

Univerzita Karlova v Praze

Pedagogická fakulta

Katedra informačních technologií a technické výchovy

Python – programovací jazyk pro výuku algoritmizace a programování

Autor: Lukáš Kotek

Vedoucí práce: Ing. Jaroslav Novák, Ph.D.

Praha 2011

PODĚKOVÁNÍ:

Rád bych poděkoval za pomoc s realizací této práce svému vedoucímu Ing. Jaroslavu Novákovi, Ph.D., dále pak Ing. Miroslavu Škopovi a Ing. Jiřímu Čepelákovi, CSc.

PROHLÁŠENÍ:

Prohlašuji, že jsem tuto práci vypracoval samostatně. Dále, že veškeré prameny použité v této práci byly řádně citovány a jsou uvedeny v seznamu použitých informačních zdrojů. Rovněž prohlašuji, že tato práce nebyla žádným způsobem využita k získání jiného nebo stejného titulu.

V Praze dne 30. března 2011

.....

NÁZEV:

Python – programovací jazyk pro výuku algoritmizace a programování

ABSTRAKT:

Práce představuje programovací jazyk Python jako nástroj pro výuku algoritmizace a programování. Pomocí průzkumu zkoumá vhodnost Pythonu pro tento účel. I na jeho základě konstatuje, že ho pro něj je možné použít. Popisuje jeho historii, charakteristiky a specifika vůči jiným jazykům. Vypočítává nástroje, editory a vývojová prostředí, které je s ním možné použít ve výuce, a konstatuje, že jich existuje dostatek. Obsahuje ukázky a příklady, které demonstrují jeho možná použití.

KLÍČOVÁ SLOVA:

python, algoritmizace, programování, nástroje, editory, příklady

Obsah

1 Úvod.....	8
2 Východiska práce.....	9
3 Software ve školách.....	11
3.1 Používané jazyky.....	11
3.1.1 Pascal a Object Pascal.....	13
3.1.2 C a C++.....	14
3.1.3 Logo a Karel.....	14
3.2 Průzkum současné situace.....	15
3.2.1 Pracovní cíle průzkumu.....	15
3.2.2 Metodika.....	16
3.2.3 Získaná data.....	16
3.2.4 Interpretace.....	16
4 Python.....	18
4.1 Základní rysy Pythonu.....	18
4.1.1 Interpretovaný jazyk.....	18
4.1.2 Základy syntaxe.....	19
4.1.3 Základ datových typů.....	20
4.1.4 Základní operátory.....	21
4.1.5 Seznamy, n-tice a pole.....	23
4.1.6 Podmínky.....	26
4.1.7 Cykly.....	29
4.1.8 Procedury a funkce.....	30
4.1.9 Objektový model.....	33
4.1.10 Moduly.....	35
4.2 Historie jazyka.....	37
4.2.1 Vznik a design Pythonu.....	37
4.3 Implementace.....	39
4.3.1 Python (CPython).....	39
4.3.2 IronPython.....	40
4.3.3 Jython.....	41
4.4 Základy použití.....	41
4.4.1 Instalace Pythonu.....	41
4.4.2 Interpreter příkazů.....	42
4.4.3 Konsolové programy.....	43
4.4.4 Programy s GUI.....	44

4.4.5	Webové aplikace.....	45
4.5	Zhodnocení Pythonu.....	45
4.5.1	Pracovní cíle průzkumu.....	45
4.5.2	Metodika.....	45
4.5.3	Souhrn získaných dat.....	46
4.5.4	Interpretace.....	48
5	Vývojová prostředí a platformy pro vývoj v Pythonu.....	48
5.1	Vývojová prostředí a editory.....	48
5.1.1	IDLE.....	49
5.1.2	Eclipse (Pydev plugin).....	50
5.1.3	The Eric Python IDE.....	51
5.1.4	VIM.....	52
5.1.5	PSPad.....	52
5.1.6	Ostatní (Visual Studio, Netbeans).....	52
5.2	Nástroje a moduly.....	53
5.2.1	Pro matematické operace.....	53
5.2.2	Pro tvorbu grafického rozhraní.....	54
5.2.3	Práce s databázemi.....	55
5.2.4	Webové nástroje.....	56
5.2.5	Ostatní.....	56
5.3	Python v praxi.....	57
6	Příklady.....	57
6.1	Jednoduchý FTP klient.....	57
6.2	Kreslení grafu střídavého signálu.....	58
6.3	Webová aplikace s HTML formulářem.....	59
7	Závěr.....	62
8	Seznam ukázek.....	64
9	Seznam obrázků.....	66
10	Seznam použitých informačních zdrojů.....	67
11	Seznam příloh.....	73

1 Úvod

V této práci představuji programovací jazyk Python jako nástroj pro účely výuky algoritmizace a programování na středních školách. Přibližuji jeho specifika v porovnání s některými dalšími jazyky běžně používanými pro tento účel a dále pak seznamuji s možnostmi a oblastmi, pro které může být Python použit a kde může být nasazen. Rovněž chci poukázat na softwarové prostředky (vývojová prostředí, balíky a moduly atd.), které mohou být pro práci s Pythonem použity. V neposlední řadě také ukazuji příklady z praxe, ve které se již nyní Python využívá.

Toto prezentuji jak ryze teoretickou formou, tak pomocí názorných ukázek kódu realizovaných programů, a to v průběhu celé práce. Další příklady jsou rovněž obsaženy v příloze na CD.

Bakalářská práce naopak není příručkou popisující vyčerpávajícím způsobem veškeré rysy a možnosti jazyka. Jazyk a jeho syntaxe jsou popsány v míře nezbytné pro pochopení uvedených ukázek a příkladů programů, a pro ozřejmění specifik tohoto programovacího jazyka.

V práci zmiňuji několik odlišných implementací jazyka Python. Není-li uvedeno jinak, jedná se o Python ve verzi 3.1 v jeho 64bitové podobě. Ta je použita, protože se jedná o stabilní větev originální implementace jazyka. Ačkoliv pro tuto verzi (v době odpovídající druhému pololetí roku 2010) stále ještě neexistovaly veškeré knihovny a nástroje jako pro verze jazyka řady 2.x, situace se rychle mění a zejména s ohledem na budoucí vývoj a fakt, že verze řady 3.x existují již od roku 2008 a odráží proto aktuální stav vývoje Pythonu, byla zvolena právě implementace ve verzi 3.1.

2 Východiska práce

Do značné míry určujícím faktorem pro tematické cílení jednotlivých kapitol a pro obsah příkladů (publikovaných zejména v závěrečné části práce) jsou současné Rámcové vzdělávací programy (dále jen RVP) pro střední školy. Prostor věnovaný algoritmizaci a programování se napříč několika zvolenými (a dále popsány) studijními obory výrazně liší. Obory (1) byly zvoleny v očekávání různé úrovně požadavků na výuku algoritmizace a programování.

Vezmeme-li RVP studijního oboru, u kterého se dá očekávat vskutku výrazné zastoupení tematického celku algoritmizace a programování – oboru Informační technologie (18-20-M/01), pak v části Programování a vývoj aplikací nalezneme následující informace:

„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, řídicí 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.“ (2)

Dále jsou podrobněji rozebírány konkrétní požadavky, kterým se má ve výuce dostat. Výše uvedený text, resp. kritéria v něm obsažená, lze hodnotit jako dostačující, nicméně je vhodné mít na zřeteli, že studijní obor, o kterém je řeč, se oblasti z logiky věci věnuje do hloubky už kvůli své podstatě.

Posuneme-li se v naší snaze o ilustraci dále, narazíme na další zvolený obor Elektrotechnika (26-42-M/01). V části Vzdělávání v informačních a komunikačních technologiích nacházíme tento text:

„Žák (...) ovládá principy algoritmizace úloh a sestavuje algoritmy řešení konkrétních úloh (dekompozice úlohy na jednotlivé elementárnější činnosti za použití přiměřené míry abstrakce);“ (3)

Ačkoliv opticky by se oba studijní obory nemusely nutně zdát jako diametrálně odlišné, důraz kladený na sledovanou oblast je výrazně nižší. Postoupíme-li ještě dále, k RVP pro gymnázia (RVP G), dočteme se (v části Informatika a informační a komunikační technologie) tyto informace:

„Vzdělávání v dané vzdělávací oblasti směřuje k utváření a rozvíjení klíčových kompetencí tím, že vede žáka k (...) uplatňování algoritmického způsobu myšlení při řešení problémových úloh;“ (4)

Nutno podotknout, že toto není jediná oblast, kde je téma algoritmizace uvedeno (dále např. v části týkající se matematiky). I tak se jedná o ještě obecnější vymezení než v předchozím případě.

Na těchto třech ukázkách, které zdaleka nejsou všezahrnující, jsem chtěl demonstrovat, jak dalece se mohou lišit požadavky na sledovanou oblast, i když se ve všech případech jedná o maturitní studijní obory. Tato práce se snaží svým charakterem držet obsahového okruhu stanoveného pro studijní obor Informační technologie (18-20-M/01), i když to není naprostým pravidlem a v odůvodněných případech je rozvedena problematika, jež je nad rámec stanoveného vymezení.

3 Software ve školách

3.1 Používané jazyky

V této části hlavně nastiňuji, jaké programovací jazyky se pro výuku algoritmizace a programování v současnosti na středních školách používají. Prostředkem pro zjištění takovýchto informací je obsahová analýza textu z dále citovaných zdrojů. V části zabývající se orientačním průzkumem současné situace se pak snažím takto získané údaje podložit. Důvodem pro tuto snahu je fakt, že v dalších částech práce jsou některé jednotlivé rysy a vlastnosti Pythonu porovnány právě s těmito jazyky.

Co se zdrojů týká, zaměřil jsem se na weby, materiály a publikace mj. těchto autorů: Jana Wagnera (5), ICT konzultanta, pedagoga a novináře, který je zároveň šéfredaktorem serveru Česká škola, a který rovněž působí v Jednotě školských informatiků, dále pak Rudolfa Pecinovského (6), který se problematikou výuky programování dlouhodobě zabývá a publikoval na toto téma mnoho prací, z nichž mnohé zveřejňuje na svých internetových stránkách. Kromě toho jsem se zaměřil i na weby, které se problematice programování (a jeho výuce) věnují.

Jan Wagner na svém blogu publikoval příspěvek, ve kterém cituje několik příspěvků z e-mailové konference Jednoty školských informatiků. Jako jazyky používané pro výuku jsou zde označeny např. Pascal a C++. Sám autor stránek hned na začátku příspěvku uvádí:

„Tradičním programovacím jazykem pro výuku programování na školách všech stupňů je bezesporu Pascal.“ (7)

V jiné části téhož textu jsou zmíněny mj. i programovací jazyky Karel a Logo:

„(..) zda učit didakticky vhodný, ale v praxi ne příliš použitelný jazyk (Karel, Logo, Baltík, ...) nebo jazyk méně didaktický, zato ihned umožňující řešit problémy 'reálného světa'.“ (7)

Odbočíme-li, pak už zde je zajímavou informací, že jeden z přispívajících považuje Python za vhodný k výuce programování. Jan Wagner v závěrečném shrnutí příspěvku jako jeden ze závěrů diskuse proběhlé na zmíněné e-mailové konferenci přesně uvádí

toto:

„S poměrně dobrým ohlasem se setkal návrh vzít jako první jazyk Python.“ (7)

Rudolf Pecinovský se k otázce výuky programování vyjadřuje opakovaně v knihách, jichž je autorem či spoluautorem. Výmluvným faktem je, že v těchto knihách opakovaně uvádí příklady v jazycích C, C++ a Pascal. Můžeme zmínit Základy algoritmizace, Práce s daty 1 nebo Objektové programování. V popisu knihy Základy algoritmizace na svém webu pak přímo píše:

„Učebnice je plná příkladů a ukázkových programů, které jsou uvedeny paralelně v jazycích C++ a Turbo Pascal.“ (8)

Ze základů algoritmizace pak pochází tato citace:

„Paralelně s jazykem C++ budeme vykládat i jazyk Pascal, protože i tento jazyk je dnes často používán k vývoji profesionálních aplikací. Navíc se tento jazyk často používá k výuce informatiky na středních školách. Proto si myslíme, že pro skutečného profesionála je nezbytná alespoň povrchní znalost obou těchto jazyků.“ (9)

O relevantní data se jedná už kvůli tematickému zaměření těchto knih – jde o učebnice programování, z nichž minimálně Základy algoritmizace se přímo kryjí s okruhy probíranými na středních školách. Kromě zmíněných jazyků je v této knize věnován prostor i programovacímu jazyku Karel.

Protože autor sám tyto publikace na svém webu řadí mezi starší, můžeme považovat za zajímavé, že se v současnosti autor věnuje programovacímu jazyku Java (zvláště v kombinaci s prostředím BlueJ) a jeho aplikaci pro výuku objektového programování na základních a středních školách (10), což náš výčet dále obohacuje.

Uděláme-li druhou odbočku, pak zjistíme, že Pecinovský zmiňuje v souvislosti s výukou OOP i Python. V seznamu kandidátů na jazyk, který by mohl být použit pro výuku OOP, píše:

„Jako horcí kandidáti, kteří se u mne probjovali do finále, se ukázaly (abecedně): C#, Delphi pro .NET, Java, Python (možná), Smalltalk a Visual Basic .NET.“ (11)

Z dalších autorů, kteří se ve svých pracích dotýkají tématu výuky programování, může zmínit Pavla Tišnovského, který na serveru Root.cz publikoval např. toto:

„Programovací jazyk Logo je vyvíjen již téměř čtyřicet let a ve světě se těší stále velké oblibě. Zejména na školách je používán pro výuku algoritmizace a především pro podporu konstruktivního učení. Někteří lidé považují Logo za pouhou dětskou hračku, ukážeme si však, že se jedná o zajímavý jazyk s možnostmi, které najdeme pouze ve vysokoúrovňových dynamických jazycích.“ (12)

V jiném článku od stejného autora pak nalezneme i následující text:

„Dalším tzv. dětským programovacím jazykem, který se v některých ohledech podobá výše zmíněnému Logu, je programovací jazyk Karel pojmenovaný po Karlu Čapkovi na paměť jeho hry R.U.R. Tento jazyk vytvořil v roce 1981 Richard E. Pattis a vzápětí po svém uvedení se tento jazyk rozšířil na všechny v té době používané osmibitové domácí mikropočítače.“ (13)

Mohu přidat i svou osobní zkušenost s praxí výuky programování na střední škole (konkrétně se jednalo o studijní obor 26-41-M/01 – Elektrotechnika), kde bylo po dva roky vyučováno programování v jazyce C.

Na základě těchto zjištění jsou v další části rozepsány základní charakteristiky jazyků C, C++, Pascal a Object Pascal. Pascal je zmíněn hlavně díky stále vysoké míře jeho rozšíření, C a C++ pak i proto, že z jejich syntaxe více či méně vychází mnoho dalších jazyků. Popsány jsou i jazyky Logo a Karel.

3.1.1 Pascal a Object Pascal

Pascal je jazyk, který byl původně vytvořen právě pro účely výuky programování. Má svůj původ v roce 1971, kdy spatřila světlo světa jeho první verze.

Je kompilovaný, staticky typovaný. Pascal jako takový je procedurálním programovacím jazykem, objektové rozšíření přišlo v roce 1985. Nejznámějšími implementacemi těchto jazyků jsou kompilátory zahrnuté do vývojových prostředí firmy Borland (Turbo Pascal, Delphi).

Zajímavou alternativou jsou kompilátory a prostředí vyvíjené Open source

projektem Free Pascal (14), dostupné jak pro Pascal, tak Object Pascal. Poskytovaná vývojová prostředí jsou svým charakterem podobná produktům firmy Borland (rozhraní prostředí Free Pascal IDE je podobné prostředí Turbo Pascal, Lazarus pak ctí stejné principy práce jako Delphi).

3.1.2 C a C++

Jazyk C (15) pochází rovněž z počátku 70. let. I tento jazyk respektuje procedurální model programování, je kompilovaný a staticky typovaný. C++ je jazyk, který z C vychází a implementuje model umožňující objektové programování. Vznikl v roce 1983.

Z nejrozšířenějších vývojových nástrojů můžeme jmenovat Open source sadu kompilátorů GCC (GNU Compiler Collection), mezi nimiž jsou zahrnuty kompilátory jak pro C, tak C++. GCC lze kombinovat s širokým množstvím vývojových prostředí, za všechny jmenujme např. Eclipse. Z dalších pak zmiňme např. prostředí Visual Studio od firmy Microsoft, které rovněž obsahuje kompilátor pro tyto jazyky.

Ze syntaxe C nebo C++ silně vychází např. jazyky Java, C#, Javascript nebo PHP. Další jazyky jsou pak touto syntaxí ovlivněny.

3.1.3 Logo a Karel

Ohledně programovacího jazyka Logo jsou např. na stránce webu Linux EXPRESS tyto informace:

„Funkcionální programovací jazyk Logo je dialektem nám již z minulých dělů známého jazyka Lisp. Byl navržen a primárně se používá pro výuku programování. Mezi jeho hlavními rysy patří interaktivita, modularita, rozšiřitelnost a flexibilita datových typů. (...) V dnešních dnech se Logo v oblasti výuky programování těší stále velké popularitě. Implementací Loga existuje více než 130. Populární implementace Loga jsou multiplatformní UCBLogo, slovenské Comenius Logo, jeho následník Imagine a dále pak MicroWorlds, LEGO Logo nebo StarLogo.“ (16)

Ze zmíněných implementací se v seriálu na serveru Root.cz (zabývajícím se programováním v Logu) o Comenius Logo dočteme:

„Názvem Comenius Logo je označena objektově orientovaná implementace Loga vyvíjená na Slovensku ale rozšířená takřka po celém světě. Autoři Comenius Loga, mezi které patří Andrej Blaho, prof. Ivan Kalaš a Petr Tomcsányi, vytvořili programovací nástroj s grafickým uživatelským rozhraním pracujícím v prostředí Microsoft Windows.“ (17)

Co se týká jeho nástupce – Imagine, na stránkách, které se mu výhradně věnují je popsán takovýmto způsobem:

„Je to kompletne objektový jazyk, ktorý je riadený udalosťami. Podporuje paralelné programovanie a tiež má prepracovanú ideu obrázkových tvarov korytnačiek. Má niektoré nové prvky, ktoré sú typické pre programy pod Windows, napr. prekrývajúce sa grafické plochy (ako listy papiera), tlačidlá aj s obrázkami, posuvné lišty, texty, lišty tlačidiel a pod. Nechýbajú ani multimédia, internet a tiež vzájomná spolupráca Imagine-programov v sieti.“ (18)

O programovacím jazyku Karel pak můžeme nalézt např. toto:

„Robot Karel je již velmi starý nástroj na výuku algoritmizace, který dnes není příliš rozšířen, i když jeho různé implementace lze najít i na Internetu v online provozu a jsou známy i české vysoké školy, jejichž studenti trénují algoritmizaci v Karlovi. Je to kvalitní nástroj na výuku algoritmizace, který záměrně postrádá práci s daty.“ (19)

3.2 Průzkum současné situace

3.2.1 Pracovní cíle průzkumu

Na základě předchozích konstatování i osobní zkušenosti stanovuji pro účely následujícího průzkumu hypotézu, že v českém středním školství se pro výuku algoritmizace a programování používají jazyky C, C++ a Pascal.

Cílem průzkumu je tuto hypotézu potvrdit či vyvrátit a spolu s tím zjistit i další informace, které mohou více osvětlit stav v oblasti algoritmizace a programování v současném českém školství.

Konkrétně se jedná o informace stanovené v těchto bodech:

- jaké nástroje se pro výuku se v této oblasti používají
- jaké tematické oblasti se ve výuce probírají
- jak hodnotí používané jazyky, nástroje a jejich účelnost studenti

3.2.2 Metodika

Pro zjištění informací vyjádřených v předchozích bodech jsem se rozhodl použít metody dotazníku. Cílem je získat větší množství dat, proto by tato metoda měla být dostatečně validní. S jejím použitím by mělo jít dosáhnout zároveň i dostatečné reliability získaných dat.

Průzkum byl realizován formou internetového dotazníku, pro který – pro možnost číselné reprezentace získaných dat a vůbec jejich snadného zpracování – by měly být vhodnou metodou škály (ať již bipolární nebo Likertovy). Dotazník byl cílen na studenty středních škol, kteří tak měli být respondenty toho průzkumu.

Dotazník byl vytvořen pomocí formuláře webové aplikace Google dokumenty a zveřejněn na osobních internetových stránkách autora této práce. E-mailem, který kromě vysvětlení charakteru průzkumu (a jeho cílení) obsahoval odkaz právě na tyto stránky, bylo následně osloveno 24 středních škol na území celé České republiky, u nichž bylo možno výuku v oblasti algoritmizace a programování očekávat (viz. citované RVP na začátku práce).

3.2.3 Získaná data

Na e-mailovou výzvu odpověděly nebo jiným způsobem reagovaly 4 školy (jedna střední průmyslová škola a tři gymnázia). Celkový počet respondentů je 30, z toho 27 respondentů je z jedné střední školy. Toto zásadním způsobem ovlivňuje vyhodnocení průzkumu. Pokládání otázky a získaná data jsou obsaženy v souboru pruzkum_jazyky.xls, který je k této práci přiložen na CD v adresáři pruzkum.

3.2.4 Interpretace

Díky nízkému počtu respondentů a jejich nerovnoměrného zastoupení v rámci škol, které na výzvu reagovaly, musím bohužel konstatovat, že získaná data bohužel nejsou

natolik validní, aby z nich šly vyvodit závěry v rozsahu, který byl stanoven v pracovních cílech průzkumu.

Přinejmenším mohu prohlásit, že data získaná tímto průzkumem nejsou v rozporu s tvrzeními uvedenými v části, jež se zabývá programovacími jazyky na školách (na základě obsahové analýzy textu). Mezi odpověďmi se opakovaně opravdu vyskytly jazyky C a C++ (a jazyky syntaxí příbuzné), Pascal a Object Pascal. Totéž můžeme říci ohledně vývojových prostředí, kde se objevilo Visual Studio od Microsoftu, Dev-C++ (obsahující kompilátor MinGW, které portuje kompilátory sady GCC na platformu MS Windows).

Zajímavým zjištěním je např. to, že se na školách stále používají 16bitová vývojová prostředí Borland Pascal 7 a Turbo Pascal 7, které je dokonce v prvním případě provozované v emulátoru prostředí MS-DOS – programu DOSBox. Je to zvláštní zejména v kontextu toho, že existují i jiné volně šiřitelné kompilátory, ze kterých třeba zmíněný Free Pascal podporuje dialekt Pascalu užitý v těchto prostředích a zároveň je kompatibilní s moderními operačními systémy, pro které je dostupný v 32 i 64bitové verzi.

Další zajímavým faktem je použití programovacího jazyku Logo (a vývojového prostředí Comenius Logo), který byl udán v jednom případě jako hlavní jazyk využívaný pro programování. Jazyk Logo je zajímavý tím, že se řadí mezi propedeutické programovací jazyky, jako takový je dobře přizpůsoben právě pro výuku programování.

4 Python

4.1 Základní rysy Pythonu

Python je dynamicky typovaný interpretovaný jazyk, který je vyvíjen od roku 1989. Je objektově orientovaný. První a originální implementace jazyka, která je někdy také nazývána jako CPython, je napsána v jazyce C. Kromě této existují další implementace, zejména Jython – implementace Pythonu v jazyce Java a IronPython – implementace Pythonu pro .NET framework. Všechny tři zmíněné implementace jsou dostupné bezplatně a zároveň pod jednou z Open source licencí.

Syntaxe Pythonu je jednoduchá na pochopení, díky tomu pak lze dosahovat při psaní programů vysoké produktivity. Zároveň syntaxe nutí k psaní pečlivě strukturovaného kódu, což se ve výsledku pozitivně odráží na přehlednosti, čitelnosti a celkové eleganci hotových programů. I přes to, že se jedná o objektově orientovaný jazyk, je v něm možné psát programy čistě procedurálně, bez konstrukcí typických pro objektové programování.

Originální implementace již v základní instalaci obsahuje balíky a moduly pro práci s databázemi (i jednu konkrétní databázi jako takovou – SQLite), pro tvorbu grafického uživatelského rozhraní (tkinter postavený nad knihovnou Tk) či jednoduché vývojové prostředí (IDLE). Zároveň je dostupná pro veškeré hlavní platformy v oblasti osobních počítačů – Microsoft Windows, GNU/Linux a unixové systémy včetně Mac OS X. Obecně lze proto říci, že jednou napsaný program můžeme bez úprav spustit na kterémkoliv z těchto platforem.

4.1.1 Interpretovaný jazyk

Python je interpretovaný jazyk. Interpretovaný proto, že zdrojový kód programu je vyhodnocován interpretem postupně řádek po řádku. Zdrojový kód ve formě textu tedy není nejprve celý překládán (kompilován) do strojového kódu. Soubory, které obsahují zdrojový kód Pythonu jsou obvykle zakončeny příponou „py“ nebo „pyw“, podle toho zda se jedná o program, při jehož vykonání potřebujeme (první případ) nebo nepotřebujeme konzolový výstup (druhý případ). U takovýchto se souborů se

předpokládá kódování ASCII či UTF-8 (výchozí), které jsou nativně podporovány.

Python je zároveň dynamicky typovaný jazyk (stejně jako velká část interpretovaných jazyků). Typová kontrola je prováděna za běhu programu (během jeho vyhodnocování interpreterem), na rozdíl od statické kontroly, kdy jsou datové typy kontrolovány už při překladu (typické pro kompilované jazyky jako jsou C a C++).

4.1.2 Základy syntaxe

Syntaxe Pythonu bývá obvykle velice úsporná. Je tzv. „Case sensitive“, rozlišuje malá a velká písmena. Výrazným rozdílem mezi Pythonem a ostatními jazyky je způsob seskupení kódu do bloků. V jazycích, které základy své syntaxe čerpají z jazyka C (Java, C#, apod.), se pro takové seskupení používá dvojice složených závorek `{ }`. V Pascalu je analogickou konstrukcí uvedení bloku pomocí klíčového slova `begin` a jeho ukončení pomocí `end`.

Python naopak eliminuje používání závorek na nezbytné minimum. Příkazy jsou do jednoho bloku seskupeny na základě shodného odsazení – shodného počtu mezer či tabulátorů (obecně přijatá konvence je používat pro odsazení čtyř mezer), tyto přitom není možné kombinovat. Příkaz, za kterým následuje nějaký vnořený blok, je obvykle uvozen dvojtečkou. Následující ukázky to ilustrují, zároveň porovnávají vytváření bloků v Pythonu a Pascalu:

```
příkaz:
```

```
    # Komentář, vložený do těla prvního vnořeného bloku
```

```
příkaz:
```

```
    # Komentář, vložený do těla druhého vnořeného bloku
```

```
    # Komentář, vložený do těla prvního vnořeného bloku
```

Ukázka 1: Vytváření bloků zdrojového kódu (Python)

```
prikaz
```

```
begin
```

```
    { Komentar, vlozeny do tela prvnioho vnoreneho bloku }
```

```
prikaz
```

```
begin
```

```
    { Komentar, vlozeny do tela druheho vnoreneho bloku }
```

```
end;  
  { Komentář, vložený do těla prvního vnoreného bloku }  
end;
```

Ukázka 2: Vytváření bloků zdrojového kódu (Pascal)

Dalším prvkem typickým pro výše zmíněné skupiny jazyků je ukončení řádku pomocí středníku. Python toto nepoužívá a maximální délku řádku stanovuje (20) na 79 znaků (doporučených znaků je 72). Oba dva zde popsané koncepty jemným způsobem nutí potenciálního programátora k pečlivosti při psaní a strukturování kódu programu.

Poslední věcí, která vyplynula z výše zmíněné ilustrace, je uvozování komentářů. V Pythonu se tak činní znakem mřížky (#), po kterém musí následovat minimálně jedna mezera.

4.1.3 Základ datových typů

Python je jazykem, který nepoužívá implicitní deklaraci typů. Není tedy nutné použít postup, který se používá např. u jazyků Pascal, C či Java, kdy musíme před použitím proměnné nejprve určit o jaký datový typ se jedná. U Pythonu je tedy typ proměnné určen až okamžikem přiřazení, což odpovídá charakteru dynamicky typovaného jazyka, který byl zmíněn výše.

Protože je Python objektový jazyk, platí, že veškeré proměnné jsou objekty, které jsou odvozeny z typu `PyObject`. Absenci jakékoliv hodnoty pak u proměnné reprezentujeme pomocí `None` (21) – je analogická vůči `null` u Javy (22). Znamená to, že název proměnné neukazuje na žádný konkrétní objekt. Rozvineme-li toto dále, budeme konstatovat, že proměnná je v terminologii Pythonu pojmenovaným odkazem na určitý objekt s danou hodnotou.

Nejlépe je tato skutečnost patrná z následujících ukázek, které ukazují způsob přiřazení do proměnných v jazycích Python a Pascal:

```
a = 7  
b = 7
```

Ukázka 3: Proměnná jako „ukazatel“ na objekt (Python)

```
program promenne;
```

```

var a, b: integer;
begin
    a := 7;
    b := 7;
end.

```

Ukázka 4: Analogický zápis předchozí ukázky (Pascal)

Python používá jako operátor pro přiřazení znak rovnítka (=). Ukázka výše pak tedy znamená, že proměnná s názvem `a` i proměnná s názvem `b` jsou pojmenovanými odkazy na stejný objekt o hodnotě 7.

4.1.4 Základní operátory

Python obsahuje standardní sadu operátorů, která je do určité míry identická s operátory dostupnými v jazyku C. Umožňují však několik specifických a zároveň elegantních způsobů zápisu, které je vhodné zmínit.

Mluvíme-li o aritmetických operátorech, nečeká nás mnoho překvapení. Za vhodné považují zmínit operátory sloužící pro umocňování, dělení, celočíselné dělení a modulo dělení, které mohou být vůči zvyklostem u jiných jazyků odlišné. Jejich použití je zřejmé z následující ukázky:

```

a = 7
b = 2
a/b      # Dělení
a//b     # Celočíselné dělení
a%b      # Modulo dělení
a*b      # Násobení
a**b     # Umocňování

```

Ukázka 5: Zápis vybraných aritmetických operátorů (Python)

Výsledek u dělení bude 3,5 – v případě, že je datový typ dělece a dělitele odlišný, se provede přetypování, stejně tak v případě, kdy dělíme dvě celá čísla – v našem případě tedy dojde k přetypování na datový typ `float`, proto dostáváme výsledek s plovoucí desetinou čárkou.

U celočíselného dělení je výsledek 3, u modulo dělení pak 1, u násobení dostáváme

14, v případě umocnění 49. Pro provedení žádné z těchto operací není třeba importovat žádný dodatečný modul. Užitečným nástrojem je zde funkce `type`, která vrací datový typ proměnné. Rovněž je užitečné, že Python podporuje zkrácené aritmetické operátory, jako jsou např.: `+=`, `-=` nebo `*=`.

Postoupíme-li dále k operátoru přiřazení a operátoru identity, je užitečné představit tuto ukázkou:

```
a = b = 7      # Použití operátoru přiřazení
a is b        # Použití operátoru identity
```

Ukázka 6: Použití operátorů přiřazení a identity (Python)

Použitím operátoru přiřazení nabývá hodnota proměnných `a` i `b` v obou případech 7. Jedná se o pro Python typický úsporný přístup. Výsledkem použití operátoru identity je `True`, protože jak proměnná `a`, tak i proměnná `b` ukazují na stejný objekt.

Jako logické operátory Python používá `and`, `or` a `not`. Jejich význam je analogický operátorům `&&`, `||` a `!` tak, jak jsou definovány v jazyce C, tento způsob zápisu však v Pythonu nelze použít. Naopak následující operátory určené pro porovnávání mají zápis identický (tj.: `==`, `!=`, `>`, `<`, `>=`, `<=`, a to ve významu rovná se, nerovná se, větší, menší, větší či roven, menší či roven). Jejich funkci můžeme ilustrovat na této ukázce:

```
a = 7
a > 0 and a < 10    # První varianta zápisu
0 < a < 10         # Druhá varianta zápisu
```

Ukázka 7: Použití logického operátoru a operátorů pro porovnání (Python)

Obě varianty zápisu zmíněných logických výrazů si jsou ekvivalentní a v obou případech je výsledkem `True`. Druhá varianta ukazuje další z elegantních možností jazyka Python, díky které můžeme zřetězit více operátorů. Tento zápis je ve výsledku mnohem přirozenější a intuitivnější, než jak je tomu u první varianty.

V Pythonu hojně užívaným je dále tzv. operátor příslušnosti `in`, jeho použití se ale úzce váže na oblast seznamů a příbuzných datových typů, proto bude vysvětlen v další

části. Kompletní výčet operátorů (včetně operátorů pro bitové operace), které jsou dostupné v jazyce Python, se nachází v jeho dokumentaci (23).

4.1.5 Seznamy, n-tice a pole

Svým přístupem k datovým strukturám, které jsou v jiných jazycích definovány jako pole či kolekce, resp. ke strukturám jim podobným (24), je Python do značné míry specifický a zaslouží si proto pozornost. Mark Summerfield udává o typech představujících posloupnosti, kam většinu z výše zmíněných datových typů řadí, toto:

„Typ představující posloupnost je takový typ, který podporuje operátor příslušnosti (`in`), funkci zjišťující velikost (`len()`) a řezání(`[]`) a který je iterovatelný. Jazyk Python nabízí pět vestavěných typů představujících posloupnost: `bytearray` (pole bajtů), `bytes` (bajty), `list` (seznam), `str` (řetězec) a `tuple` (n-tice).“ (25)

Protože se jedná o komplexní problematiku, která je dalece nad rámec této práce, budu se z těchto datových typů dále věnovat n-ticím, seznamům a slovníkům (ty Summerfield řadí mezi datové typy představující mapování, od posloupností se značně liší), které pro pochopení problematiky považuji za nejdůležitější (stejně tak se zde věnuji tématům, která se dotýkají pouze základů práce s nimi).

Začneme-li n-ticemi a seznamy, zjistíme, že se jedná o datové typy, které jsou si do značné míry podobné. Zásadním rozdílem mezi nimi je možnost editace. Zatímco n-tice jsou neměnné, takže nemůžeme mazat jejich prvky či je měnit, seznamy toto naopak umožňují. Pomocí funkcí `list` a `tuple` lze převádět n-tici na seznam a obráceně. N-tici a seznam, resp. způsoby, kterým je indexujeme, jsou vidět ve vzorové ukázce (opět v porovnání s jazykem Pascal, tentokrát s jeho syntaxí poli):

```
n = (1, "slovo", 2, ("vnořená", "n-tice")) # Zápis n-tice
s = [3, "popis", 4, "název"]             # Zápis seznamu
n[1]                                     # První indexace prvku
n[3][0]                                  # Druhá indexace prvku
s[2]                                     # Třetí indexace prvku
s[-2]                                    # Čtvrtá indexace prvku
```

Ukázka 8: Práce s n-ticí a seznamem (Python)

```

program pole;
var j: array[1..7] of Integer;
var d: array[1..7,1..7] of Integer;
begin
    j[1] := 16;      { Prirazeni hodnoty do jednorozmerneho pole }
    d[3,5] := 64;   { Prirazeni hodnoty do dvourozmerneho pole }
    writeln(j[1]);  { Výpis hodnoty prvku pole }
    writeln(d[3,5]);      { Výpis hodnoty prvku pole }
end.

```

Ukázka 9: Práce s jedno a dvourozměrným polem (Pascal)

Prvkem n-tice nebo seznamu může být jakýkoliv datový typ, takže můžeme do jedné n-tice vnořit n-tici další tak, jak je to znázorněno v ukázce výše. Jednotlivé prvky v n-tici nebo seznamu můžeme indexovat pomocí celého čísla, které odpovídá pozici konkrétního prvku. Číslování probíhá od zleva od nuly a zprava od posledního prvku. První prvek má tedy index 0, druhý 1 apod. (indexujeme zleva), poslední prvek má vždy index -1, předposlední -2 apod. (při indexování zprava).

Popíšeme-li si ukázkou č. 8, pak u první indexace prvku bude výstupem řetězec „slovo“. Stejný výstup bychom dostali, kdybychom indexovali číslem -3. U druhé indexace bude výstupem řetězec „vnořená“, protože jsme indexovali první prvek n-tice (index 0), která se sama nalézá na čtvrté pozici n-tice n (index 3). Třetí a čtvrtá indexace prvku bude mít výstup totožný, v obou případech jím bude celé číslo 4 (index 2 odkazuje na třetí prvek zleva, index -2 pak na druhý prvek zprava).

N-tice a seznamy můžeme řezat, porovnávat, řadit, připojovat další prvky (pomocí funkce `append`), spojovat (pomocí operátoru `+`), můžeme na ně uplatnit operátor příslušnosti `in` a mnoho dalšího. V případě seznamu pak mazat a měnit jednotlivé prvky. Některými z těchto směrů se ubírá i další ukázkou:

```

s = [1, 2, 3, 4]           # Zápis seznamu s
n = ("a", "b", "c", "d") # Zápis n-tice n
2 in s                    # Použití operátoru příslušnosti
s[1] = n                  # Vložení n-tice n do seznamu s

```

Ukázka 10: Operace s n-ticemi a seznamy (Python)

V této ukázce jsme si vytvořili jeden seznam a jednu n-tici. Použitím operátoru příslušnosti `in` se nám dostane výsledku `True` nebo `False` v závislosti na tom, zda se prvek v n-tici či seznamu nachází nebo nenachází. Protože se číslo 2 v seznamu `s` nachází, je vrácena hodnota `True`. V další části vkládáme na druhou pozici (index 1) seznamu `s` n-tici `n`. Celá n-tice `n` tak na této pozici nahradí číslo 2, jednotlivé prvky vnořené n-tice opět můžeme indexovat jako v předchozím případě. Pokud bychom zkusili opačný postup, a vkládali tak seznam do n-tice, došlo by k vyvolání výjimky.

```
n[1:4]          # První řezání
n[-3:]         # Druhé řezání
n[:3]          # Třetí řezání
s[1:1] = [7]   # Vložení prvku do seznamu za použití řezání
s[:] = [7]     # Nahrazení všech prvků seznamu za použití řezání
```

Ukázka 11: Řezání n-tic a seznamů (Python)

Řezáním můžeme vybírat části stávajících n-tic či seznamů, pomocí operátoru `+` je pak snadno spojovat do n-tic nových. Výstupem prvního řezání v ukázce č. 11 (která dále rozvíjí ukázkou č. 10) bude n-tice, která obsahuje prvky s indexy 1, 2 a 3 – čili: „b“, „c“, „d“. Výstupem druhého řezání bude identická n-tice jako v předchozím případě, protože neudáním parametru za dvojtečkou jsme vytvořili n-tici obsahující veškeré prvky n-tice `n` počínaje prvkem s indexem -3 (ten odpovídá prvku s indexem 1). U třetího řezání pak získáme n-tici, která obsahuje všechny prvky n-tice `n` předcházející prvek s indexem 3.

Zajímavým použitím řezání je vkládání mezi již existující prvky tak, jako je tomu u vložení prvku za použití řezání. V tomto případě dojde k vložení čísla 7 do seznamu `s` na index 2, ostatní prvky se posunou doprava. Stejným způsobem bychom mohli vybrat libovolnou část seznamu a nahradit ji jiným seznamem s jiným počtem prvků, než kolik měla vybraná část upravovaného seznamu. Tato situace je v extrémním případě ilustrována jako poslední. N-tice je vybrána a nahrazena celá, protože jsme kromě dvojtečky neudali žádný parametr.

V našem výčtu se ještě krátce podíváme na datový typ slovník. V jiných jazycích je

přítomen pod názvem asociativní pole, což je výraz, který jeho funkci ve stručnosti poměrně přesně vystihuje. Slovníky nejsou vnitřně uspořádané – neobsahují indexové pozice, proto je nelze řezat ani krokovat (26).

Nejlepším prostředkem pro znázornění funkce slovníků je další ukázka:

```
# Vytvoření slovníku
slovník = {'a':'Toto', 'b':'je', 'c':'slovník'}
slovník["a"]           # Vypsání prvku slovníku
slovník["b"] = (1, 2, 3) # Úprava slovníku
slovník["b"][0]        # Indexace prvků vnořené n-tice
```

Ukázka 12: Práce se slovníkem (Python)

Jak je vidět na ukázce výše, slovník se definuje pomocí jedné či více položek oddělených čárkou, které dodržují syntaxi 'klíč':'hodnota', které jsou uzavřeny ve složených závorkách. K hodnotě přistupujeme dle zadaného klíče, proto při indexaci řetězcem „a“ dostaneme řetězec „Toto“. Stejně klíče se nemohou opakovat. Hodnotou slovníku může být jakýkoliv datový typ, proto je možné, jak je znázorněno v naší ukázce, do něj vložit i n-tici. Její prvky pak už lze indexovat normálním způsobem, jak je tomu u předchozích ukázek a jak je to znázorněno i v ukázce č. 12.

4.1.6 Podmínky

Pro řízení toku programu se v Pythonu užívá standardní konstrukce `if - else` (nebo také `if - elif - else`), která je v různých variacích typická pro mnoho dalších programovacích jazyků. Python neobsahuje konstrukci analogickou `switch - case` u jiných jazyků. Zápis podmínky `if - else` je odvislý od struktury uvedené v ukázce č. 13.

Konstrukce podmínky pak může vypadat například takto:

```
promenna = 50
if promenna > 10:
    # Komentář u první podmínky
elif promenna > 100:
    # Komentář u druhé podmínky
else:
```

```
# Komentář pro ostatní případy
```

Ukázka 13: Zápis podmínky (Python)

V uvedené ukázce je jako pravdivá vyhodnocena první podmínka, v reálné situaci by tak byl vyhodnocen kód, který by na stejné úrovni odsazení následoval komentář u první podmínky.

V programovacích jazycích, jejichž syntaxe je inspirovaná jazykem C, je možné užití podmínky v kombinaci s operátorem přiřazení – dohromady tvoří tzv. ternární operátor. V praxi se ho používá pro úspornost zápisu, protože pracujeme pouze se třemi operandy. Předvádím to na následující ukázce, jejímž obsahem je kód v jazyce C:

```
#include <stdio.h>
int main(void) {
    int znamka = 2;
    char* vysledek = (znamka < 5) ? "Prosel" : "Neprošel";
    printf("%s", vysledek); // Vypis pomoci knihovni funkce
    return 0;
}
```

Ukázka 14: Použití ternárního operátoru (C)

Do proměnné `vysledek` se přiřadí řetězec na základě vyhodnocení podmínky uvedené v závorce. Je-li podmínka vyhodnocena jako pravdivá, přiřadí se proměnné `vysledek` hodnota uvedená na prvním místě, tj. hned za znakem otazníku, v opačném případě dojde k přiřazení hodnoty, která následuje znak dvojtečky. V tomto konkrétním případě tedy proměnná `vysledek` nabývá hodnoty „Prosel“.

V jazyce Python můžeme použít konstrukci, která je použitím ternárního operátoru do jisté míry analogická:

```
znamka = 2
vysledek = "Prošel" if znamka < 5 else "Neprošel"
```

Ukázka 15: Analogie k ternárního operátoru (Python)

Vyhodnocení kódu je identické s ukázkou č. 14 v jazyce C. Je-li podmínka pravdivá, je vrácena původní hodnota proměnné `vysledek`, pokud ne, je přiřazena hodnota uvedená za `else`. Pokud bychom nepoužili této zkrácené formy zápisu,

vypadal by kód v Pythonu s identickou funkcionalitou takto:

```
znamka = 2
if znamka < 5:
    vysledek = "Prošel"
else:
    vysledek = "Neprošel"
```

Ukázka 16: Zápis podmínky bez 'ternárního' operátoru (Python)

Podrobnější popis podmínek je obsažen v dokumentaci Pythonu 3, sekci 7.1 (27).

Python obsahuje, stejně jako mnoho jiných jazyků, konstrukci pro odchyčení výjimek, zde konkrétně `try - except` (nebo také `try - except - else - finally`). Výjimky jsou typicky generovány, pokud dojde při běhu programu k chybě či jinému výjimečnému stavu.

Pomocí konstrukce `try - except` tak lze zamezit situaci, v níž by mohlo dojít k nesprávné funkci programu nebo dokonce jeho pádu. Ilustruje to následující ukázka:

```
try:
    cislo = int(input("Zadejte cele cislo: "))
except:
    print("Nezadali jste cele cislo!")
```

Ukázka 17: Ošetření výjimky (Python)

V této ukázce používáme tři nové funkce. Funkci `input` pro přijetí dat ze standardního vstupu, funkci `print` pro výpis na standardní výstup a funkci `int` pro převod zadané proměnné na celočíselný datový typ, který je tudíž její návratovou hodnotou.

Pokud zadáme hodnotu, která není celým číslem, dojde k vyvolání výjimky. Program následně provede blok kódu následující klíčové slovo `except`. Pokud nedojde k vyvolání výjimky (zadaná hodnota byla celočíselná), provede se pouze blok kódu následující klíčové slovo `try`. V obou případech vykonávání programu posléze pokračuje.

Komplexní problematika výjimek je podrobně popsána v oficiální dokumentaci

Pythonu 3, sekci 7.3 (27), popř. v další části oficiálních stránek (28).

4.1.7 Cykly

Python obsahuje dva typy cyklů. Jsou jimi cykly `while` a `for`. Jejich použití je v případě cyklu `while` analogické s použitím v jiných jazycích. Python neobsahuje cyklus `do - while`, který je typickým prvkem u mnoha jazyků.

Následující ukázka ilustruje použití cyklu `while` v Pythonu:

```
cislo = 7
while cislo != 0:
    print(cislo)
    cislo -= 1
```

Ukázka 18: Zápis cyklu typu `while` (Python)

Výše zmíněný cyklus se provádí dokud je splněna podmínka uvedená v zápisu. Výpisem takového programu je potom řada čísel začínající číslicí 7 a končící číslicí 1.

Cyklus `for` je svým charakterem i zápisem podobný cyklu `foreach`, a to tak, jak je implementovaný např. v jazycích C# nebo PHP. Samotný cyklus `foreach` proto Python neobsahuje. Cyklus `for` je u Pythonu poměrně flexibilní konstrukcí, která se často kombinuje s funkcí `range`. Ta např. umožňuje pomocí krátkého zápisu napodobit `for` cyklus tak, jak je obsažen v jazycích na bázi jazyka C.

Funkci tohoto typu cyklu (v porovnání s `for` cyklem jazyka C) lze osvětlit následujícími ukázkami:

```
vzory = {"město", "moře", "kuře", "stavení"}
for vzor in vzory:
    print(vzor)
```

Ukázka 19: Jednoduchý zápis cyklu typu `for` (Python)

```
#include <stdio.h>
int x;
char* vzory[] = {"město", "moře", "kuře", "stavení"};
int main(void) {
    for(x = 0; x <= 3; x++){
```

```

        printf("%s\n", vzory[x]); // Vypis pomoci knihovni funkce
    }
    return 0;
}

```

Ukázka 20: Analogický zápis předchozí ukázky (C)

Cyklus v ukázce č. 19 postupně prochází všechny prvky zadané n-tice `vzory` a přiřazuje je do proměnné `vzor`. Výpisem takového programu jsou potom všechny prvky n-tice `vzory`.

Dalšími častými požadavky v algoritmizaci a programování jsou for cykly o přesně daném počtu průběhů, cykly, ve kterých iterovaná proměnná nabývá hodnot v přesně daném rozsahu, a také cykly, kde se tato proměnná mění podle zadaného kroku. Právě pro tyto účely se v Pythonu používá zmíněná funkce `range`:

```

for x in range(6): # Použití funkce range s jedním parametrem
    print(x)
for y in range(10, 20): # Použití funkce range se dvěma parametry
    print(y)
for z in range(0, 10, 2): # Použití funkce range se třemi parametry
    print(z)

```

Ukázka 21: Různé zápisy for cyklu s použitím funkce `range` (Python)

Výpisem `for` cyklu používající funkci `range` s jedním parametrem bude posloupnost čísel od 0 do 5 (parametr funkce `range` zde udává počet průběhů). Při použití funkce `range` se dvěma parametry pak posloupnost čísel od 10 do 19 (parametry funkce `range` zde udávají počáteční a koncovou hodnotu mimo), `for` cyklus s funkcí `range` se třemi parametry pak dává řadu čísel 0, 2, 4, 6, 8 (platí, co v předchozím případě, třetí parametr udává krok).

4.1.8 Procedury a funkce

Procedury a funkce jsou v Pythonu jednoduchou konstrukcí, která je podobná konstrukcím z jiných jazyků. Chceme-li definovat funkci nebo proceduru, uvodíme ji klíčovým slovem `def`, za nímž následuje název doplněný o jednoduché závorky, které

uvnitř mohou obsahovat parametry. Proceduru či funkci voláme jejím názvem. Typický zápis procedury v Pythonu (v porovnání se zápisem téhož v Pascalu) vypadá takto:

```
def procedura():
    print("Jsem procedura")    # Funkce v těle procedury
procedura()                    # Volání procedury
```

Ukázka 22: Jednoduchá procedura a její volání (Python)

```
program proc;
procedure procedura;
begin
    writeln('Jsem procedura');
end;
begin
    procedura;
end.
```

Ukázka 23: Analogický zápis předchozí ukázky (Pascal)

Pokud jsou procedury z ukázek č. 22 a 23 volány, vždy se vypíše text „Jsem procedura“. Neobsahují žádné vstupní parametry – pouze zabalují funkci `print` (resp. `writeln`). Ukázka č. 24 už je komplexnější:

```
def fibonacci(n, a = 0): # Výpočet Fibonacciho posloupnosti
    fib = [a, a + 1]
    for x in range(n - 2):
        fib.append(fib[-1] + fib[-2])
    return fib
fibonacci(7) # Volání funkce
```

Ukázka 24: Funkce s parametry a její volání (Python)

Funkce `fibonacci` má dva parametry a slouží k výpočtu volitelného počtu čísel Fibonacciho posloupnosti. Parametr `n` je povinný a udává kolik Fibonacciho čísel má být spočítáno. Parametr `a`, který značí hodnotu prvního členu posloupnosti, povinný není, jeho výchozí hodnotou je 0. Celá posloupnost je postupně ukládána do seznamu `fib`, který je také návratovou hodnotou funkce.

Je-li funkce z ukázky č. 24 volána, výstupem je seznam s prvky 0, 1, 1, 2, 3, 5, 8.

Proměnná `x` není v těle cyklu použita, v tomto konkrétním případě proto může být nahrazena znakem podtržítka.

S funkcemi a procedurami souvisí problematika lokálních a globálních proměnných, kterou ilustruje další ukázka:

```
prom = 7
def globalni():      # Vypis hodnoty globální proměnné
    print(prom)
def lokalni():      # Vypis hodnoty lokální proměnné
    prom = 8
    print(prom)
def prepis():       # Přepsání hodnoty globální proměnné
    global prom
    prom = 9
globalni()
lokalni()
print(prom)        # Vypis hodnoty globální proměnné
prepis()
print(prom)        # Vypis hodnoty globální proměnné
```

Ukázka 25: Obory platností proměnných (Python)

Všechny procedury v předchozím případě pracují s proměnnou se stejným názvem, ale každá jiným způsobem. Zatímco první procedura `globalni` pouze vypíše hodnotu proměnné `prom`, která je nastavena na hodnotu 7 mimo tělo procedury samotné, v proceduře `lokalni` je do proměnné `prom` přiřazena hodnota 8.

Hodnota 8 je také posléze vypsána funkcí `print`, avšak není tím změněna hodnota globální proměnné `prom` (ta je stále 7), ale pouze hodnota lokální proměnné `prom`, která má obor platnosti jen v těle procedury `lokalni`. Obory platnosti obou proměnných `prom` se tedy překrývají, v těle procedury však má přednost lokální proměnná a její hodnota.

To se liší od další situace, ve které voláme proceduru `prepis`. Rozdíl je v použití klíčového slova `global`. Díky tomu je dále přiřazováno přímo do globální proměnné `prom`. Ta tak nabývá hodnoty 9, jak si můžeme ověřit jejím vypsáním funkcí `print`

zcela na konci ukázky č. 25.

Python obsahuje několik zajímavých prvků jako jsou lambda funkce nebo operátor pro rozbalení posloupnosti (*), díky kterému je možné vytvořit funkce a procedury s předem neznámým počtem proměnných. Tyto vlastnosti již ale rozhodně nepatří ke zde probíraným základním rysům Pythonu.

4.1.9 Objektový model

V této části si popíšeme základy práce s třídami a jejich instancemi v Pythonu, a to hlavně ze syntaktického hlediska. Rozsáhlejší příklad, které demonstuje objektový přístup k programování, je obsahem kapitoly 6, která se výhradně věnuje příkladům v jazyce Python.

Chceme-li vytvořit třídu, která bude obsahovat několik proměnných, konstruktor a metodu pracující s proměnnými třídy, můžeme tak učinit následujícím způsobem:

```
class trida:
    def __init__(self, a = 0, b = 0):          # Konstruktor třídy
        self.a, self.b = a, b              # Další možný způsob přiřazení
    def vynasob(self):                       # Metoda implementující násobení
        return self.a * self.b
instance = trida(2, 4)                      # Vytvoření instance třídy
vysledek = instance.vynasob()              # Volání metody instance třídy
print(vysledek)                            # Vypsání výsledku
```

Ukázka 26: Základy objektového modelu (Python)

Pro porovnání složitosti zápisu syntaktických konstrukcí v objektovém programování si můžeme ukázat, jak by vypadal kód v C++ s identickou funkcionalitou (tj. s funkcionalitou ukázky č. 26 v Pythonu):

```
#include <iostream>
using namespace std;
class Trida{
    int a;
    int b;
public:
    /* Deklarace konstruktoru třídy */
```

```

        Trida(int ia, int ib);
        /* Deklarace metody implementující násobení */
        int vynasob();
};
/* Definice konstrukturu */
Trida::Trida(int ia = 0, int ib = 0){
    this->a = ia;
    this->b = ib;
}
/* Definice metody pro násobení */
int Trida::vynasob(void){
    return this->a * this->b;
}
int main() {
    /* Vytvoření instance třídy */
    Trida instance = Trida(2, 4);
    /* Volání metody instance třídy */
    int vysledek = instance.vynasob();
    /* Výpis výsledku pomocí knihovní funkce */
    cout << vysledek << endl;
    return 0;
}

```

Ukázka 27: Analogický zápis předchozí ukázky (C++)

V ukázce č. 26 jsme si nadefinovali třídu (pomocí klíčového slova `class`). Ta obsahuje dvě vlastní proměnné `a`, `b`, konstruktorem (nastavující hodnoty těchto proměnných) a metodu `vynasob`, která tyto dvě proměnné dokáže vynásobit. Klíčové slovo `self` má stejný význam jako `this` v jazycích C++, Java nebo C#. Jedná se o povinný parametr u každé metody třídy. Pro přístup k proměnným a metodám třídy (nebo metodám instance třídy) přistupujeme pomocí operátoru tečky (`.`). Pro vytvoření instance třídy používáme operátor přiřazení, kdy na levé straně uvádíme název instance, na pravé straně pak název třídy, na jejímž základě je instance vytvářena.

Výsledkem je v našem případě číslo 8. Pokud bychom ponechali parametry konstrukturu při vytváření instance prázdné, budou použity výchozí hodnoty a výsledek

by byl 0. V případě, že bychom chtěli na základě této existující třídy vytvořit třídu novou s širší funkcionalitou, doplnili bychom předchozí ukázkou č. 26 o takovýto kód:

```
class odvozena(trida):    # Vytváření odvozené třídy
    def umocni(self):    # Metoda implementující umocňování
        return self.a ** self.b
instance2 = odvozena(2, 4)
vysledek2 = instance2.umocni()
print(vysledek2)
```

Ukázka 28: Vytvoření odvozené třídy (Python)

V odvozené třídě jsme nemuseli opět implementovat konstruktor, který je zděděn (stejně jako metoda `vynasob`) v nezměněné podobě z původní třídy. Pouze jsme tedy přidali metodu `umocni`. Výsledek této metody u vytvořené instance z nové odvozené třídy pak mít hodnotu 16, protože číslo 2 je umocňováno na čtvrtou. I v této instanci bychom mohli použít metodu `vynasob` – se stejným výsledkem jako v předchozím případě (díky shodným parametrům předávaným konstruktorem).

4.1.10 Moduly

Modul je v jazyce Python možností, jak rozdělit zdrojový kód do více souborů. Vyplývá z toho jednak větší přehlednost kódu, jednak dobrá znovupoužitelnost takového kódu. Moduly mohou být v sduřovány do balíčků. V této části jsou popsány základní způsoby práce s moduly a vytvoření jednoho demonstrativního modulu. Seznam a popis některých důležitých modulů (ať již obsažených v Pythonu nebo modulů třetích stran) je obsažen v části 5.2 Nástroje a moduly.

Moduly v Pythonu tedy můžeme jednak sami vytvářet, jednak si díky nim zpřístupnit rozsáhlou funkcionalitu obsaženou ve standardní knihovně Pythonu. Dva způsoby jak to provést jsou zde:

```
# První metoda
import os                # Import modulu
os.system("cls")        # Volání konkrétní funkce
# Druhá metoda
from os import *        # Import všech prvků modulu
```

```
system("cls") # Volání konkrétní funkce
```

Ukázka 29: Dva způsoby práce s moduly (Python)

Rozdíly v obou použitých způsobech jsou patrné. Jak v prvním, tak i druhém případě voláme funkci modulu `os`, která slouží pro předání příkazu přímo operačnímu systému (zde předáváme příkaz pro smazání obrazovky v konzoli, který platný pouze v prostředí MS Windows – na GNU/Linuxu bychom museli nahradit předávaný příkaz `cls` příkazem `clear`). Zatímco první metoda nám prostředky modulu pouze zpřístupní, druhá nám umožní používat celý jmenný prostor daného modulu přímo (popř. balíčku v závislosti na konkrétním případě).

Nevýhodnou prvního přístupu je o něco delší zápis, u druhého pak nebezpečí kolize názvů. Samozřejmě ve druhém případě nemusíme používat hvězdičku (díky které je importováno vše v daném modulu, co lze naimportovat), ale místo ní importovat pouze příslušný prvek.

Obdobným způsobem můžeme pracovat i s moduly vlastními. Představme si, že máme dva soubory – `run.py` (který obsahuje samotný program) a `doplnek.py` (rozšiřující modul):

```
# Soubor doplnek.py
def ctverec(a): # Implementace funkce v modulu
    return a*a

# Soubor run.py
import doplnek # Import modulu
obsah = doplnek.ctverec(3) # Použití funkce z modulu doplnek
print(obsah)
```

Ukázka 30: Práce s vlastním modulem (Python)

V souboru modulu (`doplnek.py`) jsme implementovali funkci `ctverec`, která vrací obsah čtverce. V souboru samotného programu, který spouštíme (`run.py`), jsme poté tento modul naimportovali a použili v něm definovanou funkci. Na základě tvrzení v předchozích odstavcích by bylo možné použít i zápis takového tvaru:

```
# Soubor run.py
```

```
from doplněk import ctverec
obsah = ctverec(3) # Přímé použití funkce z modulu doplněk
print(obsah)
```

Ukázka 31: Alternativní zápis importu modulu (Python)

Výsledkem je v obou případech 9. Problém by nastal v okamžiku, kdybychom měli v souboru `run.py` definovanou další funkci se shodným názvem s některou s funkcí modulu, v našem případě `ctverec`. V takovém případě by se provedla funkce, která je implementovaná v přímo spouštěném souboru `run.py`.

4.2 Historie jazyka

4.2.1 Vznik a design Pythonu

První verze Pythonu řady 1 (podrobnější informace o verzích Pythonu jsou uvedeny dále v části 4.3, která se zabývá jednotlivými implementacemi Pythonu) byla vyvíjena od roku 1989. Verze 1.0.0 pak byla vydána roku 1994 (29). Jejím autorem, stejně jako autorem Pythonu celkově, je původem nizozemský programátor Guido van Rossum.

Cíle, kterých chtěl s Pythonem dosáhnout blíže specifikoval v těchto bodech:

- „vytvořit snadný a intuitivní jazyk, který je zároveň dostatečně mocný, aby obstál mezi hlavními konkurenty
- mít otevřený kód, ať se může každý zapojit do jeho vývoje
- umožnit psát kód, který je srozumitelný jako běžná angličtina
- vytvořit jazyk vhodný pro běžné každodenní úkoly, umožňující vývoj v krátkém čase“ (30)

Na svém blogu, ve kterém se autor prioritně zaměřuje právě na historii Pythonu, pak nalezneme i zajímavé vyjádření k inspiraci v jiných jazycích – konkrétně v často nápadné podobnosti se syntaxí a jmennými konvencemi jazyka C. Guido van Rossum o tom říká následující:

„Když se lidé poprvé seznámí s Pythonem, jsou často zaujati tím, jak kód napsaný v Pythonu vypadá – přinejmenším na povrchu je podobný kódu napsaném v dalších

konvenčních programovacích jazycích – jako jsou C nebo Pascal. Není to žádná náhoda – syntaxe Pythonu je z velké části převzatá z jazyka C. Například, mnoho z klíčových slov v Pythonu (`if`, `else`, `while`, `for`, etc.) jsou stejné jako v jazyce C, jména identifikátorů jsou tvořena podle stejných pravidel jako tomu je v jazyce C, mnoho standardních operátorů má shodný význam v jazyce C. Jistě, Python není C - jednou z hlavních oblastí, kde se liší, je použití jednotného odsazení místo závorek pro vytvoření bloku.“ (31)

Navážeme-li na několik výše v textu zmíněných bodů, které si Guido van Rossum stanovil jako cíle při vývoji Pythonu, můžeme pokračovat dalšími, kterými se řídil přímo při vývoji jazyka a která měla – jak sám konstatuje – hlavně šetřit čas (32):

- „Půjčuj si nápady odkudkoliv jen to jde - pokud to dává smysl.
(...)
- Dělej jednu věc dobře ('Unixová' filosofie).
- Nerozčiluj se příliš kvůli výkonu - když to bude potřeba uděláš optimalizaci později.
(...)
- Nesnažte se o dokonalost, protože 'dost dobré' je často právě tak akorát.“ (32)

Přeneseme-li se o něco dále ve stejném zápisku blogu, nalezneme informace, ve kterých autor přímo popisuje jednotlivé designové prvky Pythonu – vlastnosti, kterých chtěl dosáhnout v kontradikeci s vlastnostmi, kterých naopak dosáhnout nechtěl (a jimiž jsou jiné jazyky typické). Znějí takto:

- „Implementace Pythonu by neměla být spjata s konkrétní platformou. Je v pořádku, když určitá funkcionalita není vždy dostupná, ale klíčové věci by měly být funkční všude.
- Neobtěžuj uživatele s detaily, které může zvládnout stroj (Tímto pravidlem jsem se vždy neřídil a mělo to za následek několik katastrofálních následků, jež jsou popsány v dalších sekcích).

- Podporuj a udržuj platformně nezávislý kód, ale neodřezávej přístup k jejím schopnostem či vlastnostem (Toto je v ostrém kontrastu k Javě).
- Velký komplexní systém by měl obsahovat mnoho úrovní rozšiřitelnosti. Jsou tím maximalizovány možnosti pro uživatele - ať už sofistikované či nikoliv - aby si mohli pomoci sami.
- Chyby by neměly být osudové. To znamená, že uživatelský kód by měl být vždy schopný obnovy z podmínek, v nichž dochází k chybám, a to dokud je virtuální stroj jazyka dosud funkční.
- Stejně tak by však chyby neměly jen tiše přijít a odplynout (Tyto dva poslední body vedly k rozhodnutí použít systém výjimek napříč implementací).
- Chyba v uživatelském kódu by neměla být schopna zapříčinit nepředvídatelné chování interpreteru Pythonu. Zásadní selhání není nikdy chybou uživatele.“ (32)

Čtení těchto slov je nesmírně zajímavé už proto, že je názorně vidět, jak silně byla tato předsevzetí dodržena. Není sice možné si tím být zcela jist – už kvůli datu vzniku, jímž počátek roku 2009 (čili po letech vývoje jazyka), ale i tak můžeme třeba na příkladu třetího bodu (o platformě nezávislém kódu) vidět, že v situaci, ve které je dnes Python často používán pro psaní systémových skriptů, se tomuto bodu velmi dostalo.

4.3 Implementace

4.3.1 Python (CPython)

Jedná se o první a původní implementaci Pythonu, jejímž autorem je přímo Guido van Rossum. Jde o s největší pravděpodobností nejvíce rozšířenou implementaci Pythonu.

Je napsána v jazyce C (odtud další často používaný název – CPython). Od počátku se jedná o volně šiřitelný (Open source) projekt – licencovaný pomocí Python Software Foundation License (PSFL) – licence s otevřeným zdrojovým kódem. Tato implementace je oficiálně dostupná pro operační systémy MS Windows, GNU/Linux,

unixové systémy včetně MAC OS X.

Python byl v průběhu času distribuován ve třech verzích. První verze jsou dnes již nepodporované, na rozdíl verzí řad 2 a 3, které v současnosti tvoří dvě stabilní vzájemně nekompatibilní větve vývoje. Řada 2 je v současnosti udržována hlavně co se týká bezpečnostních oprav. Taktéž jsou do ní zařazovány některé novinky, které byly původně obsaženy pouze v řadě 3 (týká se to zejména vydání verze 2.6).

Řada 3 (označovaná také jako Python 3000 nebo P3k) je kuproti tomu dalším milníkem ve vývoji Pythonu, kam jsou primárně zařazovány nové vlastnosti. Tato řada si klade za úkol zejména pročistit syntaxi Pythonu, změny jsou většinou spíše evolučního charakteru, nicméně, jak již bylo řečeno, i tak není zachována zpětná kompatibilita s verzemi řady 2. Důležitými změnami jsou např. změny v podpoře Unicode, který je nyní implicitním datovým typem veškerých řetězců, dále změny názvů některých funkcí a chování některých operátorů – např. operátorů pro dělení, resp. celočíselné dělení (33).

4.3.2 IronPython

Jedná se o implementaci Pythonu pro .NET framework a kompatibilní prostředí (Mono). IronPython je napsán v jazyce C#, i tak je výkonově srovnatelný s CPythonem (34). Rozdíl ve výkonu je odvislý od konkrétní úlohy. Díky tomu je možné provozovat IronPython na více platformách než jen na MS Windows, pro kterou je framework .NET vyvinut. Množství podporovaných platform je shodné s těmi, které jsou uvedené v předchozí části o implementaci CPythonu, protože je pro ně dostupná některá verze projektu Mono (35).

Cílem projektu je skloubit vlastnosti Pythonu s možnostmi .NET frameworku. Program napsaný v IronPythonu tak může pracovat s objekty prostředí .NET a využívat jejich možností. Například tedy lze přímo použít rozhraní Window Forms pro vytváření grafických a formulářových aplikací (36).

Stejně jako CPython je tento projekt volně šiřitelný Open source projekt, který je v tomto případě licencován pod Apache License 2.0 (Apache). IronPython se snaží

o kompatibilitu s jazykem Python jako takovým. Verze IronPythonu 2.6 by podle webu projektu měla funkčně odpovídat identicky značené verzi Pythonu (37).

4.3.3 Jython

Projekt Jython má cíle do značné míry identické s IronPythonem, tentokrát však jde o projekt orientovaný na platformu Java (38). Ve stejnojmenném jazyce je také Jython napsán. Vlastnosti Javy jsou určující pro výkon, který je opět v porovnání s CPythonem v různých úlohách o něco rychlejší nebo pomalejší. Oba projekty dále spojuje i osoba zakladatele projektu, kterým je Jim Hugunin.

Dostupnost Javy – přesněji Java Virtual Machine – pro konkrétní operační systém pak určuje, kde je možné Jython provozovat (39). Jedná se o MS Windows, GNU/Linux, Solaris a Mac OS X.

Díky provázanosti s platformou Java je možné využít třídy, které jsou v ní dostupné. Použijeme-li příklad analogický tomu, který je uveden u IronPythonu, pak je možné vytvořit grafické uživatelské rozhraní pomocí toolkitu Swing.

Jython implementuje vždy danou verzi jazyka Python, takže verze Jythonu 2.5 by měla odpovídat Pythonu 2.5 (40). Nejsou však implementovány veškeré prvky – není například možné v Jythonu použít modul napsaný v jazyce C. Licence Jythonu je identická s licencí CPythonu – jde o Python Software Foundation License (PSFL). Jython je tedy volně šiřitelný Open source projekt.

4.4 Základy použití

4.4.1 Instalace Pythonu

Python je možné získat přímo z oficiálních stránek Pythonu, kde nalezneme instalátory pro MS Windows i Mac OS X. V prostředí GNU/Linuxu nalezneme Python s největší pravděpodobností vždy v repozitáři konkrétní distribuce. Veškeré ukázky a příklady uvedené v bakalářské práci byly testovány v prostředí MS Windows 7 (64bitová edice) a Debian GNU/Linux 6 (32bitová verze). Vše s Pythonem verze 3.1. V prostředí GNU/Linuxu nemusí být (spolu s balíkem obsahujícím přímo Python)

nezbytně nainstalovány veškeré dodatečné balíky nutné pro plnou funkčnost těchto příkladů (např. tkinter).

Příklady a ukázky v této práci dále očekávají, že v prostředí operačního systému MS Windows se interpreter Pythonu nachází v adresáři C:\Python31. V GNU/Linuxu je pak nutné mít vhodně nakonfigurovanou systémovou proměnnou PATH. Není-li tomu tak, je nutné tyto příklady poupravit dle vlastní konfigurace instalace, popř. spouštět jiným možným způsobem, jak je popsáno dále.

4.4.2 Interpreter příkazů

Interpreter Pythonu je mocným nástrojem už sám o sobě – aniž bychom jeho prostřednictvím spouštěli konkrétní v nějakém souboru obsažený program. Protože Python obsahuje moduly, které mají úzkou vazbu na operační systém, na němž je používán, je možné využít jeho příkazovou řádku (command line) pro systémové skriptování, nebo – vzhledem k jednoduché syntaxi – také není problémem ho využít např. i jako kalkulačku.

Budeme-li se chtít dostat do režimu příkazové řádky, stačí v prostředí MS Windows spustit pomocí příkazu `cmd` spustit emulátor terminálu operačního systému a zde posléze zadat příkaz `python` (po instalaci Pythonu by měl být obsažen v systémové proměnné, takže není nutné zadávat celou cestu k interpreteru). Výstup režimu příkazové řádky Pythonu poznáme podle těchto znaků: „>>>“ (bez uvozovek). Poté již můžeme zadávat konkrétní příkazy.

Postup v prostředí GNU/Linuxu je analogický. V používaném emulátoru terminálu opět zadáme příkaz `python`, načež dostaneme obdobný výstup jako v předchozím případě. Veškeré ukázky a příklady uvedené v této práci jsou platformně nezávislé, není-li výslovně uvedeno jinak.

Pro účely této práce se také předpokládá, že výchozím interpreterem Pythonu v prostředí daného operačního systému je Python 3.1. Není-li tomu tak, je nutné nahradit příkaz `python` odpovídajícím příkazem pro spuštění tohoto interpreteru (v GNU/Linuxu je častým výchozím nastavením `python3`), jinak se postup nemění.

Alternativou k výše zmíněnému postupu (v MS Windows) je spuštění příkazové řádky přímo z nabídky start menu ve Windows, kde je po instalaci vytvořena podnabídka s názvem příslušné nainstalované verze Pythonu (např. Python 3.1), v níž je položka pro její spuštění obsažena.

Výstup tedy může vypadat například takto:

```
Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Ukázka 32: Výstup interpreteru na emulátoru terminálu (Python)

Z výstupu uvedeného v ukázce č. 32 můžeme vyčíst mj. informace o verzi Pythonu, i počítači, na kterém je interpreter spuštěn. Chceme-li opustit režim příkazové řádky, můžeme tak učinit dvěma způsoby – klávesovou zkratkou CTRL + Z, nebo pomocí funkce `exit`, kterou zadáváme ve tvaru se závorkami – `exit()`.

4.4.3 Konsolové programy

Program v Pythonu, který máme uložený v nějakém souboru, má typicky příponu „.py“. Jedná se o konvenci, přípony mohou nabývat i jiných podob. Zároveň je přípona „.py“ znakem napovídajícím, že se pravděpodobně jedná o program s konzolovým vstupem a výstupem (resp. o program interagující se standardním vstupem a výstupem operačního systému – může se jednat i o webovou aplikaci).

V prostřední operačního systému může být soubor s takovouto příponou asociován s interpreterem. Po poklepání na příslušný soubor pak dojde automaticky k jeho vyhodnocení (spuštění) přiřazeným interpreterem Pythonu. V případě interpreteru `python.exe` (v prostředí MS Windows) je pak vždy otevřeno okno emulátoru terminálu. Program můžeme spouštět i přímo – zadáním příkazu `python`, který je následován absolutní či relativní adresou k souboru se zdrojovým kódem.

Program v Pythonu může mít tuto vnitřní strukturu:

```
#!/usr/bin/env python3          # řetězec shebang
```

```
print('Ahoj světe')           # Volání konkrétní funkce
input('Stiskněte klávesu')    # Čekání na stisk klávesy
```

Ukázka 33: Základní struktura programu (Python)

Řetězec shebang je volitelný. Udává, jakým interpreterem Pythonu má být program spuštěn. Pohybujeme-li se v prostředí unixového systému, je nutné aby měl soubor se zdrojovým kódem nastavený příznak spustitelnosti (pomocí příkazu `chmod` s příslušným parametrem). Výše uvedený program může být spuštěn několika možnými způsoby. Hlavní jsou uvedeny v další ukázce. Pro jeho účely předpokládám, že soubor programu se zdrojovým kódem z ukázky č. 33 (s názvem `priklad.py`) je obsažen v aktuálním adresáři.

```
priklad.py    # První varianta funkční na MS Windows
./priklad.py  # První varianta funkční na GNU/Linuxu
python priklad.py    # Druhá varianta
```

Ukázka 34: Spouštění programu (Python)

Ve všech případech vypíše spuštěný program text „Ahoj světe“ a poté čeká na stisk klávesy. V případě první varianty funkční na MS Windows na tvaru shebang řetězce (41) nezáleží (asociace s interpreterem je dána systémovými registry). Ve stejné variantě (avšak v prostředí unixového systému) je program spuštěn pomocí interpreteru uvedeném v shebang řetězci. Ve druhé variantě už se opět na shebang řetězec nehledí – konkrétní interpreter je přímo zadán a program je jím vyhodnocen.

4.4.4 Programy s GUI

Programy, které mají grafické uživatelské rozhraní (GUI), mají v Pythonu příponu „py“. Opět mluvíme pouze o konvenci – nic nebrání tomu, aby se jednalo o příponu jinou, např. už zmiňovanou „py“. Sluší se však podotknout, že zatímco soubory s příponou „py“ jsou v prostředí MS Windows ve výchozím stavu spouštěny interpreterem `python.exe`, soubory s příponou „pyw“ interpreterem `pythonw.exe` (42). Spustíme-li takový program poklepaním na jeho ikonu, pak u programu s příponou „pyw“ není zobrazeno okno emulátoru terminálu (jak je tomu v případě programů s příponou „py“), ale přímo okno s grafickým rozhraním spouštěného programu.

Tato fakta jsou relevantní v okamžiku, kdy pro vykreslování GUI používáme balík `tkinter`, který je obsažen ve výchozí instalaci Pythonu. Jeho možnosti nejsou (zvláště vůči jiným existujícím alternativám) zvlášť velké, ale i tak dokáže pokrýt širokou škálu programátorových potřeb. Podrobnější informace o modulech, které poskytují prostředky pro tvorbu GUI, jsou rozepsány v části 5.2.2.

4.4.5 Webové aplikace

Python je rovněž možno použít pro tvorbu webových aplikací. V kapitole 6, která se věnuje příkladům, je jedna webová aplikace uvedena. V takovémto případě je řetězec shebang povinný. Python může být použit ve spojení s různými webovými servery na různých platformách, se kterými může komunikovat pomocí několika rozhraní (CGI, FastCGI apod.). V příkladu obsaženém v této práci se omezují na webový server Apache a rozhraní CGI.

4.5 Zhodnocení Pythonu

4.5.1 Pracovní cíle průzkumu

V rámci průzkumu týkajícího zhodnocení Pythonu pro oblast výuky algoritmizace a programování jsem si primárně na základě jak samotného tematického zaměření práce, tak osobní zkušenosti s tímto programovacím jazykem a nástrojů kolem něj, stanovil hypotézu, že Python je pro tento účel vhodný a obsahuje (či je pro něj dostupno) dostatečné množství nástrojů, díky kterým je schopen tomuto účelu dostát.

4.5.2 Metodika

Pro ověření či vyvrácení výše stanovené hypotézy jsem zvolil metodu nestrukturovaného rozhovoru se dvěma dlouholetými školskými odborníky, kteří mají v oblasti výuky algoritmizace a programování dlouhodobé zkušenosti. Vzhledem k tomu by se mělo jednat o dostatečně validní nástroj, díky kterému budou získané údaje mít potřebnou reliabilitu – jde tedy o expertní názory.

Konkrétně se jedná, v abecedním pořadí, o Ing. Jiřího Čepeláka, CSc., učitele s dlouholetou zkušeností s výukou jak na středních školách (SOŠ a SOU Nymburk,

V Kolonii 1804 a Gymnázium Nymburk, Komenského 779), tak na vysokých školách (FEL ČVUT, Pedagogická fakulta UK v Praze). Ing. Jiří Čepelák, CSc. je autorem učebnice zabývající se aplikací programovacího jazyka C na středních školách - Řešené příklady v jazyku C (43).

Dále pak o Ing. Miroslava Škopa, rovněž středoškolského učitele (Střední průmyslová škola elektrotechnická, V Úžlabině 320, Praha 10) i vysokoškolského učitele (Pedagogická fakulta UK v Praze). Ing. Škop se výuce programování v Pythonu dlouhodobě věnuje. Je autorem seriálu Python krok za krokem (44), který je určen pro zájemce o programování v Pythonu.

Protože průzkum probíhal metodou nestrukturovaného rozhovoru o spíše větším rozsahu (řádově desítky minut), jsou získané odpovědi utříděny a zformulovány do podoby jednotlivých tematicky ucelených bodů, vždy zaměřených na specifickou problematiku.

4.5.3 Souhrn získaných dat

V následující části jsou shrnuty odpovědi získané od Ing. Jiřího Čepeláka, CSc.:

- Ve srovnání s jinými jazyky užívanými ve školství – jako je např. jazyk C nebo Pascal – Python neobsahuje žádné explicitní deklarace proměnných, neboť se jedná o dynamicky typovaný jazyk. Ve výsledku však taková koncepce nevede k systematickosti.
- Python tlačí programátora k užití většího množství dalšího softwaru (knihoven pro grafiku apod.), pokud chce docílit širší funkcionality.
- Výchozí distribuce Pythonu obsahuje nepřiliš dobrý editor IDLE, individuálně lepší volbou je např. český volně šiřitelný editor PSpad, který syntaxi Pythonu rovněž podporuje.
- Python má vůči jiným jazykům netypickou syntaxi, byť silně inspirovanou jazykem C. Vytváření bloků na základě odsazení a absence středníků může sťažovat ladění vytvářeného programu. Na druhou stranu může být tato vlastnost i výhodou, neboť středník je prvkem, na který studenti často zapomínají.

- Práce s různými balíky – např. `tkinter` pro vykreslování grafického uživatelského rozhraní – je díky nestejnému způsobu zápisu nekonzistentní. Tato oblast je důležitá, protože výstupy programů, které často slouží pro výpočty, je velmi vhodné a názorné prezentovat právě grafickou formou.
- Celkově však Python dnes je pro účel výuky algoritmizace a programování perspektivním a vhodným řešením – na rozdíl od nedávné minulosti. Pro profesionální sféru nicméně zůstává vhodnějším řešením jazyk C.

V této části jsou shrnuty odpovědi autorizované Ing. Miroslavem Škopem:

- Mezi výhody Pythonu patří především fakt, že je zadarmo a neustále probíhá jeho vývoj (nejedná se „mrtvý“ jazyk).
- Podporuje práci s komplexními čísly a při výpočtech dosahuje vysoké přesnosti. V neposlední řadě je multiplatformní.
- Zápis kódu je velmi úsporný, práci usnadňuje také implicitní deklarace typů proměnných. Na rozdíl od některých jiných jazyků rozlišuje velká a malá písmena (u názvů proměnných apod.).
- Je pro něj dostupné velké množství knihoven (balíků a modulů), byť některé jsou placené. Je např. dostupný i modul pro socketovou komunikaci.
- Za nevýhodu lze považovat relativně nízký výkon (jedná se o interpretovaný jazyk), nicméně je možné jednotlivé moduly kompilovat do nativního kódu.
- Balík `tkinter` pro práci s grafikou je limitující.
- Dále také nemá klasickou konstrukci pole, jak je známá např. z jazyků Pascal nebo C. Je jí ale možno doplnit pomocí modulu NumPy.
- Kvůli absenci příkazu pro skok, také není obsažen příkaz `CASE` či jemu analogický.
- Systém odsazování může zkomplikovat situaci, ve které do sebe vnořujeme více bloků kódu.

- Vytvořit zápis programu z vývojového diagramu je komplikovaná záležitost, která mnohdy nevede k cíli.
- Pravděpodobně není vhodné začínat s Pythonem jako prvním jazykem pro výuku.

4.5.4 Interpretace

Při srovnávání obou sad odpovědí se pokusím nalézt průniky, tak odlišnosti.

Předně je zajímavé, že se v obou dvou sadách odpovědí objevuje negativní náhled na balík `tkinter`, byť ne z identických důvodů. Společná je rovněž určitá obava z některých syntaktických prvků Pythonu, zvláště na bloky tvořené pomocí odsazování. První názor koresponduje s mou osobní praxí. Mohu konstatovat, že syntaktické konstrukce modulů z balíku `tkinter` jsou silně poplatné grafickému toolkitu Tk, k němuž je balík `tkinter` rozhraním. Osobně se však domnívám, že na druhou stranu je tato syntaxe také stručná a lehce pochopitelná.

Co se týká vhodnosti Pythonu pro účel výuky algoritmizace a programování – žádná z odpovědí nebyla s touto hypotézou v přímém rozporu, odpovědi se lišili pouze v otázce míry. Zatímco Ing. Čepelák považuje Python vhodný pro výuku programování obecně, Ing. Škop tento rozsah omezuje na výuku programování v případech, kdy student již nějakou praxi s programováním má.

Rozpor nepanuje ani v otázce dostupnosti dostatečného množství nástrojů. Názor Ing. Škopa o velkém množství dostupných modulů se plně shoduje s mým. I na základě názoru Ing. Čepeláka se ale můžeme ptát, nakolik je (a kdy) žádoucí a nutné použít software třetích stran.

5 Vývojová prostředí a platformy pro vývoj v Pythonu

5.1 Vývojová prostředí a editory

Textové editory pro Python můžeme rozdělit na dva druhy. Na ty, které obsahují alespoň základní podporu pro Python (zvýraznění syntaxe apod.) a další, obecně zaměřené editory, které sice neobsahují žádnou specifickou podporu pro Python, ale

mohou být použity, protože prakticky jedinou podmínkou pro editaci zdrojového kódu v Pythonu je podpora Unicode (konkrétně UTF-8).

Pro Python existuje také několik dostupných vývojových prostředí (IDE). Vývojovým prostředím je zde myšlena taková aplikace, která mimo umožnění základní editace zdrojového kódu, poskytuje ještě další nástroje pro vývoj (debugger apod.). Některá z těchto vývojových prostředí byla napsána výhradně pro Python, jiná jeho podporu přidávají prostřednictvím zásuvného modulu (pluginu) nebo jiného doplňku.

5.1.1 IDLE

Jedná se o vývojové prostředí (45), které je distribuované spolu s Pythonem od verze 2.3 (v rámci instalátoru s implementací CPython). Název je zkratkou slov Integrated DeveLopment Enviroment. Je napsán přímo v Pythonu. Jeho grafické uživatelské rozhraní pak s pomocí knihovny Tk (práce s ní je v Pythonu možná pomocí modulů balíku `tkinter`).

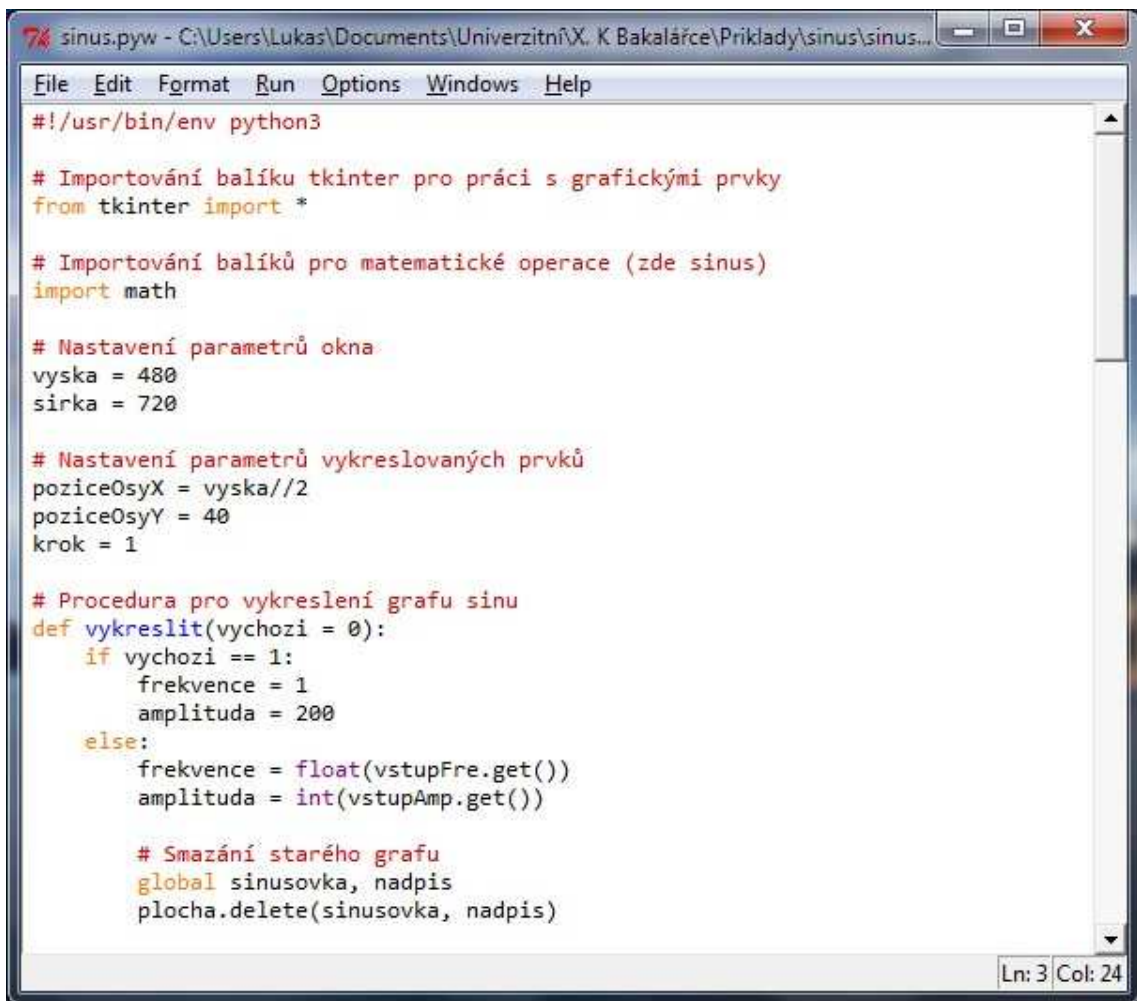
Ačkoliv se jedná o relativně jednoduchý nástroj, umožňuje zvýraznění syntaxe, obsahuje debugger, zvládá doplňování kódu, obsahuje škálu nástrojů pro práci s bloky kódu (vyhledávání, přejmenování, hromadné zakomentování a odkomentování) i mnoho dalších. Doplňování kódu je dostupné pomocí klávesové zkratky Control + mezerník.

IDLE je dostupný na všech platformách, na kterých je dostupný CPython. V některých distribucích Linuxu může distribuován jako samostatný balík odděleně od samotného interpreteru Pythonu.

Mezi jeho výhody patří jednoduchost a snadná pochopitelnost, stejně jako relativní hardwarová nenáročnost (náročnost na paměť je řádově v desítkách MB).

IDLE je (stejně jako CPython) dostupný pod Python Software Foundation License (PSFL). Ta je řazena mezi Open source licence. V IDLE je proto možný nekomerční i komerční vývoj software. IDLE je možno volně distribuovat.

Veškeré ukázky a příklady v této práci byly realizovány v tomto prostředí.



```
sinus.pyw - C:\Users\Lukas\Documents\Univerzitní\X. K Bakalářce\Priklady\sinus\sinus...
File Edit Format Run Options Windows Help
#!/usr/bin/env python3

# Importování balíku tkinter pro práci s grafickými prvky
from tkinter import *

# Importování balíků pro matematické operace (zde sinus)
import math

# Nastavení parametrů okna
vyska = 480
sirka = 720

# Nastavení parametrů vykreslovaných prvků
pozice0syX = vyska//2
pozice0syY = 40
krok = 1

# Procedura pro vykreslení grafu sinu
def vykreslit(vychozi = 0):
    if vychozi == 1:
        frekvence = 1
        amplituda = 200
    else:
        frekvence = float(vstupFre.get())
        amplituda = int(vstupAmp.get())

    # Smazání starého grafu
    global sinusovka, nadpis
    plocha.delete(sinusovka, nadpis)
```

Obrázek 1: Integrované vývojové prostředí IDLE

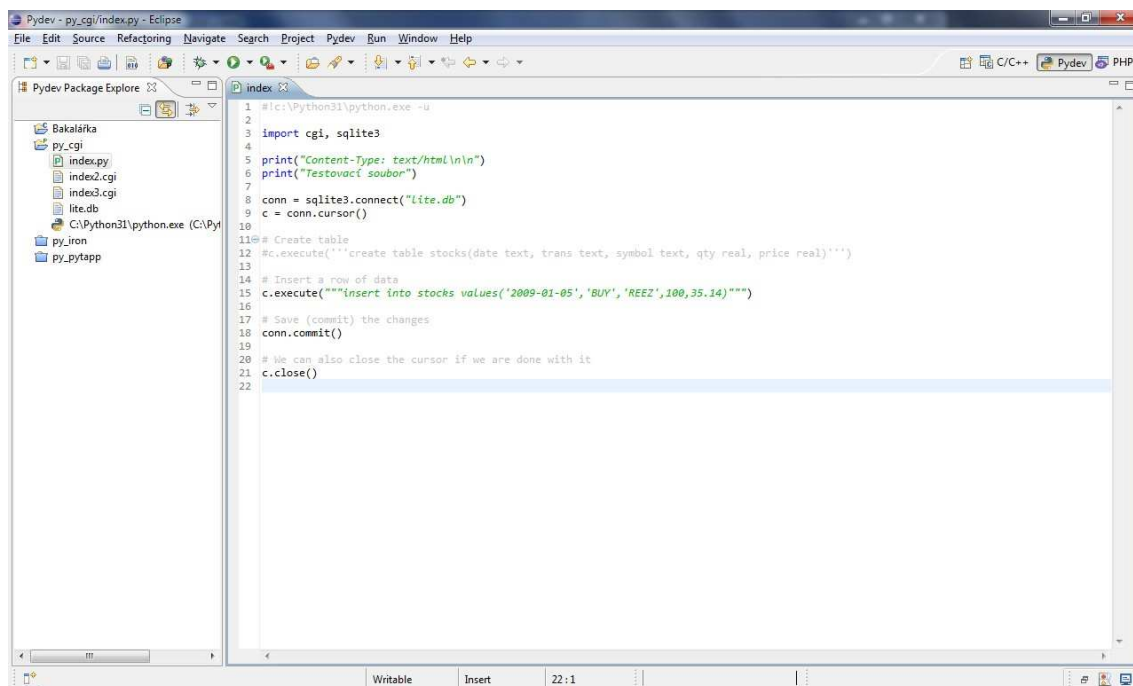
5.1.2 Eclipse (Pydev plugin)

Pro vývojové prostředí Eclipse je dostupný plugin Pydev (46). Jedná se o komplexní prostředí pro vývoj s mnoha pokročilými vlastnostmi (např. refactoring, spolupráce s produkty a moduly třetích stran – Google App Engine, Django).

Protože Eclipse je založené na rozšiřitelnosti pomocí pluginů, je dosažitelné spektrum funkcionalit velice široké (možno doplnit např. o verzovací systémy jako CVS apod.). Pydev podporuje kromě původní a nejrozšířenější implementace Pythonu (CPython) i Jython a IronPython. Eclipse je oficiálně distribuované pro (MS Windows, GNU/Linux i Mac OS X).

Eclipse je napsané v programovacím jazyku Java, grafické uživatelské rozhraní je

vytvořeno pomocí knihovny SWT. Hardwarová náročnost je podstatně vyšší než v případě IDLE, závisí jak na použité verzi Eclipse, tak na množství nainstalovaných pluginů. Obecně lze říci, že zatímco v případě IDLE je paměťová náročnost v řádech desítek MB, u Eclipse se již jedná o stovky MB.



Obrázek 2: Integrované vývojové prostředí Eclipse s pluginem Pydev

Eclipse i Pydev jsou dostupné pod Eclipse Public Licence (EPL), která opět patří mezi Open source licence. Je proto opět možné je jak dále šířit, tak využít pro nekomerční i komerční vývoj software.

5.1.3 The Eric Python IDE

Opět se jedná o komplexní vývojové prostředí. Je určeno pro Python a Ruby (47). Uživatelské rozhraní je postavené na knihovně Qt (resp. PyQt). Opět podporuje širokou škálu možností a pokročilých vlastností.

Jedná se o multiplatformní software, který je dostupný pro MS Windows, GNU/Linux, unixové systémy včetně Mac OS X. Jeho instalace v prostředí MS Windows se však rozhodně řadí mezi ty netriviální. Díky závislosti hned na čtyřech produktech třetích stran (vyjma Pythonu – v implementaci CPython – jako takového,

ještě na Qt knihovnách, PyQt – sloužící k napojení právě na Qt knihovny a nástroje QScintilla), z nichž některé je při použití jejich bezplatné verze nutné ručně zkompileovat (QScintilla). Na distribucích Linuxu, při použití balíčkovacího systému, tento problém nevzniká.

Samotné vývojové prostředí je právně ošetřené podmínkami GNU General Public License (GNU GPL) ve verzi 3, jež je opět Open source licenci. Prostedí je možné volně šířit a jeho užití je možné pro nekomerční i komerční vývoj.

5.1.4 VIM

Jedná se o textový editor s podporou zvýraznění syntaxe Pythonu. Je dostupný na celé škále operačních systémů a platform, které zahrnují MS Windows, GNU/Linux, unixové systémy včetně Mac OS X. Licence definuje VIM jako tzv. „charityware“. Protože je tato licence kompatibilní s GNU GPL, je možné VIM volně šířit i v něm psát aplikace s nekomerčním i komerčním charakterem.

5.1.5 PSPad

Další textový editor, který podporuje zvýraznění syntaxe Pythonu. Je dostupný pouze pro MS Windows. Obsahuje např. i podporu pro správu souborů vytvářených projektů. Nejedná se o Open source, nicméně jeho licence spadá do kategorie freeware. Je možné ho volně šířit i v něm psát komerční i nekomerční software.

5.1.6 Ostatní (Visual Studio, Netbeans)

Mimo zmíněná vývojová prostředí a textové editory ještě existuje mnoho takových, která umožňují vývoj v Pythonu. Lze zmínit např. IronPython tools přidávající podporu Pythonu (IronPythonu) do Visual Studia od společnosti Microsoft. Dále také IronPython studio, taktéž pro IronPython, které je postavené nad tzv. Visual Studio Shellem, díky čemuž je výsledný (minimálně grafický efekt) podobný Visual Studiu.

Python je pomocí pluginu podporován rovněž v populárním vývojovém prostředí Netbeans (48). Podporovány jsou zde CPython, IronPython i Jython, který je předinstalován. Python je podporován rovněž v textovém editoru Emacs.

5.2 Nástroje a moduly

Pro Python existuje velké množství modulů, které umožňují rozšíření jeho funkcionality o některé žádané vlastnosti. Mnoho zajímavých modulů je rovněž zahrnuto ve výchozí instalaci Pythonu. V následující části jsou zmíněny jak tyto moduly, tak moduly třetích stran. Uváděný výčet je jednak ilustrativní, co se možností použití Pythonu týká, tak popisný – vzhledem k použití některých těchto modulů v dalších částech práce, zejména u příkladů v kapitole 6.

5.2.1 Pro matematické operace

Zde lze vyzdvihnout zejména balík `numpy`. Tento balík je dílem třetí strany a není zahrnut ve standardní distribuci Pythonu. Jeho tvůrci ho popisují takto:

„NumPy je zásadním balíčkem potřebným pro realizaci vědeckých výpočtů v Pythonu. Mimo jiné obsahuje:

- objekt n-rozměrného pole
- pokročilé (vysílací) funkce
- nástroje pro integraci kódu napsaného v C/C++ a Fortranu
- nástroje pro lineární algebru, Fourierovu transformaci a práci s náhodnými čísly.“ (49)

Z našeho hlediska je tento balík důležitý zejména proto, že přináší klasickou konstrukci pole, kterou můžeme znát z jiných programovacích jazyků. Ve výsledku je možné vytvořit takovouto konstrukci:

```
from numpy import *
# Vytvoření neinicializovaného pole
pole1 = empty((2,2), int)
# Vytvoření inicializovaného pole
pole2 = arange(12).reshape(2,3,2)
# Vytvoření pole z n-tice
ntice = ("Město", "Moře", "Kuře", "Stavení")
pole3 = array(ntice).reshape(2,2)
```

```
# Zobrazení vytvořených polí
print(pole1)
print(pole2)
print(pole3)
```

Ukázka 35: Různé způsoby tvorby polí pomocí NumPy (Python)

V ukázce č. 35 je `pole1` neinicializovaným dvourozměrným polem, které používá celočíselný datový typ. Může obsahovat 4 hodnoty (dle hodnot v závorkách). Pole v druhém případě (`pole2`) je vytvořeno podobně, ale je trojrozměrné a naplněno posloupností čísel 0 až 11 (dle hodnoty v závorkách funkce `arrange`). Počet těchto čísel musí být shodný s počtem prvků pole (definovaných parametry funkce `reshape`). Poslední příklad pole demonstruje způsob, díky kterému můžeme vytvořit pole z obyčejné n-tice. Ohledně počtu prvků platí totéž, co v předchozím případě.

NumPy je licencován pomocí BSD licence, která ho řadí mezi Open source software. Je možné ho volně redistribuovat nebo upravovat. Je dostupný pro mnoho platform. Oficiálně je v binární formě poskytován pro MS Windows (32bitovou verzi), Mac OS X a Ubuntu Linux. Neoficiálně je však dostupný i pro ostatní linuxové distribuce, různá vydání BSD, popř. i pro 64bitové verze Windows (tato byla mj. použita pro realizaci ukázek v této práci).

5.2.2 Pro tvorbu grafického rozhraní

Pro vytváření grafického uživatelského rozhraní (GUI), popř. pro vykreslování grafických prvků obecně, existuje mnoho balíků a modulů. Mezi nejdůležitější z nich patří `tkinter` (50), který je dostupný přímo v oficiální distribuci Pythonu. Dostupnost i licencování se kryje s oficiální distribucí Pythonu. V jeho popisu na oficiálních stránkách Pythonu je udáno, že se pravděpodobně jedná i o balík nejpoužívanější.

Elementární program napsaný za pomoci balíku `tkinter` může být zapsán např. takovýmto způsobem:

```
from tkinter import *      # Načtení prvků z balíku tkinter
root = Tk()                # Vytvoření hlavního okna programu
```

```

root.title("Okno")          # Název vytvořeného okna
# Vytvoření tlačítka, určení jeho funkce v konstruktoru
obsah = Button(root, command=root.destroy)
# Další možný způsob určení vlastnosti objektu
obsah["text"] = "Kliknutím zde toto okno uzavřete"
obsah.pack()              # Zobrazení tlačítka
root.mainloop()          # Hlavní smyčka programu

```

Ukázka 36: Jednoduchý grafický program (Python)

Program z ukázky č. 36 zobrazí okno, které je zcela vyplněno jediným tlačítkem (nejsou udány žádné rozměry). Po kliknutí na něj se program ukončí, protože je zavolána metoda `destroy` objektu `root`, čímž dojde k ukončení běhu programu (je ukončena hlavní smyčka, díky které je program zobrazen, a čeká se na interakci uživatele). Program rovněž ukazuje dva způsoby zápisu, kterými můžeme definovat vlastnosti grafických objektů (widgetů).

Jde-li nám o dosažení širší funkcionality, můžeme sáhnout po některém z komplexnějších balíčků, jako jsou např. PyQt nebo PyGTK. V prvním případě se jedná o balík obstarávající napojení na Qt framework – můžeme tak tvořit grafické aplikace se složitým rozhraním, např. i za využití OpenGL (51). V druhém případě pak jde o napojení na grafický toolkit GTK, který je standardem linuxového prostředí GNOME (52), ale je rovněž multiplatformní.

5.2.3 Práce s databázemi

Projekt MySQL-python vyvíjí stejnojmenný balíček, který tvoří rozhraní pro napojení Pythonu na populární relační databázi MySQL (53). Jedná se o Open source projekt. Protože Python obsahuje standardizované rozhraní pro práci s databázemi (Python database API 2.0), je jen logické, že se ho tvůrci balíku MySQL-python drží. Toto rozhraní je v zatím poslední verzi (1.2.3) dostupné pouze pro Python řady 2 (konkrétně verze 2.3 až 2.7), ale s vydáním pro řadu 3 se do budoucna počítá také.

SQLite (54) se opět řadí mezi relační databáze. Její výhodou je – stejně jako v případě balíku `tkinter` – fakt, že je obsažena přímo ve standardní instalaci Pythonu. Není tedy třeba instalovat žádný další speciální software. Python obsahuje

balík tvořící rozhraní pro práci s ním (`sqlite3`), který rovněž dodržuje Python database API 2.0.

I když se jedná o relativně jednoduchou databázi (co se schopností a implementovaných prvků týká), pro potřeby výuky více než dostačuje. V prostředí MS Windows lze v souvislosti s ním doporučit program SQLite Database Browser, ve kterém lze vytvářet, prohlížet i upravovat soubory, které tvoří vlastní data této databáze. Databáze i zmíněný program jsou Open source.

5.2.4 Webové nástroje

Python je relativně často používán na webu jako skriptovací jazyk na straně serveru. Podporuje několik existujících rozhraní, které to umožňují. Základním je CGI, které je sice postarším, ale ze strany webových serverů (Apache, lighttpd) podporovaným řešením. Způsob práce s ním není zvláště odlišný od psaní skriptů v PHP. V kapitole 6 je rozveden příklad se skriptem v Pythonu, který využívá právě CGI (a stejnojmenný modul), a který řeší vyhodnocení HTML formuláře (55).

Python samozřejmě obsahuje mnoho dalších rozhraní, která jsou pravděpodobně využívána v reálném prostředí mnohem častěji (např. WSGI), protože jsou na nich postavené populární webové frameworky, z nichž můžeme jmenovat např. Django. Zde však lze říct totéž, co již padlo dříve – pro potřeby výuky bohatě vystačíme s CGI. Bohužel v úvodu zmíněné RVP se vyjadřují k programování na webu poměrně málo (pokud vůbec).

5.2.5 Ostatní

Množství balíků a nástrojů, které jsou dostupné pro Python, jsou tak široké, že jejich popis by svým rozsahem vydal na další práci. Omezím se proto na výčet jen několika ze standardní knihovny Pythonu (některé z nich jsou použity v programech v příloze na CD, které jsou rozebrány v kapitole 6):

- `math` – tento modul zpřístupňuje matematické funkce (např. goniometrické), které pracují s čísly s desetinou čárkou. Protože v jeho pozadí je knihovna, jež je napsaná v jazyce C, lze očekávat, že se bude jednat o (z hlediska výkonu)

zajímavé řešení i u náročnějších úloh.

- `os` – modul, který zprostředkovává interakci s operačním systémem. Umožňuje např. pokročilou práci s adresáři a soubory, ale i procesy běžící v operačním systému.
- `sys` – modul, který umožňuje např. předávat při spouštění skriptu z konzole různé parametry.
- `string` – modul pro práci s řetězci.
- `ftplib` – modul, který nám umožňuje pracovat s klientskou částí FTP protokolu.

Vyčerpávající výčet balíků a modulů standardní knihovny je obsažen v oficiální dokumentaci Pythonu (56). Pokud by nás zajímaly balíky třetích stran, jejich obsáhlý seznam nalezneme na dále na webu Pythonu (57).

5.3 Python v praxi

V návaznosti na předchozí spíše teoretický úvod k možnostem, kterými Python disponuje, je zajímavé přiblížit, v jakých programech, resp. jakými firmami je Python používán v praxi. Z těch nejznámějších se jedná např. o Google, který Python využívá pro svůj Google App Engine. V českém prostředí je vhodné zmínit portál seznam.cz (58), kde je Python (spolu s C a C++) jednou z hlavních technologií.

Budeme-li se bavit přímo o aplikacích napsaných v Pythonu, můžeme zmínit např. populární webovou službu Dropbox, původní implementaci BitTorrentu, nebo nelineární video editor OpenShot (59). Rozsáhlý výpis těchto aplikací nalezneme i přímo na oficiálních stránkách Pythonu (60).

6 Příklady

6.1 Jednoduchý FTP klient

Cílem programu je demonstrovat základy objektového programování v Pythonu, práci se soubory (otevření a uložení dat textového souboru, jejich následné zpracování),

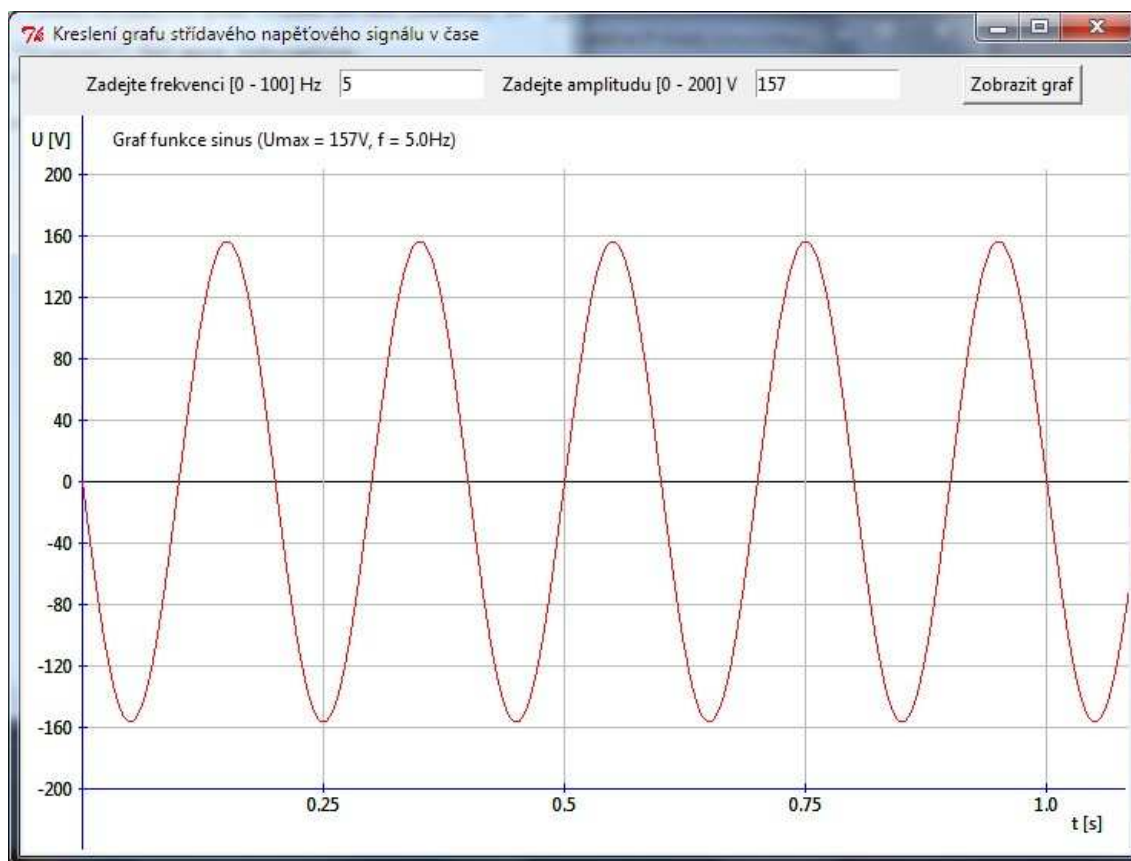
vstup a výstup z konzole (včetně ošetření pomocí odchyťávání výjimek a přetypování zadaných proměnných), práci s moduly (vytvoření vlastního modulu) a obecně bohatost základní knihovny Pythonu (jako prostředek byl zvolen modul `ftplib`).

Cílem programu není vytvořit plně funkčního FTP klienta, jeho funkcionalita je podřízena demonstrativním účelům celého programu. Tomu odpovídá i struktura programu a volba použitých prostředků. Program samotný je pak obsažen v příloze na CD v adresáři `ftp`. Spouští se pomocí souboru `run.py`. Protože se jedná o výhradně konzolový program, je obsaženo pouze jednoduché textové rozhraní.

6.2 Kreslení grafu střídavého signálu

Smyslem programu je nastínit schopnosti balíku `tkinter` pro vykreslování grafického uživatelského rozhraní. Program ukazuje, jakým způsobem vykreslit okno programu, následně do něj zasadit prvky umožňující kreslení grafických primitiv (přímek, křivek apod.), a jak vytvořit informační a ovládací prvky (popisky, vstupní pole a tlačítka). Je předvedeno, jak se provazuje stisknutí tlačítka s konkrétní obslužnou procedurou.

Program se spouští pomocí souboru `sinus.pyw`. Jeho rozhraní je výhradně grafické. Logika některých částí tohoto programu (zejména těch, které řeší vykreslení grafu funkce sinus) byla převzata z diskuzního serveru DaniWeb (61) a upravena pro použití v prostředí Pythonu 3. Program se nachází v příloze na CD v adresáři `sinus`.



Obrázek 3: Program pro kreslení grafu střídavého napětového signálu

6.3 Webová aplikace s HTML formulářem

Tento program generuje jednoduchou webovou stránku (v HTML5), která obsahuje webový formulář. Program tento formulář vyhodnocuje (zobrazuje zadaná data). Účelem programu je demonstrovat tvorbu jednoduché serverové webové aplikace, která využívá CGI, a ukázat jakým způsobem se může pracovat s proměnnými odeslanými formulářem (jejich vyhodnocení a ošetření).

Pro správný běh programu je vyžadován nainstalovaný webový server (testováno na 64bitové verzi serveru Apache 2.2 v prostředí MS Windows a 32bitové verzi téhož serveru v prostředí Debian GNU/Linuxu, nic by však nemělo bránit funkci u jiných správně nakonfigurovaných serverů, např. IIS).

V souboru nastavení serveru Apache (typicky httpd.conf) musí být povoleno zpracování CGI skriptů pro soubory s příponou „py“, nebo alespoň zpracování

souboru `.htaccess`, kterým můžeme řešit totéž.

Soubor musí být umístěn ve složce, která je určena pro zpracování obecně webových nebo přímo CGI skriptů. Řešíme-li konfiguraci pomocí souboru `.htaccess`, pak tento soubor musí být umístěn ve stejné složce, v níž je umístěn spouštěný skript (v našem případě soubor `index.py`). Jeho obsah musí být následující:

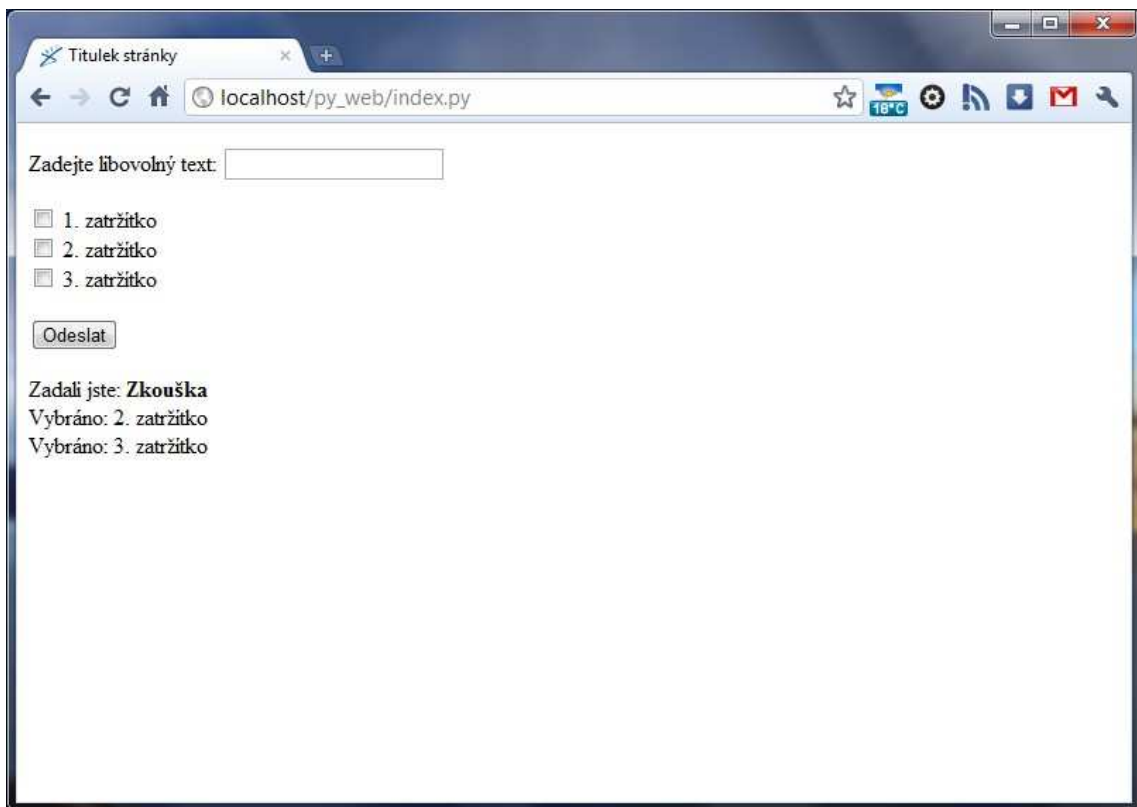
```
Options +ExecCGI
AddHandler cgi-script .py
```

Ukázka 37: Obsah souboru `.htaccess` pro konfiguraci Apache

Po spuštění serveru (v závislosti na jeho konfiguraci) zadáme do adresního řádku prohlížeče adresu, na které je náš skript umístěn (testováno na `http://localhost/py_web/index.py`, kde `py_web` je adresář umístěný v kořenovém adresáři, u Apache tzv. `DocumentRoot`, webového serveru).

V prostředí MS Windows lze doporučit použití programu WampServer (62), který v sobě Apache již obsahuje. Vyjma výše zmíněného není nutná prakticky žádná další konfigurace. V prostředí GNU/Linuxu je Apache s největší pravděpodobností obsažen vždy v repozitáři dané distribuce. Zde je nutné aby skript měl správně nastavená práva (a zejména příznak spustitelnosti), aby mohl být webovým serverem spuštěn.

Soubor skriptu v příloze má shebang řetězec v takovém tvaru, aby mohl být spuštěn v prostředí MS Windows. Na jiné platformě ho je nutné pozměnit tak, aby směřoval na správný interpreter Pythonu. Program se nachází v příloze na CD v adresáři `web`.



Obrázek 4: Webová aplikace s HTML formulářem

7 Závěr

V práci předvádím, že Python může být použit pro potřeby výuky algoritmizace a programování. Průzkum mezi učiteli, kteří se výuce programování věnují, s tímto tvrzením nebyl v rozporu. Na možnost využít Python pro tento účel poukazovali i autoři několika textů, které v práci cituji.

Formou praktických ukázek kódu jsem ukázal, že syntaxe Pythonu je vůči jiným používaným jazykům obvykle méně komplikovaná, k zápisu kódu s identickou funkcí je často třeba podstatně méně kódu. Toto je názorně předvedeno hlavně v kapitole 6, kde je popsáno několik již komplexnějších aplikací s relativně širokou funkcionalitou realizovanou na minimálním prostoru.

Python je jazykem systematicky navrženým s ohledem k již existujícím zvyklostem a řešením. Vzhledem k tomu, že jeho historie je už více než dvacetiletá, a k jeho nasazení ve firemním prostředí (jak ukazuji v části věnované aplikacím a firmám, které Python používají), je možné konstatovat, že je i jazykem široce zavedeným.

Python není platformě závislý a existuje pro něj široké množství vývojových prostředí, editorů a nástrojů, které mohou dále rozšiřovat jeho funkcionalitu. Také jsem ukázal, že licenční politika Pythonu umožňuje jeho nasazení a užití bez jakýchkoliv licenčních poplatků. I tak však existují implementace Pythonu, které jsou úžeji svázané s konkrétní technologií či operačním systémem. Potenciálnímu programátorovi v Pythonu však spíše umožní cílenější profilaci.

Python může být využit jak pro výuku procedurálního, tak objektového paradigmatu programování, aniž by bylo třeba jakkoliv měnit prostředky použité platformy (jazyk, vývojové prostředí či editor). Snažil jsem se v maximální míře využívat prostředků výchozí instalace jako takové, jejíž možnosti jsem předvedl v šíři zahrnující i program s grafickým uživatelským rozhraním.

Další zkušeností byla realizace průzkumu na středních školách, ze 24 oslovených škol reagovala šestina. Průzkum byl tedy orientační, v případě realizace dalšího průzkumu bych byl patrně nucen navýšit počet oslovených škol – je však otázkou, zda

by úsilí vynaložené na takové navýšení posléze bylo ospravedlněno získanými daty. S výsledky získanými v jiné části práce metodou rozhovoru jsem byl spokojen, proto by se toto i zde nabízelo jako další možnost. Pokud by takový rozhovor byl dostatečně jasně strukturovaný, mohla by získaná data být pro tento účel dostatečně reliabilní.

V úvodu práce jsem citoval z několika RVP, které oblasti výuky algoritmizace a programování na středních školách definují. Vymezení v nich uvedené je vesměs nedostatečně obecné (snad s výjimkou prvního případu), vzhledem k tomu, že ICT kompetence jsou dnes tak zásadní oblastí. Už proto, že české školství dlouhodobě nedokáže vyprodukovat dostatečný počet takto zaměřených specialistů, by tyto oblasti měly být vymezeny lépe a ve větší šíři. I kvůli naději, že se tato situace snad v budoucnu změní, je jejich rozsah několikrát v různé míře překročen.

Realizace této práce pro mě byla v mnoha ohledech poučným zážitkem, už proto, že byla první svého druhu, kterou jsem kdy vypracovával. V průběhu prací jsem získal neocenitelné zkušenosti, a to i ve tom směru, že bych některé věci nyní udělal jinak, vhodnější formou, nebo obecně lépe. V nezanedbatelné míře jsem si rovněž rozšířil vědomosti o popisovaném tématu jako takovém. Doufám, že čtení této práce bude příjemným a přínosným zážitkem, stejně jako byla práce na ní pro mě.

Souhrnně tedy uvádím, že Python je pro potřeby výuky algoritmizace a programování velmi rozumnou alternativou k ostatním používaným prostředím, a že zejména praktické ukázky v práci mohou být přínosné pro svoje okamžité praktické využití v prostředí reálných středních škol.

8 Seznam ukázek

1. Vytváření bloků zdrojového kódu (Python)
2. Vytváření bloků zdrojového kódu (Pascal)
3. Proměnná jako „ukazatel“ na objekt (Python)
4. Analogický zápis předchozí ukázky (Pascal)
5. Zápis vybraných aritmetických operátorů (Python)
6. Použití operátorů přiřazení a identity (Python)
7. Použití logického operátoru a operátorů pro porovnání (Python)
8. Práce s n-ticí a seznamem (Python)
9. Práce s jedno a dvourozměrným polem (Pascal)
10. Operace s n-ticemi a seznamy (Python)
11. Řezání n-tic a seznamů (Python)
12. Práce se slovníkem (Python)
13. Zápis podmínky (Python)
14. Použití ternárního operátoru (C)
15. Analogie k ternárního operátoru (Python)
16. Zápis podmínky bez 'ternárního' operátoru (Python)
17. Ošetření výjimky (Python)
18. Zápis cyklu typu while (Python)
19. Jednoduchý zápis cyklu typu for (Python)
20. Analogický zápis předchozí ukázky (C)
21. Různé zápisy for cyklu s použitím funkce range (Python)
22. Jednoduchá procedura a její volání (Python)
23. Analogický zápis předchozí ukázky (Pascal)
24. Funkce s parametry a její volání (Python)

25. Obory platností proměnných (Python)
26. Základy objektového modelu (Python)
27. Analogický zápis předchozí ukázky (C++)
28. Vytvoření odvozené třídy (Python)
29. Dva způsoby práce s moduly (Python)
30. Práce s vlastním modulem (Python)
31. Alternativní zápis importu modulu (Python)
32. Výstup interpreteru na emulátoru terminálu (Python)
33. Základní struktura programu (Python)
34. Spouštění programu (Python)
35. Různé způsoby tvorby polí pomocí NumPy (Python)
36. Jednoduchý grafický program (Python)
37. Obsah souboru .htaccess pro konfiguraci Apache

9 Seznam obrázků

1. Integrované vývojové prostředí IDLE
2. Integrované vývojové prostředí Eclipse s pluginem Pydev
3. Program pro kreslení grafu střídavého napěťového signálu
4. Webová aplikace s HTML formulářem

10 Seznam použitých informačních zdrojů

1. *Český statistický úřad* [online]. Český statistický úřad, c2011 [cit. 2011-04-01]. Předmět a účel klasifikace KKOV. Dostupné z WWW: <http://www.czso.cz/csu/klasifik.nsf/i/klasifikace_kmenovych_oboru_vzdelani_%28kkov%29>.
2. *Rámcový vzdělávací program pro obor vzdělání 18-20-M/01 Informační technologie*. [online]. Praha : Ministerstvo školství, mládeže a tělovýchovy, 2008. 79 s. [cit. 2011-04-01]. Dostupné z WWW: <<http://zpd.nuov.cz/RVP/ML/RVP%201820M01%20Informacni%20technologie.pdf>>.
3. *Rámcový vzdělávací program pro obor vzdělání 26-41-M/01 Elektrotechnika*. [online]. Praha : Ministerstvo školství, mládeže a tělovýchovy, 2007. 79 s. [cit. 2011-04-01]. Dostupné z WWW: <<http://zpd.nuov.cz/RVP/ML/RVP%202641M01%20Elektrotechnika.pdf>>.
4. *Rámcový vzdělávací program pro gymnázia*. [online]. Praha: Výzkumný ústav pedagogický, 2007. 100 s. [cit. 2011-04-01]. Dostupné z WWW: <http://www.vuppraha.cz/wp-content/uploads/2009/12/RVPG-2007-07_final.pdf>. ISBN 978-80-87000-11-3.
5. WAGNER, Jan. *Google profiles* [online]. c2011 [cit. 2011-04-01]. Janek Wagner - Google profile. Dostupné z WWW: <<https://profiles.google.com/janek.wagner/about>>.
6. PECINOVSKEÝ, Rudolf. *Pecinovsky.cz* [online]. c2001-2009 [cit. 2011-04-01]. Rudolf Pecinovský – domovská stránka. Dostupné z WWW: <<http://rudolf.pecinovsky.cz>>.
7. WAGNER, Jan. *Jankovy názory* [online]. 7.9.2005 [cit. 2011-04-01]. Jaký programovací jazyk do škol? Dostupný z WWW: <<http://jankovy.nazory.cz/2005-09.html#071642>>.
8. PECINOVSKEÝ, Rudolf. *Pecinovsky.cz* [online]. c2001-2009 [cit. 2011-04-01]. Starší publikace a seriály. Dostupné z WWW: <<http://publikace.pecinovsky.cz/>>.
9. PECINOVSKEÝ, Rudolf; VIRIUS, Miroslav. *Učebnice programování - základy algoritmizace* [online]. Praha : Grada Publishing, 1997. 177 s. [cit. 2011-04-01]. Dostupné z WWW: <http://publikace.pecinovsky.cz/Zaklady_algoritmizace.pdf>. ISBN 80-7169-577-7.
10. PECINOVSKEÝ, Rudolf. Výuka objektově orientovaného programování žáků základních a středních škol. In SNÁŠEL, Václav. *Objekty 2003*. Ostrava : VŠB-TU Ostrava, 2003. ISBN 80-248-0274-0.

11. PECINOVSKÝ, Rudolf. Proč a jak učit OOP žáky základních a středních škol. In *Zborník príspevkov*. Žilina : Žilinská univerzita, 2004. Dostupné z WWW: <http://vyuka.pecinovsky.cz/prispevky/Proc_a_jak_ucit_OOP_na_ZS_a_SS.pdf>. ISBN 80-8070-271-3.
12. TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 26.6.2007 [cit. 2011-04-01]. Logo – dětská hračka nebo programovací jazyk?. Dostupné z WWW: <<http://www.root.cz/clanky/logo-ndash-detska-hracka-nebo-programovaci-jazyk/>>.
13. TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 1.7.2010 [cit. 2011-04-01]. Programovací jazyky určené pro výuku programování. Dostupné z WWW: <<http://www.root.cz/clanky/programovaci-jazyky-urcene-pro-vyuku-programovani/>>.
14. *Free Pascal* [online]. Free Pascal team, c1993-2010 [cit. 2011-04-01]. Dostupné z WWW: <<http://freepascal.org/>>.
15. *Wikipedia, The Free Encyclopedia : C (programming language)* [online]. C2011, last modified on 30 March 2011 [cit. 2011-04-01]. Dostupný z WWW: <[http://en.wikipedia.org/wiki/C_\(language\)](http://en.wikipedia.org/wiki/C_(language))>
16. FALTÝNEK, Lukáš. *Linux EXPRESS* [online]. 18.10.2006 [cit. 2011-04-01]. Logo. Dostupné z WWW: <<http://www.linuxexpres.cz/praxe/logo>>.
17. TIŠNOVSKÝ, Pavel. *Root.cz* [online]. 10.7.2010 [cit. 2011-04-01]. Komerční implementace Loga. Dostupné z WWW: <<http://www.root.cz/clanky/komercni-implementace-loga/>>.
18. AB, KVI. *Imagine* [online]. c2002 [cit. 2011-04-01]. Popis. Dostupné z WWW: <<http://imagine.input.sk/popis.html>>.
19. VANÍČEK, Jiří. *Přednášky z didaktiky informatiky a výpočetní techniky* [online]. 2004 [cit. 2011-04-01]. Programování. Dostupné z WWW: <http://eamos.pf.jcu.cz/amos/kat_inf/externi/kat_inf_0548/7_programovani.pdf>.
20. VAN ROSSUM, Guido; WARSAW, Barry. *Python* [online]. Python Software Foundation, 05-Jul-2001, last-modified: 2011-02-17 [cit. 2011-04-02]. Style Guide for Python Code. Dostupné z WWW: <<http://www.python.org/dev/peps/pep-0008/>>.
21. *Python* [online]. Python Software Foundation, c1990-2011, last updated on Apr 02, 2011 [cit. 2011-04-02]. Built-in Constants. Dostupné z WWW:

<<http://docs.python.org/py3k/library/constants.html>>.

22. GOSLING, James, et al. *Java Language Specification* [online]. Third Edition. Santa Clara : Sun Microsystems, c1996-2005 [cit. 2011-04-02]. Objects. Dostupné z WWW: <http://java.sun.com/docs/books/jls/third_edition/html/typesValues.html#4.3.1>. ISBN 0-321-24678-0.

23. *Python* [online]. Python Software Foundation, c1990-2011, last updated on Apr 02, 2011 [cit. 2011-04-02]. Operator. Dostupné z WWW: <<http://docs.python.org/py3k/library/operator.html>>.

24. *Python* [online]. Python Software Foundation, c1990-2010, last updated on Nov 27, 2010 [cit. 2011-04-02]. Data Structures. Dostupné z WWW: <<http://docs.python.org/release/3.1.3/tutorial/datastructures.html>>.

25. SUMMERFIELD, Mark. *Python 3 : Výukový kurz*. Vyd. 1. Brno : Computer Press, 2010. Typy představující posloupnost, s. 110. ISBN 978-80-251-2737-7.

26. SUMMERFIELD, Mark. *Python 3 : Výukový kurz*. Vyd. 1. Brno : Computer Press, 2010. Slovníky, s. 128. ISBN 978-80-251-2737-7.

27. *Python* [online]. Python Software Foundation, c1990-2011, last updated on Apr 02, 2011 [cit. 2011-04-02]. Compound statements. Dostupné z WWW: <http://docs.python.org/py3k/reference/compound_stmts.html>.

28. BRANDL, Georg. *Python* [online]. Python Software Foundation, 04- May- 2005, last- modified: 2007-05-01 [cit. 2011-04-02]. Unifying try-except and try-finally. Dostupné z WWW: <<http://www.python.org/dev/peps/pep-0341/>>.

29. VAN ROSSUM, Guido. *The History of Python* [online]. January 20, 2009 [cit. 2011-04-02]. A Brief Timeline of Python. Dostupné z WWW: <<http://python-history.blogspot.com/2009/01/brief-timeline-of-python.html>>.

30. *Wikipedie: Otevřená encyklopedie : Guido van Rossum* [online]. c2010 [cit. 2011-04-02]. Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=Guido_van_Rossum&oldid=6114703>.

31. VAN ROSSUM, Guido. *The History of Python* [online]. January 13, 2009 [cit. 2011-04-02]. Introduction and Overview. Dostupné z WWW: <<http://python-history.blogspot.com/2009/01/introduction-and-overview.html>>.

32. VAN ROSSUM, Guido. *The History of Python* [online]. January 13, 2009 [cit. 2011-04-02]. Python's Design Philosophy. Dostupné z WWW: <<http://python-history.blogspot.com/2009/01/pythons-design-philosophy.html>>.
33. ZADKA, Moshe; VAN ROSSUM, Guido. *Python* [online]. Python Software Foundation, 11-Mar-2001, 2009-01-18 [cit. 2011-04-02]. Changing the Division Operator. Dostupné z WWW: <<http://www.python.org/dev/peps/pep-0238/>>.
34. *IronPython* [online]. Microsoft, 11/19/2009, last edited Dec 12 2009 [cit. 2011-04-02]. IronPython Performance Report. Dostupné z WWW: <<http://ironpython.codeplex.com/wikipage?title=IP26FinalVsCPy26Perf&referringTitle=IronPython%20Performance>>.
35. *Mono* [online]. Mono project [cit. 2011-04-02]. Download - Mono. Dostupné z WWW: <<http://www.go-mono.com/mono-downloads/download.html>>.
36. *Wikipedia, The Free Encyclopedia : IronPython* [online]. c2011, last modified on 24 March 2011 [cit. 2011-04-01]. Dostupný z WWW: <<http://en.wikipedia.org/wiki/IronPython>>.
37. *IronPython* [online]. Microsoft [cit. 2011-04-02]. Documentation. Dostupné z WWW: <<http://ironpython.net/documentation/>>.
38. *Python : JythonWiki* [online]. Jython project, 2005-02-07 [cit. 2011-04-02]. General Information. Dostupné z WWW: <<http://wiki.python.org/jython/JythonFaq/GeneralInfo>>.
39. *Java* [online]. Oracle [cit. 2011-04-02]. Java Downloads for All Operating Systems. Dostupné z WWW: <<http://java.com/en/download/manual.jsp>>.
40. *Wikipedia, The Free Encyclopedia : Jython* [online]. c2011, last modified on 2 April 2011 [cit. 2011-04-01]. Dostupný z WWW: <<http://en.wikipedia.org/wiki/Jython>>.
41. *Python* [online]. Python Software Foundation, c1990-2010, last updated on Nov 27, 2010 [cit. 2011-04-02]. Using Python on Unix platforms. Dostupné z WWW: <<http://docs.python.org/release/3.1.3/using/unix.html?highlight=shebang>>.
42. *Python* [online]. Python Software Foundation, c1990-2010, last updated on Apr 02, 2011 [cit. 2011-04-02]. Using Python on Windows. Dostupné z WWW: <<http://docs.python.org/py3k/using/windows.html?highlight=pythonw>>.
43. ČEPELÁK, Jiří. *Řešené příklady v jazyku C*. Praha : Computer Press, 2001. 156 s.

ISBN 80-7226-575-X.

44. *SPŠE, V Úžlabině 3, Praha 10* [online]. Střední průmyslová škola elektrotechnická, V Úžlabině 320 Praha 10, c2006 [cit. 2011-04-02]. Výukové materiály. Dostupné z WWW: <http://www.uzlabina.cz/projekty_vyukove-materialy.html>.

45. *Python* [online]. Python Software Foundation, c1990-2010, last updated on Nov 27, 2010 [cit. 2011-04-02]. IDLE. Dostupné z WWW: <<http://docs.python.org/release/3.1.3/library/idle> >.

46. *Pydev* [online]. Appcelerator, c2008-2011, last site update: 03 February 2011 [cit. 2011-04-02]. Dostupné z WWW: <<http://pydev.org/>>.

47. *The Eric Python IDE* [online]. [cit. 2011-04-02]. Dostupné z WWW: <<http://eric-ide.python-projects.org/>>.

48. *NetBeans* [online]. Oracle, 5 November 2009, last modified on 5 November 2009 [cit. 2011-04-02]. NetBeansPythonTutorial. Dostupné z WWW: <<http://wiki.netbeans.org/NetBeansPythonTutorial>>.

49. *NumPy* [online]. Numpy developers, c2009 [cit. 2011-04-02]. Dostupné z WWW: <<http://numpy.scipy.org/>>.

50. *Python* [online]. Python Software Foundation, 2002-07-12 [cit. 2011-04-02]. TkInter. Dostupné z WWW: <<http://wiki.python.org/moin/TkInter>>.

51. *Riverbank Computing* [online]. Riverbank Computing, c2010 [cit. 2011-04-02]. TkInter. Dostupné z WWW: <<http://www.riverbankcomputing.co.uk/software/pyqt/intro>>.

52. *PyGTK* [online]. The GNOME project, c2004-2010, last changed on Tue Oct 12 20:29:25 2004 [cit. 2011-04-02]. What is PyGTK?. Dostupné z WWW: <<http://faq.pygtk.org/index.py?req=show&file=faq01.001.htm>>.

53. DUSTMAN, Andy. *Python* [online]. Python Software Foundation, 2010-07-22 [cit. 2011-04-02]. MySQL-python 1.2.3. Dostupné z WWW: <<http://pypi.python.org/pypi/MySQL-python/>>.

54. HAERING, Gerhard. *Python* [online]. Python Software Foundation, c1990-2011 [cit. 2011-04-02]. Pysqlite 2.6.3. Dostupné z WWW: <<http://pypi.python.org/pypi/pysqlite/2.6.3>>.

55. *Python* [online]. Python Software Foundation, 2004-07-15 [cit. 2011-04-02]. CGI Scripts. Dostupné z WWW: <<http://wiki.python.org/moin/CgiScripts>>.
56. *Python* [online]. Python Software Foundation, c1990-2010, last updated on Nov 27, 2010 [cit. 2011-04-02]. The Python Standard Library. Dostupné z WWW: <<http://docs.python.org/release/3.1.3/library/index.html>>.
57. *Python* [online]. Python Software Foundation, c1990-2011 [cit. 2011-04-02]. PyPI. Dostupné z WWW: <<http://pypi.python.org/pypi>>.
58. *Novinky.cz* [online]. Borgis, 19. října 2005 [cit. 2011-04-02]. Seznam.cz koketuje s technologií PHP. Dostupné z WWW: <<http://www.novinky.cz/komercni-clanky/67691-seznam-cz-koketuje-s-technologiei-php.html>>.
59. *Wikipedia, The Free Encyclopedia : List of Python software* [online]. c2011, last modified on 29 March 2011 [cit. 2011-04-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/List_of_Python_software>.
60. *Python* [online]. Python Software Foundation, c2010 [cit. 2011-04-02]. Python Success Stories. Dostupné z WWW: <<http://www.python.org/about/success/>>.
61. *DaniWeb : IT Discussion Community* [online]. DaniWeb, Oct 25th, 2006 [cit. 2011-04-02]. Plotting a Line Curve (Python). Dostupné z WWW: <<http://www.daniweb.com/software-development/python/code/216811>>.
62. BOURDON, Romain. *WampServer* [online]. 2004 [cit. 2011-04-02]. Dostupné z WWW: <<http://www.wampserver.com/>>.

11 Seznam příloh

1. Zadání práce
2. CD se zdrojovými kódy programů a výsledky průzkumu