak (indexováno od 0)
`for x in st`	procházení znak po znaku

Užitečné interní funkce a metody [38, 42, 43]:

`len(st)`	délka řetězce `st` (počet znaků)
`st.lower()`	vytvoří kopii řetězce, kde budou všechna písmena malá
`st.upper()`	vytvoří kopii řetězce, kde budou všechna písmena velká
`st.capitalize()`	vytvoří kopii řetězce, kde bude první písmeno velké, ostatní malá
`st.count(x)`	počet výskytů `x` v řetězci
`st.split('znak oddělovače')`	rozdělí řetězec na seznam řetezců podle zadaného oddělovače
`st.splitlines()`	rozdělí víceřádkový řetězec na seznam řetězců, kde každá položka reprezentuje jeden řádek
`st.replace(stare, nove)`	vytvoří kopii řetězce, kde budou podřetězce `stare` nahrazeny za `nove`
`str()`	vrátí informaci v argumentu jako typ str

Jednotlivé znaky řetězce jsou zpřistupněny pomocí indexů. Číslování pořadí zepředu využívá kladná čísla a začíná 0, zezadu používá záporná čísla a začíná od –1.

S	t	r	i	n	g
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

K jednotlivým znakům přistupujeme pomocí indexové notace, tj. operátoru `[pořadí znaku]` psaného za řetězec. [37] Výsledkem je kopie daného znaku. Úsek řetezce (angl. slice) získáme zadáním dvou indexů. Výsledkem bude nový řetězec obsahující znaky od prvního indexu po znak před druhým indexem. Pokud je první index nulový nebo druhý index poslední (zde 5), je možné je vynechat. [38] Pomocí indexové notace a úseku nelze řetezce měnit. Řetězce je možné změnit pouze vytvořením nového řetězce, do kterého je možné přiřadit původní proměnné (pomocí indexování, výběru úseků, operátorů, ...). [37] Výběry prvků a částí mají vyšší prioritu než aritmetické a logické operátory.

Ukázka indexování a výběru úseků:

```
>>> s = 'String'
>>> s[0]
'S'
>>> s[1:3]
'tr'
>>> s[:2]              #řetězec od počátku po druhý znak
'St'
>>> s[2:]              #řetězec kromě prvních 2 znaků
'ring'
```

Python má speciální sekvence znaků, tzv. escape sekvence, které začínají zpětným lomítkem. Pokud jsou použity v řetězci je znak za zpětným lomítkem čten odlišným způsobem. V tabulce 3.7 jsou uvedeny některé běžně používané escape sekvence v Pythonu. [17, 36]

Escape sekvence	Představuje znak
\n	nový řádek
\t	tabulátor
\\	zpětné lomítko
\'	apostrof
\"	uvozovky

Tab. 3.7: Některé escape sekvence [17, 36]

3.3.4 Seznam (list)

Seznam (angl. list) je posloupnost (tj. s definovaným pořadím prvků), která může obsahovat prvky různých datových typů (čísla, řetězce, seznamy, atd.). Na rozdíl od řetezců jsou seznamy měnitelné, tedy je možné prvky přidávat, odebírat, měnit za jiné, a pod. Indexování, výběr prvku a částí seznamu je obdobné jako u řetězců, navíc je možné jejich použití pro změnu seznamu. Seznam vznikne zápisem čárkami oddělených hodnot do hranatých závorek. [36, 37, 38]

Operátory [37, 43]:

`list[i] = x`	položka s indexem `i` je narazena prvkem `x`
`del list[i]`	vymazání prvku s indexem `i`
`list[i:j] = t`	úsek seznamu je narazen `t` se stejnou délkou
`del list[i:j]`	vymazání úseku seznamu (stejné jako `list[i:j] = []`)
`x in list`	test příslušnosti (je `x` prvkem seznamu?)
`x not in list`	negace testu příslušnosti
`list1 + list2`	spojení seznamů
`k*list, list*k`	`k` -opakování (`k` je typu int)
`list[i]`	kopie `i`-tého prvku (indexováno od 0)
`list[i:(j+1)]`	kopie části seznamu od `i`-tého po `j`-tý znak (indexováno od 0)
`for x in list`	procházení prvky seznamu

Užitečné interní funkce a metody [17, 36, 43]**:**

`len(list)`	délka seznamu
`min(list)`	nejmenší prvek seznamu
`max(list)`	největší prvek seznamu
`sum(list)`	součet prvků seznamu (pokud jsou číselného typu)
`list.append(x)`	přidání jednoho prvku `x` na konec seznamu
`list1.extend(list2)`	rozšíří seznam list1 o prvky seznamu list2
`list.insert(i, x)`	vloží prvek `x` na pozici i
`list.pop(i)`	odebere prvek na pozici i, pokud i není zadáno odebere poslední prvek
`list.remove(x)`	odebere první výskyt prvku `x` v seznamu (pokud `x` v seznamu není, vrátí chybu)
`list.sort()`	setřídí seznam od nejmenšího po největší
`list.reverse()`	převrátí pořadí prvků v seznamu
`list.index(x)`	index prvního výskytu prvku `x` v seznamu (pokud `x` v seznamu není, vrátí chybu)
`list.count(x)`	kolikrát se prvek `x` v seznamu nachází

Při odstranění prvku seznamu jsou posunty všechny následující položky tak, aby vzniklá mezera byla zaplněna. [38]

3.3.5 N-tice (tuple)

N-tice (angl. tuple) jsou podobné seznamům, ale nemohou být modifikovány, chovají se jako neměnitelné seznamy. Neposkytují žádné metody, které by je dovolovaly měnit (nemají metody jako `append()`, `extend()`, `insert()`, `remove()`, `pop()`, …). Můžeme používat funkce (`len`, `max`, `min`, `sum`)

a operátory (+, *, `in`), které nemění originál. [36, 38] N-tice je definována pomocí čárek. Kulaté závoky je nutné použít při zápisu prázdné n-tice, tedy (), v ostatních případech jsou pomocné. Čárku je nutné napsat i v jednoprvkové n-tici, závorky je možné vynechat: [43]

Příklad deklarace jednoprvkové n-tice:

```
>>> n = (1)
>>> type(n)
<class 'int'>
>>> t=(1,)
>>> type(t)
<class 'tuple'>
>>> r=1,
>>> type(r)
<class 'tuple'>
```

Převod mezi n-ticí a seznamem [38]:

`tuple(`*x*`)`	převede seznam *x* na n-tici („zmrazí“ seznam)
`list(`*x*`)`	převede n-tici *x* na seznam („rozpustí“ n-tici)

3.3.6 Množina (set)

Množina (angl. set) je měnitelnou neuspořádanou (tj. nezáleží na pořadí) kolekcí jedinečných (nemohou se opakovat) hodnot neměnitelného datového typu. Množinu o jednom a více prvcích vytvoříme uzavřením hodnot oddělených čárkou do složených závorek. Prázdná množina vznikne voláním `set()` bez argumentů. Prázdnými složenými závorkami {} se totiž vyjadřuje prázdný slovník.[38]

Příklad deklarace prázdného slovníku a prázdné množiny:

```
>>> A = {}
>>> type(A)
<class 'dict'>
>>> B=set()
>>> type(B)
<class 'set'>
```

Pro standardní množinové operace je možné použít jak metodu, tak operátor (viz tab. 3.8). Oba zůsoby vytváří novou množinu, původní množiny A a B zůstávají nezměněny. Metody pro množinové operace akceptují v argumentu jakýkoliv iterovatelný typ, zatímco příslušné operátory umožňují pouze množinový typ. [43] Existuje také metoda pro symetrický rozdíl (*set*`.symmetric_difference()`), ten ale není obsahem výuky středoškolské matematiky.

Metoda	Ope-rátor	Název	Matemat. symbolické označení	Definice
`A.union(B)`	|	sjednocení	A ∪ B	množina všech prvků, které patří alespoň do jedné z množin A, B
`A.intersection(B)`	&	průnik	A ∩ B	množina všech prvků, které patří do množiny A a zároveň do množiny B
`A.difference(B)`	-	rozdíl	A – B	množina všech prvků, které patří do množiny A a zároveň nepatří do množiny B

Tab. 3.8: Běžné množinové operace (A, B – množiny) [39, 43]

Srovnání použití operátorů a metod pro množinové operace:

```
>>> A={1,2,3,4,5}
>>> B={4,5,6,7,8,9}
>>> A&B                              >>> A.intersection(B)
{4, 5}                              {4, 5}
>>> A|B                              >>> A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8, 9}         {1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> A-B                              >>> A.difference(B)
{1, 2, 3}                           {1, 2, 3}
>>> B-A                               >>> B.difference(A)
{8, 9, 6, 7}                        {8, 9, 6, 7}
```

Další operátory, metody a funkce [43]

x `in` *set*	test příslušnosti (je *x* prvkem množiny?)
x `not in` *set*	negace testu příslušnosti
`for` *x* `in` *set*	náhodné procházení množinou
`len(`*set*`)`	počet prvků množiny
`set()`	vytvoří množinu z objektu v argumentu
set`.add(`*prvek*`)`	přidá *prvek* do množiny (pokud v ní již není)
set`.remove(`*prvek*`)`	odstraní *prvek* z množiny, není-li v množině vyvolá výjimku `KeyError`
set`.discard(`*prvek*`)`	odstraní *prvek* z množiny, při nepřítomnosti *prvku* výjimku nevyvolá
set`.pop()`	odstraní náhodný prvek z množiny, je-li množina prázdná vyvolá výjimku `KeyError`
set`.clear()`	odstraní všechny prvky z množiny
set1`.update(`*set2*`)`	do množiny *set1* přidá prvky ze *set2* (tak, aby nedošlo k duplicitě prvků)
relační operátory	pro zjištění, zda je jedna množina podmnožinou druhé a (ne)rovnosti dvou množin

Python nabízí i neměnitelný množinový typ frozenset, na který nemohou být aplikovány operace a metody měnící obsah množiny. [43]

3.3.7 Slovník (dict)

Slovník je měnitelná neuspořádaná kolekce dvojic klíč: hodnota. Klíče musí být neměnného typu a v rámci jednoho slovníku jedinečné. Hodnoty se mohou opakovat a být libovolného typu. [40, 43] Všechny klíče nemusí být v jednom slovníku stejného typu, totéž platí i pro hodnoty. [38] K hodnotám slovníku přistupujeme použitím jejich klíče, prvky slovníku nemají definované pořadí. Slovník zapisujeme do složených závorek jako páry klíč: hodnota oddělené čárkou, prázdný slovník jako prázdné složené závorky. [37]

operátory, metody a funkce [36, 37, 40, 43]

`len(`*`dict`*`)`	počet párů klíčových hodnot slovníku
`dict``[`*`klic`*`]`	výběr hodnoty pomocí klíče
`dict``[`*`klic`*`] = `*`hodnota`	přidání páru (pokud již klíč existuje, dojde k přepsání hodnoty)
`del` *`dict`*`[`*`klic`*`]`	smaže celý pár (klíč i hodnotu) podle zadaného klíče
`klic` `in` *`dict`*	test příslušnosti (je klíč ve slovníku?)
`klic` `not in` *`dict`*	negace testu příslušnosti
`dict``.keys()`	vrátí pohled se všemi klíči slovníku
`dict``.values()`	vrátí pohled se všemi hodnotami slovníku
`dict``.items()`	vrátí pohled se všemi dvojicemi (klíč: hodnota) slovníku
`dict``.clear()`	odstraní všechny páry

Výsledky metod `keys()`, `values()`, `items()` jsou tzv. „view object", dynamické pohledy, které reflektují změny slovníku. Pro převod na statický typ seznam je možné použít funkci `list()`. [43]

3.3.8 None

`None` je zvláštní datový objekt, který reprezentuje prázdnou hodnotu. V celém systému Python existuje pouze jediný objekt `None` (všechny odkazy ukazují na tentýž objekt). `None` je vracen funkcemi, které explicitně nevrací žádnou hodnotu. `None` není 0, ani `False`, ani prázdný řetězec; při porovnání s čímkoliv jiným než s `None` dostaneme `False`. [36, 38]

3.3.9 Souhrn a srovnání některých vlastností datových typů

Na závěr kapitoly o datových typech jsou uvedeny souhrnné tabulky pro přehled měnitelnosti datových typů (tab. 3.10) a priority operátorů (tab. 3.9) používaných při výuce na SŠ. Postup při vyhodnocování výrazů je uveden v kap. 3.3.1.

Operátor	Popis	Priorita
(), [], {}	vazba nebo n-tice, seznam, slovník, množina	nejvyšší
`x[index]`, `x[index1:index2]`, `x(argumenty)`, `x.metoda`	indexování, úseky, volání funkce, metody	
**	umocňování	
-	aritmetická negace (změna znaménka)	
*, /, //, %	násobení, dělení, zbytek	
+, -	sčítání, odčítání	
`in, not in,` `is, is not,` <, <=, >, >=, !=, ==	testy příslušnosti testy identity relační operátory	
`not x`	logická negace	
`x and y`	logický součin	
`x or y`	logický součet	
`if – else`	podmíněné větvení	nejnižší

Tab. 3.9: Priorita operátorů [17, 37, 43]

Datový typ	Měnitelnost
Čísla	ne
Bool	ne
Řetězce	ne
Seznamy	ano
N-tice	ne
Množiny	ano
Slovníky	ano

Tab. 3.10: Měnitelnost datových typů [37]

Je možné změnit seznamy, množiny (set nikoliv frozenset) a slovníky, ale už ne n-tice, řetězce, logické hodnoty bool a čísla. Nelze změnit hodnotu neměnitelných typů, změnou samotné hodnoty. Jednička zůstane jedničkou, True zůstane True. [37] Naproti tomu je možné změnit hodnotu proměnné (dokonce i jejího typu) vytvořením nového odkazu (viz kap. 3.2).

Kombinace faktu, že seznamy, množiny a slovníky mohou být modifikovány a toho, že proměnné odkazují na objekty, vede k problémům se současnou změnou tzv. mělkých kopií u těchto datových typů. [40]

Příklad modifikace mělké kopie:

```
>>> a = [0, 1, 2, 3]
>>> b = a
>>> a[1] = 5
>>> b
[0, 5, 2, 3]
```

Mělkou kopii lze vytvořit i úplným řezem `x[:]`. Hlubokou kopii, která je na svém originálu nezávislá lze vytvořit pomocí funkce `deepcopy()` z modulu `copy`. [40]

3.4 Zásobník a fronta

Zásobník ukládá data tak, že poslední vložený prvek jde na výstup jako první. Tento způsob bývá označován LIFO z anglického „Last In – First Out“. Ze seznamu lze vytvořit zásobník pomocí metod `append()` a `pop()`. Využívá se především k dočasnému uložení dat během výpočtu. [43, 46]

Fronta ukládá data tak, aby prvek, který byl uložen jako první, byl také jako první vybrán. Tento způsob bývá označován FIFO z anglického „First In – First Out“. Ze seznamu lze vytvořit frontu pomocí metody `append()`, pro přidání prvku na konec fronty a metody `pop()` s indexem 0 (tedy `pop(0)`), pro odebrání prvního prvku. Fronta má využití například ve vyrovnávacích pamětech pro datové toky. [43, 46]

3.5 Vstup a výstup

Pro vstup dat z klávesnice slouží vestavěná funkce `input()`, kde nepovinným argumentem je řetězec, který se vypisuje jako výzva [40, 42]:

```
>>> vstup = input('Zadej cislo:')
Zadej cislo:
```

Funkce čeká, až uživatel zadá odpověď, kterou ukončí stiskem klávesy Enter. Vložené údaje jsou uloženy jako řetězce. Pokud je požadována číselná hodnota musí se konvertovat [40, 42]:

```
>>> vstup_cislo = int(input('Zadej cislo:'))
```

Příkaz `print()` vypíše data uživateli, přičemž v argumentu jsou zobrazované hodnoty, kterých může být více. Po přidání dalšího argumentu ve tvaru `end=''` bude následující výpis pokračovat na stejném řádku (v Pythonu 2.7 plní tuto funkci čárka za příkazem). Pokud `end` není zadán, je nastaven na `\n`.[43]

Příklad výpisu na jednom řádku:

```
>>> for i in range(1,10):
        print(i, end='')

123456789
```

3.6 Řídící struktury

Při zpracovávání programu se provádí jeden příkaz za druhým po řádcích. Tento postup lze odklonit řídící strukturou, jako je podmíněná větev nebo příkaz cyklu. Odklonění je možné i voláním funkce či metody a při vyvolání výjimky. [40] Výjimky se neobjevují v náplni výuky vybraných gymnázií (viz kap. 2.1.).

3.6.1 Podmíněné větvení

Podmíněné větvení umožňuje Python přostřednictvím příkazu `if`. Začíná částí `if` („jestliže“), následuje žádná nebo více klauzulí `elif` („jestliže ne, tak zkus jestli“) a na závěr můžeme zařadit volitelnou klauzuli `else` („jestliže ne, tak“). [37, 40]

Obecná syntaxe:

```
if logický_výraz1:
    příkazy1
elif logický_výraz2:
    příkazy2
…
elif logický_výrazN:
    příkazyN
else:
    další_příkazy
```

Pokud je `logický_výraz1` pravdivý, provedou se `příkazy1` a `if` se ukončí. Jinak se postupně prochází jednotlivé logické výrazy částí `elif`, dokud není některá z nich splněna a provedeny její příkazy. Pokud není splněna žádná z nich, provedou se `další_příkazy` části `else`. [37]

Zkrácený příkaz if … else na jediný podmíněný výraz:

```
příkaz1 if logický_výraz else příkaz2
```

V případě, že se `logický_výraz` vyhodnotí na `True`, bude výsledkem `příkaz1`, v opačném případě `příkaz2`. [40]

Pokud v určité části podmíněného větvení nechceme provést žádnou akci, můžeme použít příkaz `pass` („nedělej nic“). [40]

V Pythonu neexistuje příkaz typu `case` a `switch`. [37]

3.6.2 Cyklus while

While je cyklus, který se neustále (cyklicky) vrací na začátek, dokud je splněna podmínka (logický výraz).

Obecná syntaxe:

```
while logický_výraz:
    příkazy_while
else:
    příkazy_else
```

Cyklus while spouští vnořený blok `příkazy_while`, dokud je `logický_výraz` na začátku vyhodnocen jako `True`. Pokud je `logický_výraz` hned napoprvé nepravdivý, tak se `příkazy_while` nespustí ani jednou. Pokud je `logický_výraz` vyhodnocen na `False`, cyklus se ukončí a provede se část `else`, je-li uvedena. [37, 40]

Volitelná část `else` zřejmě nebude při výuce na střední škole příliš využívaná, je proto vhodnější použití následující zkrácené syntaxe.

Zkrácená syntaxe, bez využití části else:

```
while logický_výraz:
    příkazy
```

Pokud dojde k nekonečnému cyklu je možné jej v interpretu přerušit stiskem Ctrl+C nebo restartovat shell stiskem Ctrl+F6.

3.6.3 Cyklus for

Cyklus for v Pythonu prochází posloupnosti, pracuje tedy nad řetězci, seznamy, n-ticemi nebo dalšími iterovatelnými objekty. [40]

Obecná syntaxe:

```
for proměnná in posloupnost:
    příkazy_for
else:
    příkazy_else
```

Část *příkazy_for* se postupně provede pro každý prvek posloupnosti, poté bude provedena případná část `else`. Nepovinná část `else` se v praxi, obdobně jako u cyklu while, často nevyužívá. [36]

Příkazy jednotlivých částí složených příkazů if, while a for jsou označeny pomocí odsazení. V dokumentaci jazyka Python je doporučené používat čtyřmezerové

odsazení bez tabulátorů. Jednotlivé řádky by neměly být delší než 79 znaků. Zalamovat řádky můžeme pomocí zpětného lomítka „\“. [36, 40, 43]

3.6.4 Range

Typ range představuje neměnitelnou posloupnost čísel používanou ve for cyklech. [43]

Zkrácená syntaxe:
`range(stop)`

`range(n)` vytvoří posloupnost čísel od 0 do n-1, kde n musí být typu int.

Obecná syntaxe:
`range(start, stop[, step])`

`range(a, n, k)` vytvoří posloupnost čísel od čísla `a` do `n-1` včetně, po krocích `k`, kde `k` je volitelné (pokud není zadáno, je nastaveno na 1) a čísla `a`, `n`, `k` jsou typu int. Tento zápis umožňuje definovat i klesající posloupnosti za použití záporného `k`. [43]

Posloupnosti typu range je možné převést na seznam či n-tici pomocí funkce `list()` a `tuple()`. Tímto způsobem je i možné vytvářet seznamy a n-tice: [40]

```
>>> list(range(4))
[0, 1, 2, 3]
>>> tuple(range(2,10,3))
(2, 5, 8)
```

3.6.5 Break, continue, pass u cyklů

Příkaz `break` ukončuje cyklus, tedy přepne tok programu na příkaz následující za nejvnitřnějším cyklem, v němž se nachází. Příkaz `continue` ukončuje aktuální iteraci nejbližšího vnějšího cyklu (cyklus pokračuje následující iterací). Příkaz `pass` nedělá vůbec nic. Část `else` se u cyklů provede jen pokud nebyl cyklus ukončen přes `break`. [37, 40, 45] Výše uvedené příkazy patří na střední škole spíše mezi rozšiřující učivo.

3.7 Funkce

Funkce je blok příkazů, které můžeme volat opakovaně a kterému můžeme předávat argumenty. [37]

Obecná syntaxe:

```
def název_funkce (argumenty):
   příkazy
   return návratová_hodnota
```

Syntaxe argumentu s výchozí hodnotou:

```
def název_funkce (argument = výchozí_hodnota):
```

Funkce jsou definovány příkazem `def`. Argumenty jsou volitelné, pokud je jich více oddělují se čárkou. Argumenty mohou mít i výchozí hodnotu, pak se stávají nepovinnými. Příkaz `return` ukončuje běh funkce a uvozuje výraz, který je vracen jako návratová hodnota. Pokud `return` není uveden, nebo nemá žádný argument, vrátí funkce hodnotu `None`. [36, 37, 40]

3.8 Rekurze

Rekurzivní funkce jsou takové funkce, které volají samy sebe. Každá přímo rekurzivní funkce by měla obsahovat základní případ, pro který je definován výsledek a rekurzivní případ. Základní případ zastavuje rekurzi. Rekurzivní funkce mohou být náročné na zdroje, nicméně některé algoritmy lze pomocí nich vyjádřit nejpřirozeněji. [40, 46]

3.9 Moduly

Modul je samostatný soubor definující skupinu funkcí Pythonu nebo jiných objektů pro opakované použití. [36, 37]

Objekty modulu lze importovat více způsoby:

```
import jmenomodulu
```

vyhledá modul *jmenomodulu*, zanalyzuje jeho obsah a zpřístupní jej. Při použití objektů modulu je třeba uvést i jméno modulu. Např:

```
>>> import math
>>> math.cos(0)
1.0
>>> math.pi
3.141592653589793
```

Tento způsob je i bezpečnější, protože nehrozí chyby způsobené kolizí jmen. [40]

```
from jmenomodulu import jmeno1, jmeno2, …
```

zpřístupní objekty jmeno1, jmeno2, … bez nutnosti uvádět jméno modulu. [36]

```
from jmenomodulu import *
```

naimportuje všechny objekty daného modulu bez nutnosti uvádět jméno modulu při jejich použití. Hvězdička (*) zastupuje jména všech objektů definovaných v modulu. Pokud je v našem kódu či v jiném importovaném modulu objekt stejného jména, dojde k chybě způsobené kolizí jmen. [36]

```
>>> from math import pi, cos
>>> pi
3.141592653589793
>>> cos(0)
1.0
```

Funkce `dir(jmenomodulu)` zobrazí seznam všech jmen definovaných modulem v argumentu. Funkce `help(objekt)` zavolá v interaktivním režimu nápovědu pro objekt v argumentu. [43]

3.9.1 Modul math

Vestavěný modul math poskytuje matematické funkce a konstanty, jako je například odmocňování, goniometrické funkce a konstanta π. V tab. 3.11 jsou uvedeny konstanty a funkce, které se vyskytují ve středoškolské matematice. Funkce arkus sinus (`math.asin()`), arkus kosinus (`math.acos()`) a arkus tangens (`math.atan()`) většinou nejsou na střední škole vyučovány přímo, ale žáci umí k hodnotám sinus, kosinus a tangens zjistit příslušné úhly.

Funkce modulu math	Popis
`math.sin()`	vrátí sinus úhlu zadaného v radiánech
`math.cos()`	vrátí kosinus úhlu zadaného v radiánech
`math.tan()`	vrátí tangens úhlu zadaného v radiánech
`math.asin()`	vrátí velikost úhlu z hodnoty sinu (v radiánech)
`math.acos()`	vrátí velikost úhlu z hodnoty kosinu (v radiánech)
`math.atan()`	vrátí velikost úhlu z hodnoty tangens (v radiánech)
`math.radians(s)`	převede číslo *s* ze stupňů na radiány
`math.degrees(r)`	převede číslo *r* z radiánů na stupně
`math.pi`	konstanta π
`math.e`	konstanta e
`math.ceil(x)`	vrátí nejmenší celé číslo větší nebo stejné jako *x*
`math.floor(x)`	vrátí největší celé číslo menší nebo stejné jako *x*
`math.trunc(x)`	vrátí celou část *x* (typu int)
`math.pow(x, y)`	vrátí x^y
`math.sqrt(x)`	vrátí $\sqrt{x}$
`math.exp(x)`	vrátí e^x
`math.log(x, b)`	vrátí $\log_b x$ (b je volitelné, výchozí hodnota je e)
`math.log10(x)`	vrátí $\log_{10} x$
`math.factorial(x)`	vrátí $x!$

Tab 3.11: Funkce a konstanty modulu math využitelné na střední škole [40, 43]

3.9.2 Modul datetime

Vestavěný modul datetime umožňuje vytvářet datum a čas a pracovat s nimi. Práce s datem a časem je složité téma [40], proto jsou uvedeny jen základní funkce a způsoby vytvoření dat.

Některé funkce modulu datetime [43]:

`datetime.date(rok, měsíc, den)`	vytvoření data
`datetime.date.today()`	dnešní datum
`datum``.weekday()`	den v týdnu zadaného data vyjádřeného celým číslem, kde pondělí je 0 a neděle je 6.
`datum1 - datum2`	počet dnů mezi zadanými daty
`datum1 < datum2`	zda `datum1` nastalo dříve než `datum2`

3.9.3 Modul random

Vestavěný modul random je určen pro různé způsoby generování náhodných čísel.

Některé funkce modulu random [43]:

`random.randrange(`*`start, stop, step`*`)`	náhodné celé číslo z `range(`*`start, stop [, step]`*`)`, pokud je *`step`* rovno 1 nebo není zvoleno, jedná se o náhodné celé číslo z intervalu $\langle start, stop)$, pokud není zvolen ani start, pak je z $\langle 0, stop)$.
`random.randint(`*`a, b`*`)`	náhodné celé číslo z intevalu $\langle a, b\rangle$
`random.random()`	náhodné reálné číslo v intervalu $\langle 0{,}0; 1{,}0)$
`random.choice(`*`posl`*`)`	náhodné číslo z neprázdné posloupnosti *`posl`*

3.10 Želví grafika

Populární způsob vysvětlení základů programování je pomocí želví grafiky, která byla součástí programovacího jazyka Logo, jenž vyvinuli Wally Feurzig a Seymour Papert v roce 1966. [43] Lze nalézt samostně, je však i standardní součástí Pythonu. Pomocí příkazů, je možné vykreslovat různé tvary a obrázky. Následuje výčet funkcí, které mohou být vhodné při řešení jednodušších příkladů na SŠ a těch, jež jsou použity v příkladech ve 4. kapitole.

Na počátku vždy: `import turtle`
`nazev_okna = turtle.Screen()`

Vytvoření a změna želvy [43]:

`jmeno_zelvy = turtle.Turtle()`	vytvoří novou želvu a pojmenuje ji zadaným jménem
`jmeno_zelvy.shape('novy_tvar')`	změní tvar želvy, možnosti: `'arrow'`, `'turtle'`, `'circle'`, `'square'`, `'triangle'`, `'classic'`
`jmeno_zelvy.stamp()`	otiskne želvu na plátno

Pohyb želvy [43]:

`jmeno_zelvy.forward(vzdalenost_v_pixelech)`	vpřed (lze i `.fd()`)
`jmeno_zelvy.backward(vzdalenost_v_pixelech)`	vzad (lze i `.bk()`, `.back()`)
`jmeno_zelvy.left(uhel_ve_stupnich)`	otočení vlevo o zadaný úhel (lze `.lt()`)
`jmeno_zelvy.right(uhel_ve_stupnich)`	otočení vpravo o zadaný úhel (lze `.rt()`)
`jmeno_zelvy.up()`	přestat kreslit (želva zvedne ocas)
`jmeno_zelvy.down()`	začít kreslit (želva položí ocas)
`jmeno_zelvy.color('barva_anglicky')`	změna barvy
`jmeno_zelvy.home()`	vrátit želvu na počátek (0,0)
`jmeno_zelvy.circle(r)`	vykreslí kružnici o poloměru
`jmeno_zelvy.fillcolor('barva_anglicky')`	zadání vyplňovací barvy
`jmeno_zelvy.begin_fill()`	před zahájením kreslení tvaru, který má být vybarven (počátek vybarvování)
`jmeno_zelvy.end_fill()`	vybarvení tvaru po skončení kreslení tvaru (konec vybarvování)
`jmeno_zelvy.speed(rychlost)`	rychlost želvy, argument buď číslo nebo řetězec (*`0:'fastest', 10:'fast', 6:'normal', 3:'slow', 1:'slowest'`*)
`jmeno_zelvy.goto(x, y)`	přejde na pozici [x, y] (v pixelech)

Práce s oknem [43]:

nazev_okna.bgcolor('*barva_okna*')	nastaví barvu okna
nazev_okna.bgpic('*název_obrázku*')	nastaví obrázek na pozadí
nazev_okna.clear()	smaže celý obsah okna (pozadí, kresby i želvy)

Dalším modulem využitelným ve výuce může být například `tkinter`, pro tvorbu jednoduchého grafického uživatelského prostředí. [43]

3.11 Práce se soubory

Pro čtení a zápis textových souborů se musí nejprve otevřít pomocí funkce `open()` a po ukončení práce opět zavřít použitím `close()`. [38]

Syntaxe funkce open [43]:

```
s = open('jmeno_souboru', 'mod')
```

Při výuce je třeba dát pozor, aby žáci pracovali v adresáři do kterého mají právo čtení a zápisu. Ve většině případů bude třeba zadat i cestu, kde je třeba použít escape sekvenci pro zápis zpětného lomítka, např.:

```
s = open('C:\\Users\\student\\Docs\\Pyt\\text.txt', 'w')
```

Soubor je možné otevřít v několika režimech (viz tab. 3.12), které jsou nepovinné. Pokud není zadán, je nastaven na `'r'`.[43] Pokud soubor neexistuje, bude v režimu `'w'` a `'a'` automaticky vytvořen. V režimu `'w'` je existující soubor přepsán.

režim	význam
`'r'`	pouze pro čtení
`'w'`	pouze pro zápis
`'a'`	pro přidání dat na konec souboru
`'r+'`	pro čtení i zápis

Tab. 3.12: Mody [43]

Metody pro čtení a zápis [17, 43]:

`soubor.write()`	zápis dat v argumentu do souboru
`soubor.read()`	soubor je načten jako jeden řetězec
`soubor.readline()`	načte a vrátí následující řádek řetězce
`soubor.readlines()`	načte a vrátí seznam jehož jednotlivé prvky jsou řádky souboru

Pokud se při čtení dostaneme na konec souboru, je všemi těmito metodami vracen prázdný řetězec.

Pro práci se soubory nabízí Python speciální příkaz `with`, který po skončení soubor uzavře a to i v případě pokud kód vyvolá výjimku. [38, 43]

Syntaxe příkazu with:

```
with open('jmeno_souboru', 'mod') as promenna:
    prikazy
```

Další možnosti získávání informací a manipulací se soubory a navíc i adresáři umožňuje modul `os`, což je zkratka pro operační systém. Pro práci s adresáři (např. získání obsahu určitého adresáře) lze využít také modul `glob`. Oba moduly jsou součástí standardní pythonovské knihovny. [38]

4. Příklady

Náplní této části jsou zejména řešené příklady, doplněné o metodické pokyny. Bylo snahou vytvořit netradiční úkoly, nebo alespoň zajímavější text. Zadání mnohdy vychází z jiných oblastí než z informatiky (např. fyzika, dějepis, literatura), čímž jsou rozvíjeny mezipředmětové vztahy. Takto stavěné úkoly bývají pro žáky zajímavější, neboť vychází ze skutečných jevů, či událostí. U těchto příkladů je vždy uvedena nápověda, aby i učitel, který nemá v těchto oblastech přehled, měl k dispozici všechny potřebné informace k jejich řešení. Pasáže citované z knih mají motivační charakter, nejsou nutnou podmínkou k vyřešení úlohy.

4.1 Doporučení pro zadávání a řešení příkladů

Každý učitel má své způsoby výuky programování a práce s žáky ve vyučovací hodině. Následující pravidla je třeba brát jako doporučení, která mohou zlepšit výuku. Nicméně existují situace, kdy některé zásady nemusí vést ke kladnému efektu (např. žák nejevící žádný zájem).

Doporučené zásady:

- Zadání by měli mít žáci neustále k dispozici, buď v tištěné nebo elektronické formě. Zadání příkladů přečíst nahlas, případně vysvětlit. Text mohou eventuelně číst i žáci, ale spíše po kratších úsecích a nevyvolávat dyslektiky.
- Pokud některému z žáků není určitá věc jasná a nedaří se mu pochopit náš výklad, vyzvat žáka, který tématu porozuměl, aby jej vysvětlil. Mnohdy dokáže problém vysvětlit srozumitelnějším jazykem.
- Různá řešení mohou být správná. Netrvat na uvedeném řešení.
- Diskutujte s žáky jejich způsoby řešení, veďte je k pochopení chyby, či vhodnosti použití.
- Cílem je naučit myslet a řešit problémy, nikoliv naučit se sadu příkazů zpaměti. Je možné žákům umožnit používat sešit, či „tahák“ s důležitými příkazy. S dokumentací by mohli mít zvláště slabší žáci problémy z důvodu jejich nedostatečné znalosti angličtiny.

4.2 Struktura řešených příkladů

Při řešení příkladů bylo vycházeno ze struktury funkce uvedené v kurzu Learn to Program: The Fundamentals [17]. Při výuce je nutné žákům vysvětlit strukturu funkce, její části a jejich význam. Také je důležité vysvětlit, zda jsou jednotlivé části povinné či volitelné, a důvody jejich uvádění, např. komentářů viz kap. 3.1. Bez náležitého výkladu by žáci mohli hůře pochopit řešené příklady.

Je zřejmé, že při výuce v běžné vyučovací hodině nebude dostatek času k takto podrobnému řešení. Nicméně by měl podobný postup zaznít alespoň ústně. V případných domácích úkolech je vhodné vyžadovat kompletní řešení.

Postup řešení

Celému postupu předchází zadání úkolu žákům a jeho přečtení. Následujícími body postupují žáci buď samostatně, nebo je jimi učitel provází (zvláště v prvních příkladech). Jednotlivé kroky jsou doplněny o aplikaci na ukázkový příklad.

Zadání úkolu: Spojené státy měří teplotu ve stupních Fahrenheita a Kanada ve stupních Celsia. Při cestování mezi oběma zeměmi je dobré mít převodní funkci. Napiš funkci, která převádí teplotu ze stupňů Fahrenheita na stupně Celsia.

1. **Příklady volání funkce** (doporučená část)
   - Uvědomte si, co by měla vaše funkce dělat.
   - Zadejte několik příkladů volání funkce.
   - Zvolte jméno funkce. Mělo by být krátkou odpovědí na otázku: "Co vaše funkce dělá?".

```
>>> prevod_na_celsia(32)
0
>>> prevod_na_celsia(212)
100
```

2. **Typová anotace** (doporučená část)
   - Uvědomte si jakého typu jsou vstupní parametry a vracená hodnota.

```
(number) -> number
```

3. **Hlavička** (povinná část)
   - Použijte zvolené jméno funkce a vyberte smysluplné názvy parametrů.

```
def prevod_na_celsia(fahrenheit):
```

4. **Popis** (doporučená část)
   - Uveďte popis pro lepší pochopení funkčnosti při pozdějším čtení.
   - V popisu se zmiňte o každém parametru.
   - Popište vracenou hodnotu.

```
Výstupem je teplota ve stupních Celsia odpovídající
zadané teplotě ve Fahrenheitově stupnici.
```

5. **Tělo** (povinná část)
   - Promyslete, jakým způsobem budete funkci řešit (zda použijete cykly, větvení, některý z modulů, ...)
   - Napište tělo vaší funkce.

```
return (fahrenheit - 32) * 5 / 9
```

6. **Test (**doporučená část)
    - Ověřte, zda funkce pracuje správně spuštěním příkladů, které jste vymysleli v prvním bodě.

Vše dohromady:

```
def prevod_na_celsia(fahrenheit):
    ''' (number) -> number

    Výstupem je teplota ve stupních Celsia odpovídající
    zadané teplotě ve Fahrenheitově stupnici.

    >>> prevod_na_celsia(32)
    0
    >>> prevod_na_celsia(212)
    100
    '''

    return (fahrenheit - 32) * 5 / 9
```

Výše uvedený popis je vhodný pro výuku. Obsahuje všechny kroky, které je nutné si uvědomit k vyřešení zadaného problému. Mnozí žáci zpočátku „neví co mají dělat“, tento návod jim může pomoci, protože obsahuje postup krok za krokem.

4.3 Řešené příklady v interaktivním režimu

Pro testování krátkých kódů či při zkoušení různých funkčností, zvláště v začátcích poznávání Pythonu, je vhodná práce v interaktivním rozhraní Pythonu. Skupina znaků >>> (tzv. prompt) naznačuje, že překladač je připraven přijímat instrukce. [42]

1) Změňte dané hodnoty výrazů využitím závorek. Ověřte výrazy v interaktivním režimu Pythonu.
    a) `4 * 8 - 7` z hodnoty 25 na 4
    b) `48 / 8 - 6` z hodnoty 0 na 24
    c) `10  - 3 * 2 ** 2` z hodnoty –2 na 16
    d) `1.5 + 0.5 * 2.6 + 3.4` z hodnoty 6,2 na 12
    e) vymyslete vlastní příklad

    Možné řešení:

    a) `4 * (8 - 7)`
    b) `48 / (8 - 6)`
    c) `(10 - (3 * 2)) ** 2`
    d) `(1.5 + 0.5) * (2.6 + 3.4)`

Metodická poznámka:
Cílem úkolu je procvičení priority operátorů a uvědomění si důležitosti závorek. Při řešení části d) Python spočítá místo 6,2 hodnotu 6.199999999999999. Diskutuje s žáky možné příčiny tohoto jevu (mnoho desetinných čísel je uloženo pouze jako přibližná hodnota, protože nemají odpovídající binární tvar [52]). Část e) si následně mohou žáci zadávat navzájem.
Používané pojmy:
Datový typ int a float, aritmetické operátory a jejich priorita.

2) Do proměnné x uložte řetězec ma. S maximálním využitím x, operátorů +, *, interních funkcí a indexování zapište jazykolam „Máma má málo máku.“ [47]. Zápis proveďte bez diakritiky a zachovejte velikost písmen.
Možné řešení:
```
>>> x = 'ma'
>>> x[0].upper() + x[1] + x + ' ' + x + ' ' + x + 'lo '
+ x + 'ku.'
'Mama ma malo maku.'
```

3) Do proměnné x uložte řetězec had. S maximálním využitím x, operátorů +, *, interních funkcí a indexování zapište jazykolam „Had hádá hadu záhadu.“ [47]. Zápis proveďte bez diakritiky a zachovejte velikost písmen.
Možné řešení:
```
>>> x = 'had'
>>> x[0].upper() + x[1:] + ' ' + x + 'a ' + x + 'u za'
+ x + 'u.'
'Had hada hadu zahadu.'
```

4) Do proměnné x uložte řetězec li. S maximálním využitím x, operátorů +, *, interních funkcí a indexování zapište jazykolam „Zeptej se Lili, nalili-li liliputáni liliím vodu!“ [47]. Zápis proveďte bez diakritiky a zachovejte velikost písmen.
Možné řešení:
```
>>> x = 'li'
>>> 'Zeptej se ' + x[0].upper() + x[1] + x + ', na' +
2*x + '-' + x + ' ' + 2*x + 'putan' + x[1] + ' ' + 2*x
+ x[1] + 'm vodu!'
'Zeptej se Lili, nalili-li liliputani liliim vodu!'
```

5) Do proměnné x uložte řetězec kapka, do y řetězec kapla. Zapište jazykolam „Kapka kapla, klapka klapla“ [47]. Zápis proveďte bez diakritiky a zachovejte velikost písmen.
   a) S maximálním využitím x, y, operátorů +, *, interních funkcí, indexování a výběrů úseků.
   b) Postupujte stejně jako v části a), ale pouze s využitím proměnné x.

Možné řešení:

```
>>> x = 'kapka'
>>> y = 'kapla'
```

část a)

```
>>> x[0].upper() + x[1:] + ' ' + y + ', ' + x[0] + 'l'
+ x[1:] + ' ' + y[0] + 'l' + y[1:]
'Kapka kapla, klapka klapla'
```

část b)

```
>>> x[0].upper() + x[1:] + ' ' + x[:3] + 'l' + x[1] +
', ' + x[0] + 'l' + x[1:] + ' ' + x[0] + 'l' + x[1:3] +
'l' + x[1]
```

Metodická poznámka k příkladům 2, 3, 4, 5:

V tomto příkladu si žáci vyzkouší manipulaci s řetězci. Možným rozšířením je tvorba vlastních příkladů žáky.

Používané pojmy v příkladech 2, 3, 4, 5:

Datový typ string, indexování, úseky řetězce, spojování řetězců, metoda `upper()`.

6) Do kurzu angličtiny chodí Lenka Rozkošná, Tomáš Rozbořil, Roman Cvrček, Dita Vařečková, Miloš Císař a Michal Nejezchleba. Kurz vaření navštěvuje Jiřina Řízková, Michal Nejezchleba, Dita Vařečková, Adéla Polívková a Marek Nesvačil. Uložte účastníky kurzu pomocí množin a zjistěte:
   a) osoby, které chodí do kurzu angličtiny i vaření;
   b) osoby, které chodí do angličtiny nebo do kurzu vaření;
   c) osoby, které chodí do kurzu vaření, ale nechodí do angličtiny;
   d) osoby, které chodí do angličtiny, ale nechodí do kurzu vaření.

Možné řešení:

```
>>> anglictina = {'Lenka Rozkosna', 'Tomas Rozboril',
'Roman Cvrcek', 'Dita Vareckova', 'Milos Cisar',
'Michal Nejezchleba'}
>>> vareni = {'Jirina Rizkova', 'Michal Nejezchleba',
'Dita Vareckova', 'Adela Polivkova', 'Marek Nesvacil'}
>>> anglictina.intersection(vareni)
{'Dita Vareckova', 'Michal Nejezchleba'}
>>> anglictina.union(vareni)
{'Roman Cvrcek', 'Marek Nesvacil', 'Michal
Nejezchleba', 'Tomas Rozboril', 'Lenka  Rozkosna',
'Adela Polivkova', 'Dita Vareckova', 'Jirina Rizkova',
'Milos Cisar'}
>>> vareni.difference(anglictina)
{'Adela Polivkova', 'Jirina Rizkova', 'Marek Nesvacil'}
>>> anglictina.difference(vareni)
{'Roman Cvrcek', 'Lenka Rozkosna', 'Tomas Rozboril',
'Milos Cisar'}
```

Metodická poznámka:
V tomto úkolu si žáci vyzkouší tvorbu množina množinové operace.
Používané pojmy:
Datový typ set, metody `intersection()`, `union()`, `difference()`.

7) Vytvořte slovník, kde klíče budou státy Evropské unie a jejich hodnoty letopočty přistoupení k EU. Zjistěte:
   a) počet členských států EU
   b) zda je Litva členem EU
   c) zda je Norsko členem EU
   d) rok přistoupení České republiky k EU

   Rozšíření: Vytvořte převrácený slovník, kde klíče budou letopočty a hodnoty seznamy státu, které v daném roce přistoupily k EU. Ověřte správnost výpisem. Následně zjistěte státy, které státy se staly členem EU společně s Českou republikou.

   Možné řešení:

```
>>> EU = {'Belgie':1952, 'Bulharsko':2007, 'Ceska
republika':2004, 'Dansko':1973, 'Estonsko':2004,
'Finsko':1995, 'Francie':1952, 'Chorvatsko':2013,
'Irsko':1973, 'Italie':1952, 'Kypr':2004, 'Litva':2004,
'Lotyssko':2004, 'Lucembursko':1952, 'Madarsko':2004,
'Malta':2004, 'Nemecko':1952, 'Nizozemsko':1952,
'Polsko':2004, 'Portugalsko':1986, 'Rakousko':1995,
'Rumunsko':2007, 'Recko':1981, 'Slovensko':2004,
'Slovinsko':2004, 'Spojene kralovstvi':1973,
'Spanelsko':1986, 'Svedsko':1995}            #[48]
>>> len(EU)
28
>>> 'Litva' in EU
True
>>> 'Norsko' in EU
False
>>> EU['Ceska republika']
2004
```

   Možné řešení rozšíření:

```
>>> letopocet = {}
>>> for stat in EU:
        rok = EU[stat]
        if not (rok in letopocet):
            letopocet[rok]=[stat]
        else:
            letopocet[rok].append(stat)
>>> letopocet
```

```
{1952: ['Nizozemsko', 'Belgie', 'Lucembursko',
'Italie', 'Nemecko', 'Francie'], 1986: ['Spanelsko',
'Portugalsko'], 2004: ['Lotyssko', 'Ceska republika',
'Litva', 'Polsko', 'Slovinsko', 'Slovensko',
'Estonsko', 'Malta', 'Madarsko', 'Kypr'], 1973:
['Irsko', 'Spojene kralovstvi', 'Dansko'], 2007:
['Rumunsko', 'Bulharsko'], 1995: ['Rakousko',
'Svedsko', 'Finsko'], 2013: ['Chorvatsko'], 1981:
['Recko']}
>>> letopocet[2004]
['Lotyssko', 'Ceska republika', 'Litva', 'Polsko',
'Slovinsko', 'Slovensko', 'Estonsko', 'Malta',
'Madarsko', 'Kypr']
```

Metodická poznámka:

Úkol je zaměřen na tvorbu slovníku a práci s ním. Členské státy EU a letopočty jejich přistoupení k EU mohou žáci vyhledat na Internetu. Častou chybou je přepisování hodnot v nově tvořeném slovníku:

```
>>> spatny_letopocet={}
>>> for stat in EU:
        rok = EU[stat]
        spatny_letopocet[rok]=[stat]
>>> spatny_letopocet
{1952: ['Francie'], 1986: ['Portugalsko'], 2004:
['Kypr'], 1973: ['Dansko'], 2007: ['Bulharsko'], 1995:
['Finsko'], 2013: ['Chorvatsko'], 1981: ['Recko']}
```

Používané pojmy:

Datový typ dict, funkce `len()`, operátor `in`, výběr hodnoty ; ve 2. části: datový typ list, metoda `append()`, cyklus `for`.

8) Vytvořte seznam států Evropské unie a pomocí cyklu for je všechny vypište:
   a) pod sebe
   b) vedle sebe oddělené čárkou a mezerou

Možné řešení:

```
>>> EU=['Belgie', 'Bulharsko', 'Ceska republika',
'Dansko', 'Estonsko', 'Finsko', 'Francie',
'Chorvatsko', 'Irsko', 'Italie', 'Kypr', 'Litva',
'Lotyssko', 'Lucembursko', 'Madarsko', 'Malta',
'Nemecko', 'Nizozemsko', 'Polsko', 'Portugalsko',
'Rakousko', 'Rumunsko', 'Recko', 'Slovensko',
'Slovinsko', 'Spojene kralovstvi', 'Spanelsko',
'Svedsko']          #[48]
```

část a)

```
>>> for stat in EU:
        print(stat)
```

část b)

```
>>> for stat in EU:
        print(stat , end=', ')
```

Metodická poznámka:

Cílem je prakticky vyzkoušet funkci `print`. Členské státy EU mohou žáci vyhledat na Internetu.

Používané pojmy:

Datový typ list, funkce `print()`, operátor `in`, cyklus `for`.

9) Importujte modul pro želví grafiku. Vytvořte čtverec vyplněný modrou barvou, při řešení použijte vhodný cyklus.

Možné řešení:

```
>>> import turtle
>>> wn=turtle.Screen()
>>> zelva = turtle.Turtle()
>>> zelva.fillcolor('blue')
>>> zelva.begin_fill()
>>> for i in range(0,4):
            zelva.fd(90)

>>> zelva.end_fill()
```

Metodická poznámka:

Cílem je jednoduché procvičení základních postupů (tvorba okna, želvy, její pohyb, vybarvení obrazce) a uplatnění řídících struktur.

Používané pojmy:

Import modulu, želví grafika, funkce pro tvorbu okna, želvy a manipulaci s ní, cyklus `for`, funkce `range()`.

4.4 Řešené příklady v textovém editoru

Program můžeme zapsat do souboru (tzv. skriptu) použitím textového editoru. Soubory s kódem v Pythonu mají příponu .py. Vytváření i spouštění skriptů umožňuje IDLE. Vytvoření volbou New File na záložce File, nebo klávesovou zkratkou Ctrl+N. Spuštění zvolením Run Module na záložce Run, nebo klávesou F5. [42]

10) Přečtete si následující úryvek z knihy Dvacet tisíc mil pod mořem:

„V noci zuřivost bouře dále vzrostla. Tlakoměr klesl na 710 milimetrů, tedy jako roku 1860 na ostrově Réunionu. Za soumraku jsem spatřil na obzoru velkou loď zoufale bojující s bouří.“ [62]

Vytvořte funkci, která vyjádří hodnotu tlaku zadnou v milimetrech rtuťového sloupce v Pascalech.

Nápověda k řešení:

1 torr = 1 mm Hg ≈ 133,322 Pa

Metodická poznámka:
Jednoduchý úkol k procvičení základní tvorby funkce. Normální atmosférický tlak u mořské hladiny je 101 325 Pa. Před bouřkou jeho hodnota klesá. Z tohoto jevu se v románu předpovídá bouře.
Používané pojmy: Aritmetické operátory.
Název souboru s možným řešením: torr_na_pascal.py

11) Přečtete si následující úryvek z knihy Moře, sníh a velkoměsta od Jacka Londona:
„*Tui Tulifau byl králem nejen každým palcem své výšky, ale také každou vteřinu svého života. Když už vládl padesát osm let a pět měsíců, byl teprve padesát osm let a tři měsíce stár. Vládl tudíž o pět miliónů vteřin déle, než dýchal, neboť byl korunován o dva měsíce dřív, než se vůbec narodil.*“ [27]
   a) Vytvořte funkci, která převede časové období z měsíců na vteřiny. Uvažujte délku měsíce 30 dnů.
   b) Ověřte údaje v úryvku.

   Metodická poznámka:
   Jednoduchý příklad k procvičení základní tvroby funkcí a převodů jednotek času.
   Používané pojmy: Aritmetické operátory.
   Název souboru s možným řešením: sekundy.py

12) Přečtete si následující úryvek z knihy Dvacet tisíc mil pod mořem:
„Kapitán, nestaraje se o mou přítomnost, otevřel skříň, zřejmě nedobytnou pokladnu, plnou prutů. Byly to zlaté pruty. Odkud pocházel tento drahý kov, který představoval obrovskou částku? Jak k němu kapitán přišel a co s ním bude dělat? Neřekl jsem ani slovo. Jen jsem přihlížel. Kapitán Nemo bral prut po prutu a skládal je pečlivě do bedny, až ji celou naplnil. Odhadoval jsem její obsah na více než tisíc kilogramů zlata, což představovalo částku skoro pět miliónů franků.“ [62]
   a) zjistěte na Internetu cenu 1 kg zlata pro dnešní den.
   b) Vytvořte funkci, která zjistí aktuální cenu zlata v Kč. Zjistěte pomocí této funkce cenu 1000 kg zlata dnes.

   Metodická poznámka:
   Jednoduchý příklad k procvičení základní tvorby funkce. V řešení je použita hodnota 802 587,30 [16], kterou je potřeba změnit dle aktuální ceny.
   Používané pojmy: Aritmetické operátory, vyhledávání informací.
   Název souboru s možným řešením: zlato.py

13) **Bouřka**
   Jestliže při bouřce nastane blesk, vidíme nejprve záblesk a teprve poté jej slyšíme. Z časové prodlevy mezi zábleskem a hromem lze zjistit, jak daleko od nás k blesku došlo. Lidová poučka říká, že pokud vydělíme čas v sekundách číslem 3, získáme vzdálenost bouřky v kilometrech. Vytvořte funkci, která bude tuto vzdálenost (v celých km) zjišťovat na základě času mezi tím, kdy blesk vidíme a slyšíme. Dále ověřte zda uvedená poučka platí. (Předpokládejte, že záblesk vidíme okamžitě.)

Nápověda k řešení:

Rychlost zvuku ve vzduchu je přibližně 340 m/s. Pro vzdálenost se použije vztah $s = v \cdot t$.

Metodická poznámka:

Jednoduchý úkol pro procvičení základní tvorby funkce. Žáci si nemusí uvědomit, že je třeba výslednou hodnotu ještě převést na km a zaokrouhlit.

Používané pojmy: Aritmetické operátory, funkce `round()`.

Název souboru s možným řešením: bourka.py

14) **Hmotnost atomu**

Relativní atomová hmotnost vyjadřuje hmotnost atomu v násobcích atomové hmotnostní konstanty. Vytvořte funkci, která podle relativní atomové hmotnosti určí hmotnost atomu v kilogramech.

Nápověda k řešení:

Relativní atomová hmotnost prvků je uvedna v periodické tabulce prvků, kterou je možné vyhledat i na Internetu (např. http://www.tabulka.cz/). Atomová hmotnostní konstanta je definována jako 1/12 klidové hmotnosti izotopu uhlíku ^{12}C. Její přibližná velikost je $u=1{,}66.10^{-27}$ kg. [3]

Metodická poznámka:

Jednoduchý příklad pro procvičení tvorby funkcí a zápis čísla v exponenciálním tvaru. Žáci mohou mít problém se zápisem čísla v tomto tvaru. Je třeba je upozornit, že desetinná čárka je zapisována tečkou a „krát deset“ je zapisováno pomocí e nebo E. Tedy číslo $1{,}66.10^{-27}$ je 1.66e-27.

Používané pojmy: Aritmetické operátory, zápis čísla v exponenciálním tvaru.

Název souboru s možným řešením: hmotnost_atomu.py

15) Přečtete si následující úryvek z knihy Dvacet tisíc mil pod mořem:

„*V hloubce čtrnácti tisíc metrů přeplul Nautilus hranici podmořského života, tak jako balón překonává stoupáním hranici pásma dýchatelného vzduchu. Dosáhli jsme hloubky šestnácti tisíc metrů. Boky Nautilu byly nyní vystaveny tlaku šestnácti set atmosfér, to znamená tlaku šestnácti set kilogramů na každý čtvereční centimetr svého povrchu.*“[62]

Vytvořte funkci, která bude udávat tlak v atmosférách v zadané hloubce pod mořskou hladinou.

Nápověda k řešení:

Tlak pod hladinou je součtem atmosférického tlaku na mořské hladině a hydrostatického tlaku v určité hloubce. Vztah pro výpočet hydrostatického tlaku je $p = h\rho g$.

1 atm = 101 325 Pa (také průměrný atmosférický tlak u mořské hladiny).

Hustota mořské vody (13 °C) = 1024 kg/m^3

g = 9,81 m/s^2

Metodická poznámka:
Je možné začít jednodušší variantou pro výpočet tlaku v Pascalech a navázat na něj převodem na atmosféry. V řešeném souboru je funkce `tlak_Pa()` řešena pomocí funkce `print()` a funkce `tlak_atm()` pomocí `return`. Diskutujte s žáky výhody a nevýhody jednotlivých variant. (Varianta s `return` bude vracet pouze hodnotu, se kterou bude možné dál pracovat. Zatímco variantu s `print` je možné rozšířit o výpis jednotek, ale nebude ji možné dále využít ve výpočtech.)
Používané pojmy: Aritmetické operátory, `return` vs. `print()`.
Název souboru s možným řešením: tlak.py

16) **Vzdálenost na číselné ose**
Napište funkci pro výpočet vzdálenosti dvou čísel na číselné ose.
Nápověda k řešení: Rozdíl čísel v absolutní hodnotě je roven jejich vzdálenosti na číselné ose.
Metodická poznámka:
Jednoduchý úkol pro procvičení základní tvorby funkce.
Používané pojmy: Aritmetické operátory , funkce `abs()`.
Název souboru s možným řešením: vzdalenost.py

17) Přečtete si následující úryvek z knihy Dvacet tisíc mil pod mořem:
„Na kapitánův rozkaz se šroub zastavil a bočné plochy byly postaveny svisle. Nautilus vylétl jako balón do vzduchu a stoupal strašnou rychlostí k hladině. Řítil se vodou s hlučnými záchvěvy. Okny jsme neviděli nic. Ve čtyřech minutách jsme prolétli třinácti tisíci metry, které nás dělily od hladiny oceánu. Vylétli jsme nad vodu jako létající ryby a dopadem zpět jsme zdvihli úžasně vysoký vodní sloup.“ [62]
Vytvořte funkci, která zjistí jakou průměrnou rychlostí se Nautilus pohyboval. Při zadání hodnot použijte jednotky, které jsou uvedeny v románu a rychlost uvádějte v km/h. Výsledek zaokrouhlujte na 2 destinná místa.
Nápověda k řešení:
1 m/s = 3,6 km/h
v = s/t
Metodická poznámka:
Jednoduchý úkol k procvičení aritmetických operátorů a zaokrouhlování i s dalšími argumenty. Je možné zvolit ještě jednodušší variantu požadováním rychlosti v m/s.
Používané pojmy: Aritmetické operátory, funkce `round()`.
Název souboru s možným řešením: rychlost.py

18) Přečtete si následující úryvek z knihy Dvacet tisíc mil pod mořem:
„Budou mi lidé věřit? Nevím; ostatně na tom nezáleží. Mohu však nyní tvrdit, že mám právo hovořit o mořích, v nichž jsem v necelých deseti měsících vykonal cestu dvacet tisíc mil dlouhou, což je osmdesát tisíc kilometrů, o podmořské cestě kolem světa, která mi odkryla tolik divů v Tichém oceánu, v Indickém oceánu, v Rudém

moři, ve Středozemním moři, v Atlantském oceánu a v severních i v jižních mořích!“ [62]

Překlad názvu je nesprávný, protože v originále jde o dvacet tisíc „lieues“, což je starobylá jednotka délky, česky zvaná „pěší hodina“ a odpovídá vzdálenosti, kterou člověk (kůň) ujde za jednu hodinu. V románu jednotka odpovídá francouzské metrické variantě a je dlouhá přesně 4000 metrů. [9]

Vytvořte funkci, která převede délku zadanou v kilometrech na dnešní délku v námořních mílích. Pomocí vytvořeného programu zjistěte, jaká hodnota by měla být v názvu románu. Pokud má výsledek hodnotu větší nebo rovnu 10 zaokrouhlujte na jednotky, pokud je menší tak na 1 destinné místo.

Nápověda k řešení:

námořní míle: 1 n mile = 1852 m = 1,852 km (přesně). [4]

Metodická poznámka:

Příklad kombinuje použití větvení `if` a funkce `round()`, jak s jedním, tak s dvěma argumenty.

Používané pojmy: Funkce `round()`, větvení `if`.

Název souboru s možným řešením: mile.py

19) **Století**

Často se setkáváme s přibližným určením událostí pomocí století. Např.: Karel IV. žil ve 14. století, škola pro 21. století atd.

a) Vymyslete, případně najděte na Internetu další příklady.
b) Vytvořte funkci, která dokáže k zadanému letopočtu určit správné století a vypíše jej.

Nápověda k řešení:

Do prvního století počítáme roky 1, 2, …, 100, do druhého století roky 101, 102, …, 200. Dvacáté století začalo 1. ledna 1901 a skončilo o půlnoci z 31. prosince 2000 na 1. ledna 2001. [10]

Metodická poznámka:

Úkol k procvičení volby vhodné operace. Je třeba použít celočíselné dělení, v opačném případě, bude století vypsáno v neobvyklém tvaru s desetinnou čárkou.

```
>>> stoleti(1800)                #“obyčejné“ dělení
18.0. stoleti
>>> stoleti(1800)                #celočíselné dělení
18. stoleti
```

Používané pojmy: Celočíselné dělení, dělení se zbytkem (modulo), větvení `if`, funkce `print()`, funkce `str()`.

Název souboru s možným řešením: stoleti.py

20) **Kolik dní jsem na světě?**

Všichni víme, kolik nám je let, ale víte kolik je vám dnů?

a) Vytvořte funkci, která zjistí kolik dní uplynulo od zadaného data.
b) Zjistěte nejen kolik je vám dnů, ale i kolik dnů je Česká republika samostatná.

Metodická poznámka: Je třeba žákům ukázat, jakým způsobem se aplikují funkce modulu datetime na proměnné. Je také vhodné ukázat, jakým způsobem zjistit všechny funkce modulu (`dir(datetime)`) a nápovědu k jednotlivým funkcím (`help(datetime.date)`).

Používané pojmy: Modul `datetime`, (funkce `dir()`, `help()`).

Název souboru s možným řešením: dny.py

21) **Den v týdnu**

Víte, který den v týdnu jste se narodili? Vytvořte funkci, která podle zadaného data zjistí, jaký den v týdnu právě byl. Zjistěte i na který den letos připadnou Vánoce a další významná data: 17. 11. 1989, 1. 1. 1993, 2. 12. 1805. Víte, k jakým událostem v těchto dnech došlo?

Metodická poznámka:

Pokud není do výuky zahrnut modul `datetime`, je možné využít soubor den_v_tydnu_zadani.py, který je předchystán tak, aby žák dotvořil jen jemu známé věci. Přesto je vhodné již vytvořené části okomentovat.

Události: sametová revoluce, vznik samostatné České republiky, bitva u Slavkova.

Používané pojmy: Větvení `if` (i s částí `elif`), modul `datetime`.

Název souboru s řešením: den_v_tydnu_reseni.py

22) **GPS**

Navigační systém GPS tvoří 24 družic, které obíhají Zemi ve výšce 20 200 km. Plánovaný evropský systém Galileo by měl být plně funkční v letech 2019/2020, jeho 30 družic bude obíhat ve výšce 23 222 km. [49, 50] Vytvořte 3 funkce. První zjistití rychlost družic, kterou musí mít, aby se udržely na kruhové dráze (v m/s). Druhá dráhu, kterou družice urazí během jednoho obletu kolem Země. A třetí dobu tohoto jednoho oběhu družice. Dobu oběhu vypište ve tvaru `x h y min z s`. Následně zjistěte uvedené veličiny pro navigační systém GPS i Galileo.

Nápověda k řešení:

Pro jednoduchost předpokládáme pohyb družice po kruhové dráze. Pak se družice musí pohybovat tzv. kruhovou rychlostí (také nazývanou první kosmickou rychlostí), při nižší rychlosti by družice dopadla na Zemi, při vyšší rychlosti by se pohybovala po elipse (vzdalovala a přibližovala by se od Země podobně jako komety). Pro kruhovou rychlost platí vztah $v = \sqrt{\frac{\kappa \cdot M_Z}{R_Z + h}}$, kde κ = 6,67.10^{-11} N.m^2.kg^{-2} je gravitační konstanta, M_Z = 5,98.10^{24} kg hmotnost Země, R_Z = 6378 km poloměr Země, h výška družice nad povrchem Země. Dráha jenoho oběhu družice se spočítá pomocí obvodu kruhu, tedy $o = 2\pi r$. Je třeba si uvědomit, že poloměr je zde součet poloměru Země a výšky družice nad Zemí. Doba oběhu $t = \frac{s}{v}$. [3]

Metodická poznámka:

V této úloze lze velmi dobře ukázat využití již vytvořené funkce v těle další funkce.

Používané pojmy: Modul `math`, funkce `sqrt()`, `pi`, `floor()` (dolní celá část), zápis čísla v exponenciálním tvaru.
Název souboru s možným řešením: druzice.py

23) **Určení stáří organických materiálů**

U archeologických nálezů biologického původu (kosti, rostliny, dřevo) lze určit stáří pomocí radiokarbonové metody. Vše živé obsahuje tentýž poměr ^{12}C a radioaktivního ^{14}C až do své smrti. Po smrti radioaktivní uhlík pomalu a nezadržitelně klesá s poločasem rozpadu 5730 let. Podle obsahu izotopu ^{14}C je možné přibližně zjistit dobu, která uplynula od smrti. Tato metoda je použitelná asi do stáří 50000 let, pak už je množství nerozpadlého ^{14}C tak malé, že metoda ztrácí přesnost. Tímto způsobem bylo zjišťováno i stáří Turínského plátna a koruny Karla Velikého. [65]

a) Vytvořte funkci, která z naměřeného množství ^{14}C zjistí stáří materiálu. Množství se udává v procentech vzhledem k původnímu množství (za života).
b) Obměňte funkci tak, aby byl výsledek vypsán celou větou.
c) K použití této metody je nezbytný biologický materiál. Jaký biologický materiál by mohl být v Turínském plátně a v koruně Karla Velikého?

Nápověda k řešení:

Stáří materiálu *t*, se určí pomocí vztahu $t = -\frac{T \cdot \ln(p)}{\ln 2}$, kde *T* je poločas rozpadu, p je počet procent částic ^{14}C v látce (100 % je počet částic v živém materiálu) vyjádřeném jako destinné číslo. Poločas rozpadu uhlíku ^{14}C je 5730 let. [3, 4] Jednotka veličiny *t* závisí na jednotce poločasu rozpadu *T*. Pokud bude *T* v letech, bude i *t* v letech.

Řešení c):

Turínské plátno – len, koruna - drahokamy jsou upevněny směsí včelího vosku a jílu.

Metodická poznámka:

Logaritmy se v matematice probírají ve vyšších ročnících, předem si ověřte, zda vaši žáci tuto problematiku znají. Je vhodné upozornit, že `math.log()` je přirozený logaritmus, pokud není zadán jiný základ.

Používané pojmy: Modul `math`, funkce `round()`, používání složitějších matematických operací.
Název souboru s možným řešením: stari.py

24) **Písmo zprava doleva**

Arabština a hebrejština se na rozdíl od našeho písma zapisuje zprava doleva, tedy „pozpátku“. Vytvořte funkci, která převrátí zadaný text a vypíše jej tak, jak by vypadal kdybychom i my psali obráceně:

a) s využitím operátoru [] a indexů;
b) bez využití operátoru [] a indexů. [59]

c) Vyhledejte na Internetu další jazyky, jejichž písmo je zprava doleva a státy, kde se s těmito jazyky můžete setkat.

Řešení části c: perština (fársijština) – Írán, Afgánistán, Pákistán, aj., [57]
urdština – Pákistán, Indie [58]

Metodická poznámka:

V řešených příkladech je v jedné z funkcí funkcí `pozpatkua` a `pozpatkub` použit pro výstup `print`() a v druhé `return`. Diskutujte s žáky, která má jaký výstup a v jakých situacích je která varianta výhodnější.

Používané pojmy: Vyhledávání informací, manipulace s řetězci, cyklus `while`, `return` vs. `print()`.

Název souboru s možným řešením: pozpatku.py

25) Francimor při útěku před loupežníky použil kouzlo, kterým se zahánějí pronásledovatelé:

„Nad tlupou loupežnickou se utvořil mrak. A z toho oblaku se počal hustě sypati hnědý prášek. Byl to šňupavý tabák. Počal loupežníkům i koním vnikati do nozder a celá tlupa i s koňstvem se dala do kýchání.

Kýchali, až se ohýbali, drželi se za břicha, chytali se za hlavy.

„Hepčí! – Řach! – Bimbo! – Eéééj uchněm!“ ozývalo se po celém lese.

Loupežníci si nestačili říkat: „Pozdrav pámbu – dejž to pámbu, děkuju!“ Byl to takový rámus a kravál, že si většího halasu aneb tartasu nedovedete představit.“ [64]

Vytvořte dvě funkce, první na kterékoliv ze slov „hepcí, rach, bimbo, eeeej uchnem“ odpoví „Pozdrav pambu“. Druhá na „pozdrav pambu“ odpoví „dejz to pambu, dekuju“. Funkci ošetřte tak, aby slova na vstupu bylo možné napsat malými i velkými písmeny.

Metodická poznámka:

V této úloze je možné s žáky vyzkoušet volání funkce, jejímž argumentem je opět funkce (např. `odpoved(kychani('hepci'))`).

Používané pojmy: Metoda `lower()`, větvení `if`, operátor `in`.

Název souboru s možným řešením: kychani.py

26) Edudant a Francimor vyváděli ve škole různé kousky. O jednom z nich se dozvíte v následujícím úryvku:

„Naše slečna učitelka napsala na tabuli úlohu pod názvem „Vyjděme si na procházku“.

„Je krásně, vyjděme si na procházku do blízkého lesa. Ptáčkové tam libě pějí. Potůček bublá. Matinka nám dá s sebou džbán a my budeme sbírat jahody. Také tam rostou květinky a my si uvijeme věneček.“

Učitelka napsala diktát a děti měly z tabule opisovat. Když dopsala, sedla si ke stolu a dívala se do třídy.

Najednou vidí, že děti nepíší, ale vyvalují oči, a pak spustili veliký pokřik. Nevěděla, co se děje, ale když celá třída ukazovala prstem na tabuli, obrátila se a s úžasem uviděla, co se stalo.

Na tabuli na místě slova „les“ vyrostl smrček a místo slova „ptáčkové“ uviděla jednoho kosa a jednoho stehlíka, kteří zpívali z plna hrdla. Místo slova „potůček“ vytékal z tabule pramének čisté vody a zurčel jako skutečný lesní potůček. Tabule se zapestřila různými květinami, které sami od sebe svinuly ve věneček.“ [64]

Vytvořte funkci, která slova les, ptáčkové, potůček nebo kytičky (bez diakritiky) vykreslí pomocí různých symbolů na klávesnici. Pokud je zadáno jiné slovo, zůstane nezměněno.

Metodická poznámka:

Úkol, při kterém žáci vymýšlí, jakým způsobem z různých symbolů vytvořit zadaný obrázek.

Používané pojmy: Větvení `if` (i s částí `elif`), funkce `print()`, možné využít klávesové zkratky různých symbolů.

Název souboru s možným řešením: tabule.py

27) **Krevní tlak**

Při kontrole u lékaře bývá měřen krevní tlak. Po jeho naměření se dozvíte dvě hodnoty, např.: 120/80. Vyšší hodnota je systolický tlak, kdy je krev vytlačena do aorty při stahu srdečního svalu. Nižší hodnota je diastolický tlak, při němž krev proudí smrštěním aorty. [5] Podle naměřených hodnot se určí, zda pacient netrpí vysokým krevním tlakem (tzv. hypertenzí).

a) Vyhledejte na Internetu, jakým hodnotám tlaku odpovídají stupně hypetenze.
b) Vytvořte funkci, která po zadání systolického a diastolického tlaku zjistí, zda pacient trpí hypertenzí (vysoký krevní tlak) a jak závažným stupněm.

Nápověda k řešení:

Kategorie krevního tlaku	Systolický tlak	Diastolický tlak
Optimální	< 120	< 80
Normální	120–129	80–84
Vysoký normální tlak	130–139	85–89
Mírná hypertenze	140–159	90–99
Středně závažná hypertenze	160–179	100–109
Závažná hypertenze	≥ 180	≥ 110

Tab. 4.1: Kategorie krevního tlaku [19]

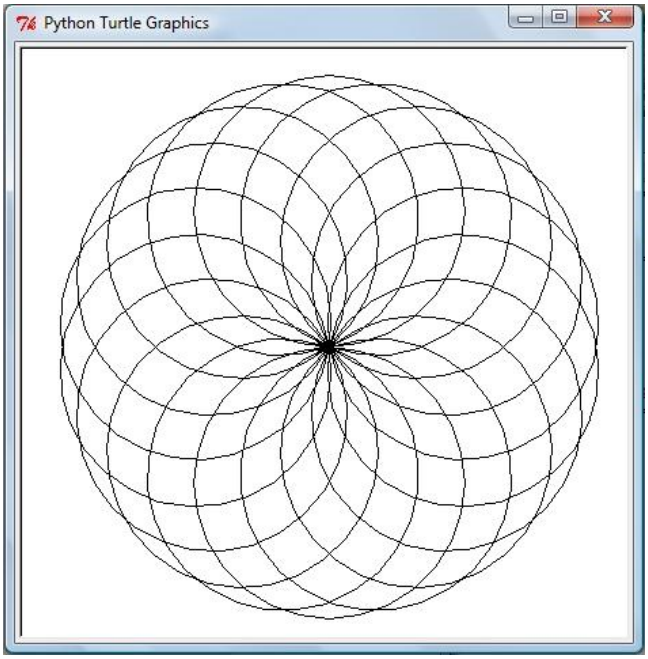
Je třeba zjistit, do které kategorie patří obě hodnoty. O výsledku rozhoduje nejhorší možnost.

Metodická poznámka:

Obtížnější cvičení, využívající několik větvení `if`. Je možné, že žáci naleznou různé kategorie a různé hranice pro kategorie krevního tlaku. Diskutujte, které zdroje jsou vhodnější k použití. Řešte s žáky, jakým způsobem zjistit správnou kategorii ze dvou hodnot. Příklad je možné rozšířit i o určení nízkého krevního tlaku.

Používané pojmy: Větvení `if` (i s částí `elif`), funkce `max()`, vyhledávání a volba vhodného zdroje informací.
Název souboru s možným řešením: krevni_tlak.py

28) Vytvořte funkci, která pomocí želví grafiky (modul turtle) vykreslí tvar uvedený na obrázku 4.1. Její vstupní argument bude barva (anglicky), kterou bude tvar vykreslen.



Obr. 4.1: Ukázka výstupu funkce příkladu č. 28

Metodická poznámka:
Diskutujte s žáky jakými způsoby mohl obrázek vzniknout. Pokuste se je navést na využití cyklu `for`. Za účelem rychlejšího vykreslování je vhodné při řešení použít funkci `speed(0)`.
Používané pojmy: Funkce `range()`, cyklus `for`, operátor `in`, modul `turtle`.
Název souboru s možným řešením: ornament.py

29) **Mohli se potkat?**
Doplňte funkci `mohliSePotkat`, tak aby podle dat narození a úmrtí zjistila, zda se mohly uvedené osoby setkat.
Metodická poznámka:
V připraveném souboru mohli_se_potkat_zadani.py jsou do proměnných dennar1, dennar2, densmrti1, densmrti2 vloženy data narození a smrti. Je možné zjišťovat, které je menší (<), větší (>), rovnost (==) apod. V docstringu řešeného příkladu jsou použita data narození a úmrtí Karla Čapka (9. 1. 1890 – 25. 12. 1938), Václava Havla (5. 10. 1936 – 18. 12. 2011), Boženy Němcové (4. 2. 1820 – 21. 2. 1862) a Ericha Maria Remarqua (22. 6. 1898 – 25. 8. 1970). [14]
Používané pojmy: `False`, `True`, podle způsobu řešení buď vnořené větvení `if`, nebo logické spojky.
Název souboru s možným řešením: mohli_se_potkat_reseni.py

30) **Telegram**

Albert Einstein se snažil zabránit justičním vraždám ve vykonstruovaném procesu se skupinou Milady Horákové, pomocí telegramu zaslaného 13. 6. 1950 tehdejšímu premiérovi Antonínu Zapotockému: „Prosím Vás o nevykonání rozsudku vyneseného nad Miladou Horákovou, Závišem Kalandrou, Oldřichem Peclem a Janem Buchalem. Byli oběťmi nacismu, vězni německých koncentračních táborů. Jsem hluboce přesvědčen, že si zaslouží žít.“ [60]

V originálním telegramu vypadal text následovně (text je přepsán včetně chyb):

I PLEAD WITH YOU NOT CARRY OUT THE DEATH SENENCE ON MILADA HORAKOVA ZAVIS KALANDRA OLDRICH PECL AND JAN BUCHAL STOP THEY WERE VICTIMS OF NAZISM INMATES OF GERMAN CONCENTRATION CAMPS STOP AM DEEPLY CONVINCED THAT THEY DESERVE TO LIVE = PROFESSOR ALBERT EINSTEIN PRINCETON UNIVERSITY + [61]

a) Vytvořte funkci, která převede text na vstupu do podoby telegramu. Tedy všechna písmena budou velká a místo teček na konci věty bude slovo STOP. Předpokládejte text zadaný bez diakritiky.
b) Einstein měl potíže se čtením (začal číst v devíti letech), psaní mu dělalo potíže i v dospělosti. [63] Naleznete chyby, kterých se v anglickém textu dopustil?

Metodická poznámka:

Fotografii originálního telegramu lze žákům promítnou z webové stránky http://zpravy.idnes.cz/foto.aspx?foto1=CHU337e9d_05KAV_Horakova_Einstein.jpg/. Žáci mohou řešit problém s použitím větvení `if` uvnitř cyklu `for`. Někteří si mohou myslet, že to nelze.

Používané pojmy: Metoda `upper()`, operátor `in`, vnořené větvení `if` v cyklu `for`, manipulace s řetězci.

Řešení části b: chybně SENENCE – správně SENTENCE (česky rozsudek)
chybně AM DEEPLY – správně I AM DEEPLY
chybně NOT CARRY OUT – správně NOT TO CARRY OUT

Název souboru s možným řešením: telegram.py

31) **Fénické písmo**

Naše písmo i většina ostatních (např. kromě čínštiny) je ovlivněna fénickým písmem. Fénická abeceda neobsahovala žádné samohlásky, což stěžovalo porozumění. [53] Vytvořte funkci, která z textu odstraní veškeré samohlásky.

Metodická poznámka:

Při zadávání ukažte žákům, možné výstupy spuštěním souboru fenicke_ukazka.py. Nechte je hádat, který fyzikální zákon a známý citát je v převedeném textu (řešení: Archimédův, Heuréka! – údajné zvolání Archiméda při objevu Archimédova zákona).

Používané pojmy: Vnořené větvení `if` v cyklu `for`, operátory `in`, `not in`.

Název souboru s možným řešením: fenicke_pismo.py

32) **Brzdná dráha**

Brzdná dráha je vzdálenost, na které se vozidlo jedoucí nějakou počáteční rychlostí úplně zastaví. Je závislá na rychlosti řidičovy reakce, kvalitě brzd a přilnavosti pneumatik k vozovce. [51] Vytvořte funkci, která brzdnou dráhu dokáže odhadnout.

Rozšíření:

funkce:

a) se zeptá na počáteční rychlost
b) vypíše hodnotu i s jednotkou zaokrouhlenou na celé číslo
c) ověří reálnou hodnotu rychlosti (zda není zadaná hodnota záporná, nebo vyšší než dovolují technické možnosti)
d) se zeptá na stav vozovky a přiřadí odpovídající hodnotu (u intervalů zvolte střední hodnotu)
e) ověří korektní odpověď pro stav vozovky
f) ověří korektní odpověď pro počáteční rychlost (zda je odpověď číslo)

Rozšíření c) a f) je možné aplikovat i na reakční dobu řidiče.

Nápověda k řešení:

Řidič má běžně reakční dobu 0,3-1s. [55] Žáci si ji mohou změřit v apletech na Internetu např. http://fyzweb.cz/materialy/aplety_hwang/Reaction/index.html/ nebo na http://www.bbc.co.uk/science/humanbody/sleep/sheep/reaction_version5.swf/.

Brzdná dráha vykazuje kvadratickou závislost na počáteční rychlosti:

$s = v_0(t_1 + t_2) + \frac{v_0^2}{2 \cdot f \cdot g}$, kde v_0 je počáteční rychlost vozu, t_1 reakční doba řidiče, t_2 reakční doba brzd, f koeficient smykového tření mezi pneumatikou a povrchem vozovky, g = 9,81 m/s^2 gravitaní zrychlení. Reakční doba brzd asi 0,15 s až 0,25 s. [54, 55]

Materiály	Statický koef. tření (*f*)
Pryž (pneumatika) na náledí	0,1-0,2
Pryž na betonu	0,7-0,8
Pryž na dlažbě (malé kostky)	0,6-0,7
Pryž na dlažbě (velké kostky)	0,6
Pryž na mokrém asfaltu	0,2-0,5
Pryž na suchém asfaltu	0,55

Tab. 4.2: Koeficienty smykového tření [56]

1 km/h = $\frac{1}{3{,}6}$ m/s

Metodická poznámka:

Na této úloze je možné pracovat v průběhu několika hodin, přidávat jednotlivá rozšíření podle probíraného učiva. Použití rozšíření c) a f) na reakční dobu řidiče je možné zadat i jako domácí úkol (zvláště pokud byla ověření na rychlost provedena v hodině). Bod f) je určen pro pokročilejší studenty, kteří jsou seznámeni

s výjimkami. Řazení doplňujících úkolů je podle obtížnosti. Žáky je vhodné upozornit, aby přemýšleli o vhodném umístění v programu a nepsali vždy pokračování nakonec. Část d) a e) již není v řešeném příkladu, možností je více a část e) je značně závislá na d).

Pozor na násobení veličin ve jmenovateli. Častou chybou je zápis $v_0^2/2*f*g$, což odpovídá $(v_0^2/2)*f*g$. Dále je třeba žáky navést k problému jednotek rychlosti, kdy běžně v této souvislosti užíváme km/h, ale do výše uvedeného vztahu je třeba dosazovat m/s. Za reakční dobu brzd je možné zvolit pevnou hodnotu, nejlépe průměr 0,2 s.

Používané pojmy: Matematické operace (včetně umocňování), v rozšíření větvení `if` (i s částí `elif`), cyklus `while`, funkce `input`, `int()`, `try - except`.

Název souboru s možným řešením: brzdna_draha.py (jen část a), brzdna_draha_rozsireni.py

33) **Pascalův trojúhelník**

Pascalův trojúhelník může ukázat kolika způsoby padne hlava (H) či orel (O) při hodech mincí. Pokud si například hodíte mincí třikrát je jedinná kombinace výsledku tří orlů (OOO), ale tři pro dva orly a jednu hlavu (OOH, OHO, HOO), opět tři pro jednoho orla a dvě hlavy (OHH, HOH, HHO) a jedna pro všechny tři hlavy (HHH). Čísla 1, 3, 3, 1 tvoří třetí řádek Pacalova trojúhelníku. [35]

1) Vytvořte funkci pro výpočet faktoriálu čísla. V případě zadání záporného čísla bude vypsán upozorňující text. Ověřte výsledky svého programu pomocí `factorial` modulu `math`.
2) Vytvořte funkci pro kombinační číslo. V případě, že jsou zadaná čísla záporná vypište upozornění.
3) Vytvořte funkci, která vypíše n-tý řádek Pascalova trojúhelníku, kde n bude argumentem.
4) Vytvořte funkci, která vytvoří Pacalův trojúhelník po n-tý řádek, kde číslo n zadá uživatel.

Nápověda k řešení:

Faktoriál značený $n!$ je součin všech přirozených čísel od 1 do n. Tedy $n! = 1 \cdot 2 \cdot ... \cdot n$ pro každé $n \in N$. Pro nulu platí 0! = 1. [39]

Kombinační číslo $\binom{n}{k}$ (čteme „n nad k“) je počet všech k-členných kombinací z n prvků. Pro každé $n, k \in N_0, k \le n$ platí $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. K-členná kombinací z n prvků ($n, k \in N_0, k \le n$) je každá množina (žádné prvky se neopakují) k prvků vybraných z n daných prvků. [39]

Kombinační čísla lze zapsat do schématu zvaného Pascalův trojúhelník, na jehož vrchulu je číslo 1. Každý řádek začíná a končí také číslem 1. Libovolné číslo uvnitř lze získat sečtením čísel ležících bezprostředně nad ním.

Metodická poznámka: Kombinatorika a pravděpodobnost, jejichž součástí jsou faktoriály a kombinační čísla, bývají většinou probírány ke konci 3. či začákem 4. ročníku. Zjistěte zda vaši žáci danou problematiku již ovládají. Pro lepší představu Pascalova trojúhelníku je možné žákům promítnout znázornění z webových stránek http://carolina.mff.cuni.cz/~jana/kombinatorika/04Pascaluv_trojuhelnik.htm/, kde se barevně zvýrazňují odpovídající si čísla ve schématu s kombinačními čísly a schématu vyčísleném. Žáci mohou mít problém se správným výstupem Pascalova trojúhelníku, tedy aby každý řádek byl zarovnán na střed. Diskutujte s žáky, jak toho docílit.

Používané pojmy: Větvení `if` (i s částí `elif`), funkce `print()`, `range()`, cyklus `for`, rekurze.

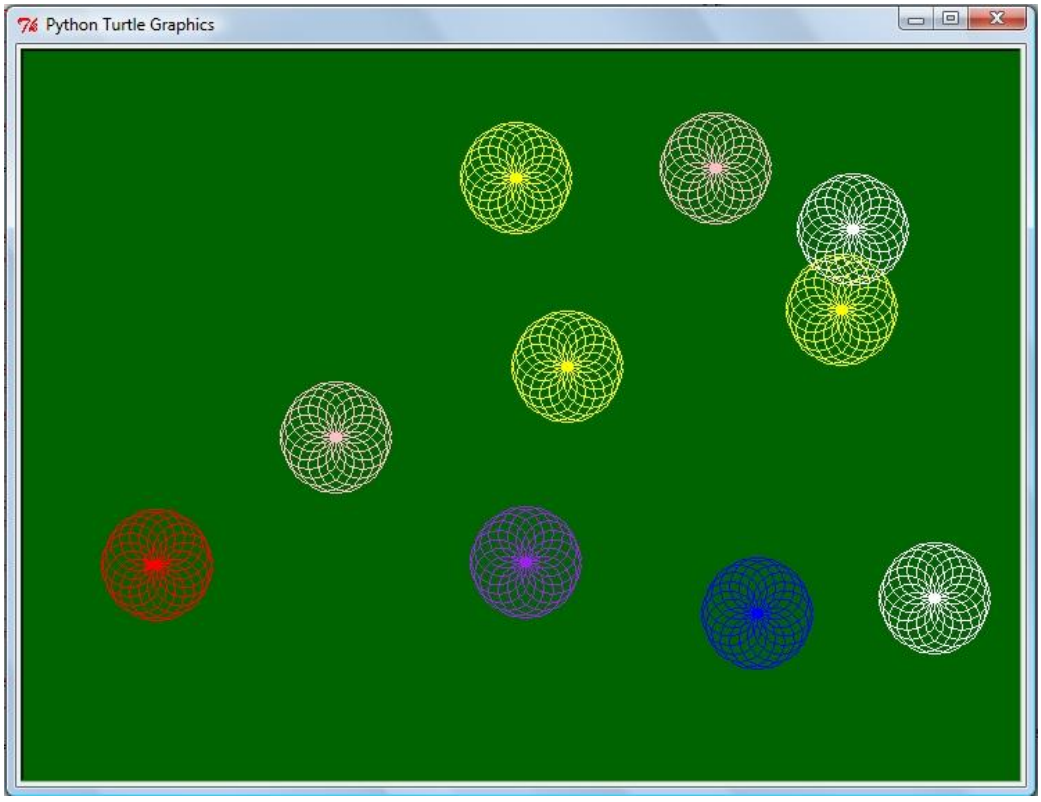
Název souboru s možným řešením: pascal.py

34) Při některých hrách se hráči mají seřadit podle různých informací, kterými mohou být např. výška, či věk. Vytvořte funkci pro seřazení prvků seznamu od nejmenší po největší hodnotu pomocí algoritmu bubble sort.

Metodická poznámka: Pro lepší názornost použitého algoritmu je možné žákům pustit vizualizaci na http://www.algoritmy.net/article/3/Bubble-sort/. Příklad je možné modifikovat použitím dalších řadících algoritmů.

Používané pojmy: Seznam, indexování prvků seznamu, funkce `len()`, `range()`, vnořený cyklus `for`, vnořené větvení `if`, řadící algoritmy (bubble sort).

Název souboru s možným řešením: bubble.py



Obr. 4.2: Ukázka výstupu funkce příkladu č. 35

35) Vytvořte funkci, která vykreslí 10 „květin na louce“ podle obrázku 4.2. Vhodná barva i poloha se budou volit náhodně (použijte modul `random`).

Metodická poznámka:

Pro rychlejší vykreslování je vhodné při řešení použít funkci `speed(0)`. Žáci mohou mít problém s tím, jak náhodně zvolit barvu. Pokuste se je navést na myšlenku pomocné funkce pro přiřazení barev k číslům. Při řešení je možné navázat na příklad č. 28.

Používané pojmy: Funkce `range()`, větvení `if` (i s částí `elif`), cyklus `for`, operátor `in`, vnořený cyklus `for`, modul `random`, modul `turtle`.

Název souboru s možným řešením: louka.py

5. Závěr

Tato diplomová práce se věnuje rozboru a zhodnocení jazyka Python z hlediska výuky na střední škole. Jedná se o poměrně nový jazyk, který je však rovnocennou volbou vedle běžně používaných (např. Pascal, C). Další podstatnou částí je použití jazyka Python v jednotlivých vyučovaných oblastech a uvedení jeho základních vlastností, které může učitelům ulehčit změnu využívaného jazyka. V poslední kapitole jsou řešené příklady, jejichž zadání byla volena z různých oblastí pro lepší motivaci žáků. Příklady jsou doplněny o používané pojmy a metodické pokyny pro učitele, kde je mnohdy upozorněno na možné chyby, kterých se žáci mohou dopouštět. Pokud je to nutné, je jejich součástí i nápověda k řešení, kde jsou uvedeny potřebné informace z jiných oblastí (např. fyzika).

Středoškolští učitelé informatiky, kteří uvažují o volbě jazyka Python pro výuku, mohou v této práci nalézt argumenty, které jim jejich rozhodování usnadní. V případě volby Pythonu pro výuku, jim usnadní přechod a také obohatí výuku o nové příklady. Zadání příkladů může být využito při výuce programování i v jiném programovacím jazyce.

Pokračováním této práce by mohlo být experimentální ověření příkladů ve výuce. Dále pak využití Pythonu při výuce objektově orientovaného programování (v návaznosti i na diplomovou práci Tomáše Foltýnka [1]) a tvorby webových stránek.

Literatura

1) FOLTÝNEK, Tomáš. *Výuka objektově orientovaného programování na středních školách*. Brno, 2003. Diplomová práce. Masarykova univerzita, Fakulta informatiky.
2) RÝDLO, Lukáš. *Programovací jazyky pro výuku programování na SŠ* [online]. Brno, 2012. Diplomová práce. Masarykova univerzita, Fakulta informatiky. [cit. 2013-09-26]. Dostupné z: http://is.muni.cz/th/139514/fi_m/dp.pdf
3) SVOBODA, Emanuel a kol. *Přehled středoškolské fyziky*. 2. přepracované vydání. Praha: Prometheus, 1996. ISBN 80-7196-006-3.
4) MIKULČÁK, Jiří, Bohdan KLIMEŠ, Jaromír ŠIROKÝ, Václav ŠŮLA, František ZEMÁNEK. *Matematické, fyzikální a chemické tabulky*. Dotisk 1. vydání. Praha: SPN, 1989.
5) ROSINA, Jozef, Hana KOLÁŘOVÁ, Jiří STANEK. *Biofyzika pro studenty zdravotnických oborů.* Vydání první. Praha: Grada, 2010. ISBN 978-80-247-1383-0.
6) *Katalog požadavků zkoušek společné části maturitní zkoušky: INFORMATIKA, základní úroveň obtížnosti* [online]. Praha: Centrum pro zjišťování výsledků vzdělávání, 2010. [cit. 2013-06-10]. Dostupné z: http://www.novamaturita.cz/index.php?id_document=1404034533&at=1
7) *Rámcový vzdělávací program pro gymnázia* [online]. Praha: Výzkumný ústav pedagogický v Praze, 2007. [cit. 2013-05-13]. ISBN 978-80-87000-11-3. Dostupné z: http://www.vuppraha.cz/wp-content/uploads/2009/12/RVPG-2007-07_final.pdf
8) *Python* [online]. The Python Software Foundation, 2001 – 2013. [cit. 2013-10-30]. Dostupné z: http://www.python.org
9) ROWLETT, Russ. *How Many? A Dictionary of Units of Measurement* [online]. Chapel Hill: University of North Carolina, 2008.[cit. 2013-11-14]. Dostupné z: http://www.unc.edu/~rowlett/units/dictL.html
10) *Internetová jazyková příručka* [online]. Praha: Jazyková poradna ÚJČ AV ČR, v. v. i., 2008-2013 [cit. 2013-11-14]. Dostupné z: http://prirucka.ujc.cas.cz/?id=795
11) OLBRICH, Libor, Alena MAŠLÁŇOVÁ. Současný katalog maturity z Informatiky očima učitele. *Cekaskola.cz* [online]. 2011. [cit. 2013-06-10]. Dostupné z: http://www.ceskaskola.cz/2011/11/libor-olbrich-alena-maslanova-soucasny.html
12) PSPad. *Py.cz* [online]. PyCZ, last edited 2011/10/10. [cit. 2013-10-30]. Dostupné z: http://www.py.cz/PSPad
13) GAVENČIAK, Tomáš. *Seriál o Pythonu* [online]. Praha: M&M, OVVP UK MFF, Poslední úprava: 19. 5. 2011. [cit. 2013-10-30]. Dostupné z: http://mam.mff.cuni.cz/?s=python
14) *Osobnosti.cz* [online]. [Praha]: Tiscali Media, 1996 – 2013. [cit. 2013-11-14]. Dostupné z: http://www.spisovatele.cz/
15) *Django* [online]. Django Software Foundation, 2006 – 2013. [cit. 2013-10-31]. Dostupné z: http://www.djangoproject.cz/
16) *Kurzycz* [online]. Praha: Kurzy.cz, AliaWeb, 2000 – 2013. [cit. 2013-06-20]. Dostupné z: http://www.kurzy.cz/komodity/index.asp?A=5&idk=87&curr=CZK&on=0&unit=1 kg&lg=1
17) CAMPBELL Jennifer, Paul GRIES. Learn to Program: The Fundamentals. *Coursera.org* [online]. 2012 [cit. 2013-06-17]. Dostupné z: https://class.coursera.org/programming1-002/lecture

18) *Rámcový vzdělávací program pro obor vzdělání 18-20-M/01 Informační technologie* [online]. Praha: Ministerstvo školství, mládeže a tělovýchovy, 2008. [cit. 2013-06-10]. Dostupné z: http://zpd.nuov.cz/RVP/ML/RVP%201820M01%20Informacni%20technologie.pdf
19) Krevní tlak. *Szu.cz* [online]. Praha: Státní zdravotní ústav. [cit. 2013-06-10]. Dostupné z: http://www.szu.cz/ehes-krevni-tlak
20) KOTEK, Lukáš. *Výzkum používaných programovacích jazyků na středních školách* [online]. Plzeň: 2013. [cit. 2013-09-24]. Dostupné z: http://www.olympiadatechniky.zcu.cz/@2013/Sbornik_OT2013_online/sekce/kotek_vyzkum_clanek.pdf
21) VYSTAVĚL, Radek. Moderní výuka programování. *Rvp.cz* [online]. [Praha]: RVP, 2008. [cit. 2013-09-24]. Dostupné z: http://clanky.rvp.cz/clanek/o/g/2646/MODERNI-VYUKA-PROGRAMOVANI.html/
22) *Školní vzdělávací program pro gymnaziální vzdělávání* [online]. [Brno]: Biskupské gymnázium Brno, 2013. [cit. 2013-09-26]. Dostupné z: http://www.bigy.cz/sites/www.bigy.cz/files/attachments/76/547/svp4letever20133.pdf
23) *Školní vzdělávací program Gymnázia, Brno, Elgartova 3 pro obor vzdělávání 79-41-K/41 - Gymnázium* [online]. [Brno]: Gymnázium, Brno, Elgartova 3, 2009. [cit. 2013-09-26]. Dostupné z: http://www.gymelg.cz/sites/default/files/GE_SVP.pdf
24) Charakteristika předmětu Seminář z informatiky. *Gyrec.cz* [online]. [Brno], Gymnázium Brno-Řečkovice, 2010 – 2012. [cit. 2013-09-26]. Dostupné z: http://www.gyrec.cz/content/charakteristika-predmetu-inf
25) TIOBE Programming Community Index for September 2013. *Tiobe.com* [online]. Eidenhoven: TIOBE Software, 2013. [cit. 2013-09-27]. Dostupné z: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
26) HAVRÁNEK, Michal. *Programovací jazyky a jejich využití* [online]. Zlín, 2011. Diplomová práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky. [cit. 2013-09-28]. Dostupné z: http://dspace.k.utb.cz/bitstream/handle/10563/18355/havr%C3%A1nek_2011_dp.pdf?sequence=1
27) LONDON, Jack. *Moře, sníh a velkoměsta*. Praha: Albatros, 1985.
28) *Prace.cz* [online]. [Praha]: LMC s.r.o, 1996-2013. [cit. 2013-09-28]. Dostupné z: http://www.prace.cz
29) Python Job Board. *Python.org* [online]. Python Software Foundation, 1990 – 2013. [cit. 2013-09-28]. Dostupné z: http://www.python.org/community/jobs/
30) *Codeacademy Labs* [online]. [New York]: Codeacademy, 2013. [cit. 2013-10-31]. Dostupné z: http://labs.codecademy.com/
31) HAZEL, Steven. *Codepad* [online]. [cit. 2013-10-31]. Dostupné z: http://codepad.org/
32) MOHTASHIM, Mohammad. *Online compilator and Execution* [online]. Aligarh. [cit. 2013-10-31]. Dostupné z: http://www.compileonline.com/
33) *Idone.com* [online]. Gdynia: Sphere Research Labs. [cit. 2013-10-31]. Dostupné z: http://ideone.com/
34) ZELLE, John M. Přeložil Pavel KOSINA, Vlaďa MACEK. *První jazyk: Python* [online]. 2004. [cit. 2013-11-2]. Dostupné z: http://macek.sandbox.cz/texty/prvni-jazyk-python/
35) Pascal's Triangle. *MathsIsFun.com* [online]. 2014. [cit. 2014-03-02]. Dostupné z: http://www.mathsisfun.com/pascals-triangle.html
36) HARMS, Daryl, Kenneth MCDONALD. *Začínáme programovat v jazyce Python.* 2. opravené vydání. Brno: Computer Press, a.s., 2008. ISBN 978-80-251-2161-0

37) LUTZ, Mark, David ASCHER. *Naučte se Python*. 1. vydání. Praha: Grada Publishing a.s., 2003. ISBN 80-247-0367-X
38) PILGRIM, Mark. *Ponořme se do Python(u) 3.* Praha: CZ.NIC, z. s. p. o., 2010. ISBN 978-80-904248-2-1
39) POLÁK, Josef. *Přehled středoškolské matematiky.* Dotisk 6. vydání. Praha: Prometheus, 1991. ISBN 80-85849-78-X
40) SUMMERFIELD, Mark. *Python 3: výukový kurz*. Vyd. 1. Brno: Computer Press, 2010. ISBN 978-80-251-2737-7.
41) STANNARD, Kevan. How to define constants in Python? *Pythonexamples.org* [online]. Sydney: 2011. [cit. 2014-04-08]. Dostupné z: http://www.pythonexamples.org/2011/01/12/how-to-define-constants-in-python/
42) ELKNER, Jeffrey, Allen B. DOWNEY, Chris MEYERS. Přeložil Jaroslav KUBIAS. *Učíme se programovat v jazyce Python 3* [online]. 2008, 2012, 2013. [cit. 2014-03-15]. Dostupné z: http://howto.py.cz/
43) *Python 3.3.5. documentation* [online]. Python Software Foundation, 1990-2014, Last updated on Mar 18, 2014. [cit. 2014-04-15]. Dostupné z: https://docs.python.org/3.3/
44) Floating-point number. *Webopedia.com* [online]. QuinStreet, 2014. [cit. 2014-04-10]. http://www.webopedia.com/TERM/F/floating_point_number.html
45) BÍLEK, Petr. *Sallyx.org* [online]. Kladno: 2005. [cit. 2014-03-15]. Dostupné z: http://www.sallyx.org/sally/python/
46) MIČKA, Pavel. *Algoritmy.net* [online]. Algoritmy.net, 2008-2014. [cit. 2014-04-26]. Dostupné z: http://www.algoritmy.net/
47) *Jazykolamy* [online]. LowArea@gmail.com, 2010. [cit. 2014-03-22]. Dostupné z: http://jazykolamy.funsite.cz/
48) Jednotlivé země. *Europa.eu* [online]. Evropská unie, 1995-2014. [cit. 2014-03-23]. Dostupné z: http://europa.eu/about-eu/countries/index_cs.htm
49) Space Segment. *GPS.gov* [online]. National Coordination Office for Space-Based Positioning, Navigation, and Timing, 2013. [cit. 2013-10-23]. Dostupné z: http://www.gps.gov/systems/gps/space/
50) *Český kosmologický portál* [online]. Odbor kosmických technologií a družicových systémů, 2013. [cit. 2013-10-23]. Dostupné z: http://www.czechspaceportal.cz/
51) Brzdná dráha. *Autolexicon.net* [online]. Mladá Boleslav: 2013. [cit. 2013-11-26]. Dostupné z: http://cs.autolexicon.net/articles/brzdna-draha/
52) ROSSUM, Guid van, Fred L. DRAK. Přeložil Jan ŠVEC. *Učebnice jazyka Python (aneb Létající cirkus)* [online]. PyCZ, last edited 2008/09/08. [cit. 2014-03-15]. Dostupné z: http://www.py.cz/UcebniceJazykaPython
53) SOCHA, Vladimír. Pramáti téměř veškerého písma. *Historieweb.cz* [online]. Extra Publishing, 2012. [cit. 2013-09-01]. Dostupné z: http://www.historieweb.cz/pramati-temer-veskereho-pisma
54) FAUS, Pavel. *Autoškola 2013 – Moderní učebnice*. Praha: Grada, 2013. ISBN: 978-80-247-4703-3
55) Kolektiv autorů. Bezpečnost vozidla. *Život s autem* [online]. [Brno]: Masarykova univerzita, Pedagogická fakulta, Katedra geografie, 2010. [cit. 2013-11-26]. Dostupné z: http://is.muni.cz/do/ped/kat/fyzika/autem/pages/bezpecnost.html
56) JANÍK, Josef. *Fyzika tření* [online]. Brno, 2008. Bakalářská práce. Masarykova univerzita, Pedgogická fakulta. [cit. 2013-11-26]. Dostupné z: http://is.muni.cz/th/199446/pedf_b/Fyzika_treni.doc

57) Perština. *Wikipedia: the free encyclopedia* [online]. 2001 – 2013. [cit. 2013-09-03]. Dostupné z: http://cs.wikipedia.org/wiki/Per%C5%A1tina
58) Urdština. *Wikipedia: the free encyclopedia* [online]. 2001 – 2013. [cit. 2013-09-03]. Dostupné z: http://cs.wikipedia.org/wiki/Urd%C5%A1tina
59) COOPER, Adam W. Practice Makes Perfect. *Codecademy.com* [online]. Codecademy, 2013. [cit. 2013-09-03]. Dostupné z: http://www.codecademy.com/courses/python-intermediate-en-rCQKw/1/1?curriculum_id=4f89dab3d788890003000096
60) Albert Einstein. *Wikipedia: the free encyclopedia* [online]. 2001 – 2013. [cit. 2013-09-03]. Dostupné z: http://cs.wikipedia.org/wiki/Albert_Einstein
61) KOUROVI, Petr a Pavlína. Před šedesáti roky probíhalo v této zemi běsnění proti Miladě Horákové a spol. *Idnes.cz* [online]. Praha: MAFRA, 5. června 2010. [cit. 2013-06-26]. Dostupné z: http://zpravy.idnes.cz/kavarna.aspx?c=A100601_140453_kavarna_chu
62) VERNE, Jules. *Dvacet tisíc mil pod mořem*. Praha: Albatros, 1983.
63) SELIKOWITZ, M.: Dyslexie a jiné poruchy učení. Praha: Grada, 2000. ISBN 80-7169-773-7
64) POLÁČEK, Karel. *Edudant a Francimor*. Praha: Albatros, 1970.
65) BROKLOVÁ, Zdeňka. *Učíme jadernou fyziku.* Praha: ČEZ. ISBN 978-80-254-1342-5

Obsah elektronické přílohy

Název souboru	popis
bourka.py	možné řešení příkladu č. 13
brzdna_draha.py	možné řešení příkladu č. 32 (část a)
brzdna_draha_rozsireni.py	možné řešení příkladu č. 32
bubble.py	možné řešení příkladu č. 34
den_v_tydnu_reseni.py	možné řešení příkladu č. 21
den_v_tydnu_zadani.py	možné zadání příkladu č. 21
dny.py	možné řešení příkladu č. 20
druzice.py	možné řešení příkladu č. 22
fenicke_pismo.py	možné řešení příkladu č. 31
fenicke_ukazka.py	ukázka výstupu příkladu č. 31
hmotnost_atomu.py	možné řešení příkladu č. 14
krevni_tlak.py	možné řešení příkladu č. 27
kychani.py	možné řešení příkladu č. 25
louka.py	možné řešení příkladu č. 35
mile.py	možné řešení příkladu č. 18
mohli_se_potkat_reseni.py	možné řešení příkladu č. 29
mohli_se_potkat_zadani.py	možné zadání příkladu č. 29
ornament.py	možné řešení příkladu č. 28
pascal.py	možné řešení příkladu č. 33
pozpatku.py	možné řešení příkladu č. 24
rychlost.py	možné řešení příkladu č. 17
sekundy.py	možné řešení příkladu č. 11
stari.py	možné řešení příkladu č. 23
stoleti.py	možné řešení příkladu č. 19
tabule.py	možné řešení příkladu č. 26
telegram.py	možné řešení příkladu č. 30
tlak.py	možné řešení příkladu č. 15
torr_na_pascal.py	možné řešení příkladu č. 10
vzdalenost.py	možné řešení příkladu č. 16
zlato.py	možné řešení příkladu č. 12