

Část F

Přílohy

V této části naleznete čtyři přílohy: první vám vysvětlí, jak lze ve *Windows* používat substituované disky, druhá vám představí syntaktické diagramy, třetí vás seznámí se základními konvencemi pro psaní programů v *Pythonu* a odkáže vás na dokumenty, v nichž je dané téma rozebráno podrobněji, a čtvrtá poskytne stručný přehled základních inovací v několika posledních subverzích jazyka.

Příloha A

Konfigurace ve Windows

A.1 Definice substituovaných disků

V operačním systému *Windows* můžete používat 26 logických disků – pro každé písmeno abecedy jeden. Většina uživatelů však používá pouze zlomek tohoto počtu. *Windows* umožňují použít volná písmena pro tzv. **substituované disky**, což jsou složky, které se rozhodnete vydávat za logický disk. Protože o této možnosti většina uživatelů neví, a přitom je to funkce velice užitečná, ukážu vám, jak ji můžete využít. Substituované disky se definují pomocí příkazu:

```
SUBST název_disku substituovaná_složka
```

Nejjednodušší způsob, jak definovat ve *Windows* např. substituovaný disk **R:**, je vložit do složky, kterou budete chtít substituovat jako disk **R:**, dávkový soubor s příkazem k substituci. (Písmeno **P** se pro *Python* hodí nejlépe, ale můžete si vybrat jakékoliv jiné, které je na vašem počítači volné.)

Pokud jste ještě nepracovali s dávkovými soubory, tak vězte, že to jsou obvyčejné textové soubory, do nichž zapisujete příkazy pro operační systém. Jejich název může být libovolný, ale musí mít příponu `bat` (zkratka ze slova `batch` – dávka). V dávkovém souboru budou následující příkazy (na velikosti písmen nezáleží):

```
SUBST R: /d
```

```
SUBST R: .
```

První příkaz má za úkol zrušit případnou doposud nastavenou substituci disku **R:** (není-li v daném okamžiku označený disk substituován, systém vypíše chybovou zprávu, ale jinak se nic nestane), druhý příkaz pak substituuje aktuální složku jako disk **R:**.

Soubor umístíte do složky, z níž budete chtít udělat substituovaný disk. Kdykoliv pak tento dávkový soubor spustíte, dávka substituuje složku, v níž je umístěna, jako příslušný disk. Dávka se přitom spouští obdobně jako aplikace – např. poklepnáním na ikonu jejího souboru v *Průzkumníku*.

Kdykoliv od této chvíle budete pracovat s diskem **R:**, budete ve skutečnosti pracovat s obsahem substituované složky. A naopak: cokoliv uděláte s obsahem substituované složky, uděláte zároveň s obsahem disku **R:**.

Budete-li chtít mít danou substituci nastavenou trvale, můžete umístit zástupce dávkového souboru do položky *Po spuštění* ve startovní nabídce. Protože je v každé verzi operačního systému jinde, bude nejlepší, když ji ve startovní nabídce najdete, klepnete na ni pravým tlačítkem a v následně otevřené místní nabídce zadáte **Otevřít**. Tím otevřete okno průzkumníka s touto složkou. Pak v druhém okně průzkumníka otevřete složku s příslušným dávkovým souborem, uchopíte jeho ikonu **PRAVÝM** tlačítkem myši, přesunete ji do složky nabídky a pustíte. Otevře se místní nabídka (ta se otevře, pouze pokud přesouváte soubor pravým tlačítkem myši), ve které zadáte, že zde chcete vytvořit zástupce, a tím celý proces končí.

Abyste mohli složku substituuovat jako nějaký disk, nesmí váš operační systém používat disk označený tímto písmenem. Písmeno může být použito nejvýše pro jiný substituovaný disk, protože tuto substituci můžete před nastavením nové substituce zrušit (pro tento případ je v dávkovém souboru první příkaz s parametrem */d* – delete).

Používáte-li operační systém *Windows*, můžete urychlit budoucí vyhledání složky s projekty právě tím, že pro ni zřídíte zvláštní substituovaný disk. Kdykoliv se pak obrátíte na příslušný disk, obrátíte se ve skutečnosti k příslušné složce. Pomocí substituce si tak můžete zkrátit cestu k často používaným složkám.

A.2 Nastavování zástupce spouštějícího IDLE

Používáte-li *Windows* a nemáte-li zkušenosti s nastavováním zástupců, doporučuji využít mého zástupce a pouze mu upravit některá nastavení. Postupujte následovně:

6. Otevřete ve svém oblíbeném správci souborů (budu předpokládat, že jím je *Průzkumník* nebo *Total Commander*) složku s doprovodnými programy **71_INP**.
7. Klepněte na soubor **!IDLE_Python_39.lnk** (používáte-li *Průzkumník*, tak ten příponu zástupců, tj. **lnk**, nezobrazuje).
8. Stiskněte **ALT+ENTER**. Otevře se okno z obrázku [A.1](#), v němž můžete nastavit parametry zástupce.
9. V poli **Cíl** je zadáno:

```
C:\_PGM\Python\Python39\pythonw.exe  
C:\_PGM\Python\Python39\Lib\idlelib\idle.pyw
```

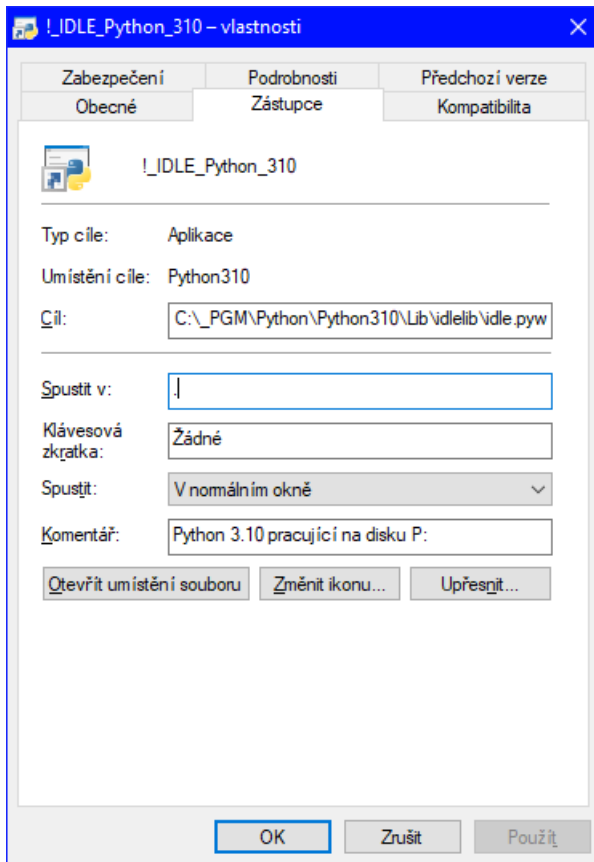
První část končí **pythonw.exe** je úplná cesta k programu spouštějícímu *Python*, který nebude otevírat okno příkazového řádku, protože očekává, že spouštěný skript používá GUI.

Tato část je nepovinná. Máte-li korektně instalovanou pouze jednu verzi *Pythonu*, nemusíte ji vůbec zadávat, protože *Windows* poznají, co mají spustit, podle

přípony spouštěného skriptu. Já ji zadávat musím, protože mám v době psaní knihy instalovanou vedle stabilního *Pythonu 3.8.3* také beta verzi *Pythonu 3.9*, kterou v této učebnici používám, a potřebuji si být jistý, která verze daný skript spouští.

Potřebujete-li proto stejně jako já také zadávat spouštěný program, musíte zde zadat úplnou cestu k tomuto programu na vašem počítači.

10. Druhá část příkazu je již povinná a zadává úplnou cestu ke spouštěnému skriptu, tj. k souboru `idle.pyw`. Vy budete mít spouštěný skript nejspíš jinde, tak zde musíte zadanou cestu příslušně upravit.
11. V poli **Spustit v** je zadána cesta ke složce se zdrojovými soubory doprovodných skriptů. Tu si také upravte podle svého počítače.
12. Když máte vše korektně upraveno, uložte upraveného zástupce stiskem **OK**, a od této chvíle můžete IDLE spouštět poklepaním na daného zástupce.



Obrázek A.1:
Okno zástupce spouštějícího IDLE

Příloha B

Syntaktické diagramy

Syntaktická pravidla zápisu složitějších konstrukcí jazyka jsou definována prostřednictvím syntaktických diagramů, které poskytují nejnázornější způsob jejich popisu. Tyto diagramy se v sedmdesátých letech minulého století používaly poměrně běžně, ale vzhledem k pracnosti jejich tvorby se následně z učebnic programování poněkud vytratily a v dnešní době se s nimi setkáte spíše výjimečně.

To ale nic nemění na skutečnosti, že jsou stále nejnázornějším popisem, prostřednictvím něž si může začátečník ověřit, jak přesně vypadá syntaxe použité konstrukce a kde pravděpodobně udělal ve svém programu chybu. Proto je také ve svých učebnicích používám.

Syntaktické diagramy jsou grafickým ekvivalentem zápisu v EBNF⁵⁹ používaném v převážné většině specifikací programovacích jazyků. V této pasáži si na příkladu zápisu čísel ukážeme základní pravidla interpretace syntaktických diagramů.

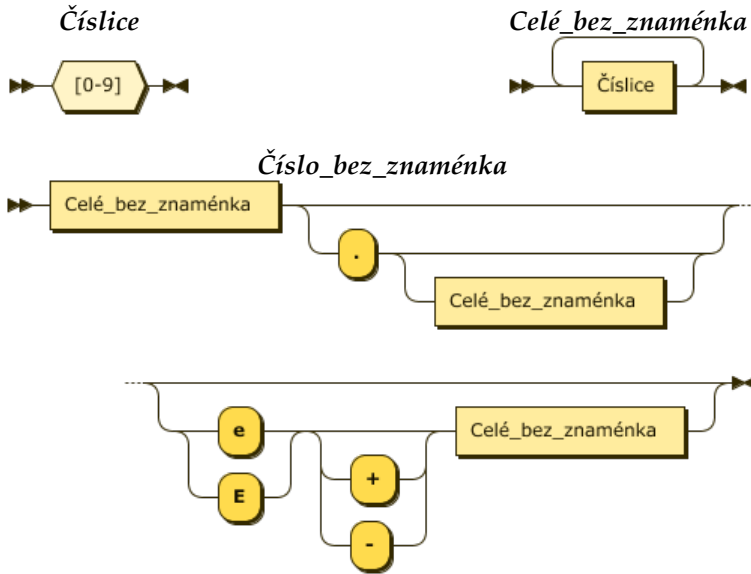
Syntaktické diagramy interpretujeme tak, že jakákoliv cesta od počáteční značky ►► (vstupního bodu) po koncovou značku ◄◄ (výstupního bodu) generuje syntakticky korektně zapsanou konstrukci jazyka. Pokud existují nějaké výjimky, bývají popsány v doprovodném textu. Cestou procházíte elementy, které budou jednoho ze tří druhů:

- Šestiúhelník obsahuje specifikaci zapsanou přímo v EBNF. Používá se spíše výjimečně, a to ve chvíli, kdy je tento zápis úspornější než odpovídající diagram, aniž by ztratil na své přehlednosti. Zápis v diagramu prvku *Číslice* na obrázku [B.1](#) např. označuje, že za číslici považujeme jakýkoliv znak od nuly do devítky.
- Obdélník označuje prvek, jenž je definován v jiném syntaktickém diagramu.
- Obdélník se zaoblenými rohy obsahuje znak či skupinu znaků, která se má na daném místě vložit (zapsat) do vytvářeného kódu.

Elementy jsou navzájem pospojovány spojnicemi určujícími možné cesty od prvku k prvku při skládání dané syntaktické konstrukce.

⁵⁹ EBNF je zkratka termínu *Extended Backus-Naur form* (Rozšířená Backus-Naurova forma), což je jazyk používaný k definici syntaxe programovacích jazyků.

Syntaxe zápisu čísel je specifikována prostřednictvím tří syntaktických diagramů na obrázku [B.1](#). Symbol v diagramu *Číslice* symbolizuje, že číslicí může být libovolný znak od 0 do 9. Diagram *Celé_bez_znaménka* prozrazuje, že tento druh entity je tvořen nejméně jednou číslicí, ale že číslic může být i více. Diagram *Číslo_bez_znaménka* pak specifikuje všechny povolené způsoby, jakými lze zapsat číslo.



Obrázek B.1:

Syntaktický diagram zobrazující pravidla pro zápis čísel

Příloha C

Konvence pro psaní programů v Pythonu

Oficiální konvence jsou včetně demonstračních ukázek podrobně popsány v dokumentu [PEP 8](https://www.python.org/dev/peps/pep-0008/),⁶⁰ který najdete na adrese <https://www.python.org/dev/peps/pep-0008/>, a [PEP 257](https://www.python.org/dev/peps/pep-0257/), který najdete na adrese <https://www.python.org/dev/peps/pep-0257/>. První probírá konvence pro psaní kódu, druhý konvence pro psaní dokumentačních komentářů. Zde uvedu jen stručný výčet.



Zájemcům, kteří chtějí rychle ověřit, že nějaký kód vyhovuje těmto konvencím, doporučuji, aby si na adrese <https://pep8.readthedocs.io> stáhli program nazvaný příhodně `pep8`, jenž dodržování těchto konvencí ověří za vás.

Uspořádání kódu

Hlavní zásadou při tvorbě kódu by měl být fakt, že program se daleko častěji čte, než zapisuje, takže bychom jej měli zapisovat tak, aby se následně dobře četl. Bude-li kód čitelnější, když v daném místě porušíte některou z konvencí, porušte ji. To je obzvláště důležité u programů určených pro výuku.

- Odsazujte o 4 znaky a v textu nepoužívejte tabulátory, ale jen mezery.
- Omezte délku řádků na 79 znaků, u komentářů (i dokumentačních) na 72 znaků.
- U delších řádků vkládejte konec řádku před binární operátor, takže výraz bude na dalším řádku začínat operátorem – např.

```
suma = (první proměnná
        + druhá proměnná)
```

⁶⁰ PEP je zkratka z anglického *Python Enhancement Proposal* (doporučení pro vylepšení *Pythonu*). Jsou to dokumenty poskytující informace komunitě *Pythonu* nebo popisující nové funkce, procesy nebo prostředí. Jejich přehled najdete na <https://www.python.org/dev/peps/>.

- Definice tříd a samostatných funkcí odděluje dvěma řádky, definice metod uvnitř třídy odděluje jedním prázdným řádkem.
- Používejte kódování UTF-8, a to bez občas nabízené úvodní deklarace.
- Vkládejte každý import modulu na samostatný řádek, import několika identifikátorů z daného modulu (`from ... import ...`) je však možné uvést společně.
- Umístěte importy na počátek modulu před definice globálních proměnných.
- Nepoužívejte hvězdičkový import (`from ... import *`).
- Identifikátory uvozené a ukončené dvojicí podtržení (tzv. *dunders* jako zkratka z anglického *double underscores*) by měly být definovány na počátku modulu za dokumentačním komentářem, ale před importy.
- Pro trojitě ohraničení stringů používejte trojitě uvozovky, abyste byli konzistentní s dokumentačními komentáři podle [PEP 257](#). Jednoduchý ohraničující znak (apostrof či uvozovky) používejte dle svých preferencí, ale buďte konzistentní.
- Mažte závěrečné mezery na koncích řádků. (Některé editory tuto funkci nabízejí.)
- U pojmenovaných argumentů nepoužívejte mezery kolem znaku `=`.
- Nevkládejte více příkazů na jeden řádek.
- U složených příkazů pokračujte na řádku za hlavičkou pouze v případě, že tělo tvoří jeden jednoduchý příkaz – např.

```
for x in lst: total += x
```

- Nebojte se použít závěrečné čárky v seznamech argumentů, mohou usnadnit přidání dalšího členu – např.

```
FILES = [
    'setup.cfg',
    'tox.ini',
]
```

- Komentáře, které neodpovídají kódu, jsou horší než žádné. Udržujte komentáře neustále aktuální (*up-to-date*).

Jmenné konvence

Jmenné konvence bohužel nejsou zcela konzistentní, nicméně existuje několik doporučení, které by měly nové programy dodržovat.

- Ve standardní knihovně musí všechny identifikátory používat pouze znaky **ASCII** a měly by pokud možno používat angličtinu. Doporučuje se toto pravidlo dodržovat i v ostatních programech.
- Názvy viditelné uživateli jako součást API by měly reflektovat spíše užití než implementaci.

- Vyvarujte se názvů tvořených pouze znakem `l` (malé L), `0` (velké o) nebo `I` (velké i). Některé fonty neumí odlišit znaky `l-l-I` (jedna – L – I) a `0-0` (nula – o).
- Používejte ve svém vývojovém prostředí font, který tyto znaky odliší.
- Názvy modulů by měly být krátké a používat jen malá písmena. Použití znaku podtržení se nedoporučuje.
- Názvy tříd by měly používat `VelbloudíNotaci`.
- Názvy funkcí by měly používat jen malá písmena a jednotlivá slova názvu oddělovat podtržítky – např. `long_function_name`.
- První parametr instančních metod by se měl vždy jmenovat `self`.
- První parametr třídních metod by se měl vždy jmenovat `cls`.
- Názvy neveřejných metod a instančních proměnných by měly začínat podtržítkem.
- Názvy konstant by měly používat pouze velká písmena a jednotlivá slova názvu oddělovat podtržítky – např. `LONG_CONSTANT_NAME`.
- U každého atributu (proměnné i metody) se vždy rozmyslete, zda má být veřejný. Neveřejný atribut lze snadno zveřejnit, obrácená operace je po rozšíření dané třídy či modulu již zakázaná.

Dokumentační komentáře ([PEP 257](#))

Dokumentační komentář je stringový literál zadaný jako první příkaz v modulu, třídě, metodě či funkci. Ten se automaticky stane hodnotou atributu `__doc__` daného objektu.

- Dokumentační komentář by měly mít všechny moduly a všechny funkce a třídy exportované z těchto modulů.
- Dokumentační komentář balíčku je možné zadat v jeho souboru `__init__.py`.
- Dokumentační komentáře ohraničujte vždy trojitými uvozovkami (`"""`), s případnou předponou `r`, používáte-li v nich zpětná lomítka.
- Trojitě uvozovky používejte, i když se komentář vejde na řádek, jak je tomu např. ve výpisu [38.2](#) na straně [573](#). Neoddělujte ohraničení mezerami.
- Víceřádkové komentáře zahajte jednořádkovým shrnutím umístěným na stejném řádku s ohraničujícími uvozovkami. Protože může být použito automatickými indexačními programy, je důležité, aby bylo na jednom řádku a aby bylo od dalšího textu odděleno prázdným řádkem. Další řádky komentáře zarovnávejte stejně jako uvozovky na prvním řádku.
- Za dokumentačním komentářem třídy vynechte řádek.
- Dokumentační komentář modulu by měl vypsat všechny třídy, výjimky a funkce daného modulu.

- Dokumentační komentář třídy by měl vypsat její chování, seznam veřejných metod a atributů včetně instančních.
- Dokumentační komentář funkce by měl vypsat její funkci, argumenty a návratovou hodnotu, vedlejší efekty a případná omezení.

Příloha D

Stručná historie posledních verzí

Jak jsem již řekl v pasáži [Potřebné vybavení](#) na straně [30](#), pokusím se knihu psát tak, aby ji mohli používat i uživatelé starších verzí než verze 3.10, která je uvedena v podtitulu. Nebudu se ale dívat příliš zpět. Kniha předpokládá, že všichni mají instalovánu minimálně verzi 3.6, která vyšla v prosinci 2016 a která je velmi často deklarována jako povinný základ. Následující přehled stručně připomíná verze, jež vyšly v posledních letech. Přidaných vlastností a rysů je samozřejmě mnohem více, uvádím jen ty, které se přímo týkají probíraných konstrukcí. Kompletní přehled najdete v dokumentaci, která je součástí standardní instalace (viz pasáž [Dokumentace](#) na straně [37](#)).

- Verze 3.6 – prosinec 2016:
 - F-stringy, např. `f'Proměnná x = {x}'`.
 - Anotace proměnných, např. `atr:str`.
 - Podtržení pro zpřehlednění čísel, např. `123_456_789`.
 - Konzole *Windows* akceptuje UTF-8.
 - Slovník iteruje v pořadí přidání.
- Verze 3.7 – červen 2018:
 - Klíčová slova `async` a `await`.
 - Zrušení `ASCII` jako implicitního kódování textu.
 - Příprava odloženého vyhodnocování anotací.
 - Funkce mohou mít více než 255 argumentů.
- Verze 3.8 – listopad 2019:
 - Přiřazovací výraz, např. `pi:=3.14`.
 - Pouze poziční parametry, např. `divmod(x, y, /)`.
 - Přiřazení v nahrazovacích polích f-stringů, např. `f'Proměnná {x}'`.

- Verze 3.9 – říjen 2020:
 - Slovníky akceptují operátor `|` a `|=`, např. `d |= {k1:v1, k2:v2}`.
 - V anotacích lze používat deklarace kontejnerů standardních typů, např. `iarr:tuple[int]`.
 - Stringům přibyly metody pro odstranění předpon a přípon.
 - Uvolnění gramatických omezení pro dekorátory.
 - Přidána třída `typing.Annotated` pro rozšíření možností anotací.
- Verze 3.10 – říjen 2021:
 - Zavedena měkká klíčová slova.
 - V příkazu `with` je nyní možné použít závorky, a tím umožnit rozložení hlavičky na více řádků.
 - Upřesnění hlášení o syntaktické chybě.
 - Zavedení přepínače `match ... case`.
 - Vyhodnocování anotací je implicitně odloženo; anotace se nyní ukládají jako stringy.
 - Datové typy lze nyní sjednocovat, např. anotace `number:int|float` deklaruje, že proměnná `number` bude buď celé, nebo reálné číslo.
 - Zaveden datový typ `typing.Calleable` umožňující deklarovat typy parametrů volatelných objektů.
 - Zavedena anotace `TypeAlias` označující, že anotovaná proměnná bude anotací – hovoří se o anotacích přezdívek typů (TypeAlias Annotation).
- Verze 3.11 – říjen 2022:
 - Hlavním vylepšením bylo zrychlené provádění programů o 20 až 60 %, v průměru o přibližně 25 %.
 - Skupiny výjimek (PEP 654).
 - Výjimky mohou být doplněny poznámkami (PEP 678).
 - Do standardní knihovny přibyla podpora formátu TOML (PEP 680)
 - Podrobněji označená pozice chyby v chybovém hlášení (PEP 657).
 - Variadické generické typy (PEP 646).
 - Možnost označení jednotlivých položek v `TypedDict` jako `required`, anebo `not-required` (PEP 655).
 - Typ `Self` (PEP 673)
 - Anotační typ `LiteralString` indikující parametr, jenž může obsahovat pouze stringovou hodnotu zadanou jako literál (PEP 675).
 - Anotace `dataclass_transform`, které označuje, že anotovaný objekt (třída, metatřída či funkce) bude sloužit jako dekorátor (PEP 681).

- V příkazu `for` jsou povoleny „hvězdičkové rozbalovací výrazy“, takže např. část kódu:

```
a = [1, 2, 3]; b = (11, 22, 33)
for x in *a, *b: print(x, end=' - ')
```

vytiskne

```
1-2-3-11-22-33-
```

(Toto rozšíření funguje od verze 3.9, ale zveřejněno bylo až ve verzi 3.11.)

- Přidána metoda `object.__getstate__()`, poskytující výchozí implementaci metody `__getstate__()`. Použití metod `copy()` a `pickle()` instancí podtříd vestavěných typů `bytearray`, `set`, `frozenset`, `collections.OrderedDict`, `collections.deque`, `weakref.WeakSet` a `datetime.tzinfo` nyní kopíruje a vybírá atributy instancí implementované jako sloty.

Část G

Seznamy

Tato část obsahuje seznamy hypertextových odkazů na výpisy programů, obrázky, tabulky a odbočky – podšeděné bloky. Není součástí tištěné knihy, aby zbytečně nezvyšovala její cenu. Je však součástí elektronických verzí, v nichž můžete hypertextové odkazy efektivně využít.

Seznam výpisů programů

Výpis 1.1:	Komunikace s interpretem spuštěným v konzolovém okně	41
Výpis 1.2:	Reakce na odsazení příkazu	45
Výpis 2.1:	Reakce na zadání velkého čísla	54
Výpis 2.2:	Zpřehlednění čísel vložením podtržitek	54
Výpis 2.3:	Zadávání a zobrazování desetinných čísel	55
Výpis 2.4:	Zadávání a zobrazování komplexních čísel	55
Výpis 2.5:	Zápis čísel v alternativních číselných soustavách	56
Výpis 2.6:	Reakce na použití objektů None a ... (Ellipsis)	58
Výpis 3.1:	Zápis jednořádkového textu	60
Výpis 3.2:	Zápis víceřádkového textu	61
Výpis 3.3:	Komentáře	63
Výpis 3.4:	Zadání znaků pomocí escape sekvencí	64
Výpis 3.5:	Zadání na více řádcích	66
Výpis 3.6:	Vytvoření bajtových stringů za použití literálů	69
Výpis 4.1:	Zadání argumentu nápovědy pomocí stringu	82
Výpis 4.2:	Přepnutí do režimu nápovědy a zpět	83
Výpis 4.3:	Volání funkce rozepsané na více řádcích	84
Výpis 5.1:	Ukázky tří druhů dělení	90
Výpis 5.2:	Komplexní číslo jako exponent – Eulerova rovnost	91
Výpis 5.3:	Nekonečna a nesmyslná čísla	92
Výpis 5.4:	Sčítání textů	93
Výpis 5.5:	Násobení textů	93
Výpis 5.6:	Indexace stringů	94
Výpis 6.1:	Přiřazení hodnoty proměnné	100
Výpis 6.2:	Skrytá běhová chyba	101
Výpis 6.3:	Současné přiřazení skupiny hodnot proměnným	102
Výpis 6.4:	Zavedení proměnných <code>inf</code> a <code>nan</code>	103
Výpis 6.5:	Ukázky vnořeného volání funkcí	104
Výpis 6.6:	Zjištění typu objektu	105
Výpis 6.7:	Přiřazení funkcí (přesněji odkazů na funkce) do proměnných	106
Výpis 6.8:	Definice a použití lambda-výrazu	107
Výpis 6.9:	Použití lambda-výrazu při snižování počtu argumentů	109
Výpis 6.10:	Přiřazovací výraz a jeho použití	111
Výpis 6.11:	Ovlivnění toho, bude-li se zobrazovat systémový či uživatelský podpis	113
Výpis 7.1:	Převody logických hodnot na čísla a jiných hodnot na logické	115
Výpis 7.2:	Zřetěžené porovnávání	117
Výpis 7.3:	Test totožnosti objektů	118
Výpis 7.4:	Chování logických operátorů	119
Výpis 7.5:	Vliv priorit na výsledek	120
Výpis 7.6:	Chování bitových operátorů	121

Výpis 7.7:	Bitové posuny.....	122
Výpis 7.8:	Použití podmíněného výrazu.....	123
Výpis 8.1:	Složené přiřazovací operátory	126
Výpis 8.2:	Použití příkazu del	127
Výpis 8.3:	Použití příkazu assert.....	129
Výpis 8.4:	Spuštění systému Python s aktivovaným a deaktivovaným příkazem assert	130
Výpis 8.5:	Kontrola shodnosti odsazení příkazů	130
Výpis 9.1:	Importování modulů	138
Výpis 9.2:	Přejmenování importovaného modulu	139
Výpis 9.3:	Import zadaných objektů ze zadaného modulu.....	141
Výpis 9.4:	Použitelnost proměnné s odkazem na modul	142
Výpis 9.5:	Import všech objektů ze zadaného modulu.....	143
Výpis 9.6:	Demonstrace nárůstu počtu atributů po hromadném importu	143
Výpis 9.7:	Modul builtins.....	146
Výpis 10.1:	Modul m10a_Modul sloužící k demonstraci chování Pythonu při zavádění modulu.....	149
Výpis 10.2:	Import modulu m10a_Modul	154
Výpis 10.3:	Nové načtení modulu m10a_Modul	158
Výpis 10.4:	„Neaktualizace“ přímo importovaných odkazů	158
Výpis 10.5:	Opětné načtení verze s chybou	159
Výpis 10.6:	Definice modulu m10b_Demo_all vyjmenovávajícího veřejné atributy	161
Výpis 10.7:	Hvězdičkový import modulu m10b_Demo_all.....	162
Výpis 10.8:	Import modulu posixpath nebo ntpath zprostředkovaný modulem os	163
Výpis 10.9:	Prověrka zprostředkování importu	163
Výpis 10.10:	Modul m10c_Circular_C importující modul m10d_Circular_D.....	164
Výpis 10.11:	Modul m10d_Circular_D importující modul m10c_Circular_C.....	164
Výpis 10.12:	Pokus o import modulu m10c_Circular_C a reakce na něj	165
Výpis 11.1:	Jednořádková definice funkce.....	169
Výpis 11.2:	Víceřádková definice funkce	170
Výpis 11.3:	Přisazení definice funkce v interaktivním režimu	173
Výpis 11.4:	Modul m11a_Funkce_v_modulu demonstrující pravidla pro definice funkcí v modulu	174
Výpis 11.5:	Dynamická povaha jazyka při definici a volání funkcí	175
Výpis 12.1:	Definice a volání funkce s parametrem.....	178
Výpis 12.2:	Definice funkce s lokálními proměnnými	179
Výpis 12.3:	Volání funkcí s parametry	179
Výpis 12.4:	Funkce s povinně pojmenovanými argumenty.....	180
Výpis 12.5:	Funkce s povinně pozičními argumenty.....	181
Výpis 12.6:	Funkce s různými kombinacemi povinných parametrů	183
Výpis 12.7:	Volání funkce print() s vlastní hodnotou implicitního argumentu	184
Výpis 12.8:	Definice funkce se dvěma implicitními argumenty a možností jejího volání.....	185
Výpis 12.9:	Konstantnost předdefinovaných hodnot.....	185
Výpis 12.10:	Funkce s vedlejšími efekty.....	186
Výpis 12.11:	Definice a použití funkcí vracejících hodnotu	187
Výpis 12.12:	Nepoužitelnost přetěžování funkcí	188
Výpis 12.13:	Deklarace typů proměnných a návratových hodnot prostřednictvím anotací.....	189
Výpis 12.14:	Definice modulu m11b_dbg_demo	193
Výpis 12.15:	Import modulu m11b_dbg_demo a zavolání jeho funkce fce1()	193
Výpis 13.1:	Definice vnitřních funkcí	196
Výpis 13.2:	Zakrytí globální proměnné stejnojmennou lokální.....	201

Výpis 13.3:	Použití příkazu <code>global</code>	202
Výpis 13.4:	Použití příkazu <code>nonlocal</code>	203
Výpis 13.5:	Demonstrace vnořeného volání funkcí.....	206
Výpis 13.6:	Použití funkce vracející funkci.....	207
Výpis 13.7:	Využití atributů funkce.....	207
Výpis 13.8:	Nelokální proměnné a uzávěry.....	209
Výpis 13.9:	Definice modulu <code>m13b_Circular_B</code> importujícího modul <code>m13c_Circular_C</code>	210
Výpis 13.10:	Definice modulu <code>m13c_Circular_C</code> importujícího modul <code>m13b_Circular_B</code>	210
Výpis 13.11:	Import modulu <code>m13b_Circular_B</code>	210
Výpis 14.1:	Ukázka použití jednoduchého podmíněného příkazu.....	213
Výpis 14.2:	Ukázka použití úplného podmíněného příkazu.....	214
Výpis 14.3:	Ukázky použití rozšířeného podmíněného příkazu.....	216
Výpis 14.4:	Řešení jednoduché úlohy prostřednictvím přepínače.....	217
Výpis 14.5:	Ukázka možného sdružování hodnot ve vzorech.....	218
Výpis 14.6:	Přímé zadání podmíněného příkazu v interaktivním režimu.....	219
Výpis 15.1:	Ukázky definic funkcí používajících rekurzivní volání.....	221
Výpis 15.2:	Rekurzivní definice faktoriálu.....	222
Výpis 15.3:	Ukázky definic funkcí používajících cyklus <code>while</code>	223
Výpis 15.4:	Nekonečný cyklus.....	224
Výpis 15.5:	Použití příkazu <code>break</code>	225
Výpis 15.6:	Použití přiřazovacího výrazu v hlavičce cyklu <code>while</code>	227
Výpis 15.7:	Cyklus <code>while</code> využívající větve <code>else</code>	228
Výpis 15.8:	Použití příkazu <code>continue</code>	229
Výpis 15.9:	Použití příkazu <code>for</code> se zdrojem definovaným jako sada hodnot.....	230
Výpis 15.10:	Použití příkazu <code>for</code> se zdrojem generovaným funkcí <code>range()</code>	231
Výpis 15.11:	Indexace znaků v textovém řetězci – stringu.....	232
Výpis 15.12:	Zpracování jednotlivých znaků v textovém řetězci bez použití indexů.....	232
Výpis 15.13:	Ukázka zanořování cyklů.....	233
Výpis 15.14:	Příkaz <code>for</code> s postupným použitím více zdrojů.....	234
Výpis 15.15:	Příkaz <code>for</code> s větví <code>else</code>	234
Výpis 16.1:	Ukázky syntaktických chyb a reakcí na ně při zadávání příkazů v prostředí IDLE.....	237
Výpis 16.2:	Ukázky syntaktických chyb a reakcí na ně při zadávání příkazů z konzolového okna.....	239
Výpis 16.3:	Zopakovaný výpis 10.5 ze strany 159 zobrazující načtení třetí verze modulu <code>m10a_Modul</code> z výpisu 10.1 na straně 149.....	240
Výpis 16.4:	Ukázka zachycení a ošetření výjimky.....	243
Výpis 16.5:	AHA-příklad používající příkaz <code>try</code> se všemi dostupnými větvemi.....	247
Výpis 16.6:	Měření rychlosti počítače a přesnosti odhadu času.....	248
Výpis 16.7:	Zdánlivé záludnosti větve <code>finally</code>	249
Výpis 16.8:	Funkce <code>assert_stmt()</code> demonstrující funkci příkazu <code>assert</code>	251
Výpis 16.9:	Přidání poznámky k zachycené výjimce.....	252
Výpis 17.1:	Demonstrace neměnnosti objektů.....	255
Výpis 17.2:	Vytváření seznamů pomocí literálů.....	257
Výpis 17.3:	Vytváření seznamů pomocí konstruktoru <code>list()</code>	258
Výpis 17.4:	Vytváření seznamů z jiných seznamů pomocí sčítání a násobení.....	259
Výpis 17.5:	Vytvoření seznamu pomocí generátorové notace.....	260
Výpis 17.6:	Použití funkcí <code>append()</code> a <code>extend()</code>	262
Výpis 17.7:	Demonstrace chování proměnných objektů.....	263
Výpis 17.8:	Postupné budování seznamu.....	264

Výpis 17.9:	Rozšiřování seznamu přičítáním	264
Výpis 17.10:	Chybné pokusy o rozšíření seznamu	265
Výpis 17.11:	Ukázky použití modifikačních funkcí a metod pracujících s indexy	266
Výpis 17.12:	Ukázky použití modifikačních metod pracujících s celým seznamem	268
Výpis 17.13:	Modul <code>m17a_Pascal</code> s funkcemi pracujícími s vícerozměrnými seznamy	269
Výpis 17.14:	Načtení modulu <code>m17a_Pascal</code> a prověření jeho funkce	269
Výpis 17.15:	Anotace odkazující na seznamy	270
Výpis 18.1:	Vytváření n-tic pomocí literálů	272
Výpis 18.2:	Vytváření n-tic pomocí konstruktoru <code>tuple()</code>	274
Výpis 18.3:	Vytváření n-tic z jiných n-tic pomocí sčítání a násobení	274
Výpis 18.4:	Vytváření n-tic přičítáním	275
Výpis 18.5:	Balení a rozbalování n-tic a seznamů	276
Výpis 18.6:	Prohazování hodnot proměnných	277
Výpis 18.7:	Hvězdičkové pravidlo	277
Výpis 18.8:	Vytvoření n-tice pomocí generátoru	278
Výpis 18.9:	Přístup k prvkům n-tic	280
Výpis 18.10:	Sčítání seznamů a n-tic	280
Výpis 18.11:	Chování n-tic s proměnnými a neměnnými prvky	281
Výpis 18.12:	Prvky standardních n-tic nemají jména	282
Výpis 18.13:	Pojmenované n-tice	282
Výpis 18.14:	Pojmenované n-tice s implicitními argumenty	283
Výpis 19.1:	Vytváření množin pomocí literálů	285
Výpis 19.2:	Vytváření množin pomocí konstruktoru <code>set()</code>	286
Výpis 19.3:	Povolené a nepovolené argumenty konstruktoru <code>set()</code>	286
Výpis 19.4:	Vytváření množin prostřednictvím množinových operací	287
Výpis 19.5:	Vytváření množin pomocí generátorové notace	289
Výpis 19.6:	Vytváření zmrazených množin	290
Výpis 19.7:	Jednoprvkové modifikační množinové operace	291
Výpis 19.8:	Modifikace množin množinami či jinými zdroji	292
Výpis 19.9:	Porovnání množin	294
Výpis 20.1:	Vytváření slovníků pomocí literálů	296
Výpis 20.2:	Vytváření slovníků pomocí konstruktoru <code>dict()</code>	297
Výpis 20.3:	Vytváření slovníků pomocí funkce <code>fromkeys()</code>	299
Výpis 20.4:	Vytváření slovníků pomocí generátorové notace	299
Výpis 20.5:	Přístup k položkám prostřednictvím klíčů	301
Výpis 20.6:	Metody pro práci s jednotlivými položkami	302
Výpis 20.7:	Použití metody <code>update()</code>	303
Výpis 20.8:	Pohledy	304
Výpis 20.9:	Pohledy jako generátory	305
Výpis 20.10:	Operace s pohledy	306
Výpis 21.1:	Demonstrace chování programu při předávání argumentů hodnotou	308
Výpis 21.2:	Definice a ukázky použití funkce <code>gr()</code>	309
Výpis 21.3:	Zpracování argumentů při použití hvězdičkového parametru	311
Výpis 21.4:	Použití a zpracování hvězdičkového argumentu	312
Výpis 21.5:	Zpracování argumentů při použití dvouhvězdičkového parametru	313
Výpis 21.6:	Zpracování dvouhvězdičkového argumentu	314
Výpis 21.7:	Příklady zpracování argumentů inspirované [5]	315
Výpis 21.8:	Možné použití hvězdičkových a dvojhvězdičkových argumentů	316
Výpis 22.1:	Použití formátovacího operátoru <code>%</code>	322
Výpis 22.2:	Použití prázdné formátové položky	324

Výpis 22.3:	Různé způsoby zadávání nahrazovaného textu	326
Výpis 22.4:	Efekt zadání konverze konverzí	327
Výpis 22.5:	Zadání požadovaného minimálního počtu pozic	328
Výpis 22.6:	Zadání požadované přesnosti, resp. maximálního počtu pozic	329
Výpis 22.7:	Typy formátů pro celočíselné hodnoty	330
Výpis 22.8:	Typy formátů pro zobrazení reálných čísel	331
Výpis 22.9:	Použití oddělovače skupin číslic	332
Výpis 22.10:	Zadání oddělovačů skupin číslic a alternativního formátu	333
Výpis 22.11:	Zarovnání a plnění	335
Výpis 22.12:	Používání vnořených nahrazovacích polí	336
Výpis 22.13:	Definice funkce <code>printpasf()</code> v modulu <code>m22a_PrintPasF</code>	337
Výpis 22.14:	Použití funkce <code>printpasf()</code> v modulu <code>m22a_PrintPasF</code>	338
Výpis 22.15:	Trasovací funkce <code>prSE()</code> v modulu <code>dbg</code>	339
Výpis 22.16:	Trasovací funkce <code>prIN()</code> v modulu <code>dbg</code>	340
Výpis 22.17:	Ukázka použití trasovacích funkcí <code>prSE()</code> a <code>prIN()</code>	341
Výpis 23.1:	Hluboké a mělké kopie	344
Výpis 23.2:	Použití operátorů <code>in</code> a <code>not in</code>	347
Výpis 23.3:	Použití funkce <code>reversed()</code>	348
Výpis 23.4:	Použití funkce <code>sorted()</code>	348
Výpis 23.5:	Indexování a vykrajování u rozsahů – objektů typu <code>range</code>	350
Výpis 23.6:	Modifikace obsahu posloupnosti	351
Výpis 23.7:	Postup při výměně prvku v <code>n</code> -tici	352
Výpis 24.1:	Zjištění a změna aktuální složky	360
Výpis 24.2:	Syntéza a analýza cest	362
Výpis 24.3:	Vytváření a mazání složek	363
Výpis 24.4:	Získání informací o souborech	364
Výpis 24.5:	Otevření souboru	367
Výpis 24.6:	Zápis dat, splachování a zavírání souborů	369
Výpis 24.7:	Čtení souborů	372
Výpis 26.1:	Definice třídy a jejích atributů	388
Výpis 26.2:	Práce s atributy objektu	390
Výpis 26.3:	Přidávání a odstraňování atributů objektu	390
Výpis 26.4:	Přidávání a odstraňování metody	391
Výpis 26.5:	Třída jako parametr funkce	392
Výpis 26.6:	Vytvoření prvních instancí	393
Výpis 26.7:	Kvalifikace atributu třídy instancí	394
Výpis 26.8:	Možná použití dekorátoru	396
Výpis 26.9:	Přidání a použití instanční metody	397
Výpis 26.10:	Definice třídy <code>C2</code>	399
Výpis 26.11:	Vytváření instancí třídy <code>C2</code> a používání jejích atributů	400
Výpis 26.12:	Vytváření nových instančních datových atributů	401
Výpis 26.13:	Změny třídních datových atributů	402
Výpis 26.14:	Třídni datový atribut jako implicitní hodnota instančního	403
Výpis 26.15:	Zavádění nových třídních funkčních atributů	404
Výpis 26.16:	Zavádění nových instančních funkčních atributů	405
Výpis 26.17:	Demonstrace nutnosti kvalifikace atributů	407
Výpis 26.18:	Ukázky použití systémových atributů	408
Výpis 27.1:	Definice tříd <code>LA</code> , <code>LB</code> a <code>LC</code> v modulu <code>m27a_LABC</code>	413
Výpis 27.2:	Jmenné prostory a dosažitelné atributy tříd z modulu <code>m27a_LABC</code>	415

Výpis 27.3:	Jmenné prostory a dosažitelné atributy instancí tříd z modulu m27a_LABC	416
Výpis 27.4:	Volání metod v hierarchii dědění	417
Výpis 27.5:	Demonstrace dědění initorů	418
Výpis 27.6:	Ukázka definice a použití vlastní výjimky	420
Výpis 28.1:	Zdrojový kód tříd z modulu m28a_Di amant demonstrujících chování tříd uspořádaných do diamantové architektury	423
Výpis 28.2:	Prověrka vlastností násobného dědění	424
Výpis 28.3:	Definice třídy Dcera2 v modulu m28a_Di amant	426
Výpis 28.4:	Použití třídy Dcera2	426
Výpis 28.5:	Definice modulu m28b_Arguments s třídami používajícími násobné dědění a konstruktory s parametry	427
Výpis 28.6:	Průběh inicializace instance třídy DceraA	428
Výpis 28.7:	Nerealizovatelná a následně opravená hierarchie tříd	430
Výpis 28.8:	Definice složitější hierarchie tříd	430
Výpis 28.9:	Pseudosoukromý atribut a komolení jmen	432
Výpis 28.10:	Definice tříd Bc a Mc v modulu m28c_Komolení	433
Výpis 28.11:	Použití tříd Bc a Mc a jejich instancí	434
Výpis 29.1:	Přímá definice vlastností jako instance třídy property	437
Výpis 29.2:	Definice vlastností pomocí lambda-výrazů	439
Výpis 29.3:	Důsledky změny třídního atributu definujícího vlastnost	440
Výpis 29.4:	Zadání vlastnosti pomocí dekorátoru	441
Výpis 29.5:	Demonstrace použití vlastností instancí třídy C29c	442
Výpis 29.6:	Definice formálně abstraktní třídy Matka29a bez abstraktních metod	444
Výpis 29.7:	Definice abstraktní třídy Matka29b s abstraktní metodou	445
Výpis 29.8:	Definice potomka abstraktní třídy nepřebíjejícího abstraktní metodu	446
Výpis 29.9:	Definice třídy Dcera29b, jež je potomkem abstraktní třídy Matka29b	446
Výpis 29.10:	Definice dalších druhů abstraktních metod	448
Výpis 30.1:	Vytvoření výčtového datového typu prostřednictvím volání funkce Enum() a demonstrace vlastností daného typu a jeho atributů–instancí	453
Výpis 30.2:	Definice výčtového datového typu jako třídy	455
Výpis 30.3:	Definice výčtového typu Směr demonstrujícího některé další možnosti	456
Výpis 30.4:	Automatické přiřazování obalovaných hodnot	458
Výpis 30.5:	Poloautomatické nastavování obalovaných hodnot	458
Výpis 30.6:	Definice datové třídy a její test	460
Výpis 30.7:	Získání hodnot v rámci vyhodnocování vzorů v příkazu match	462
Výpis 30.8:	Prověřování posloupností	463
Výpis 30.9:	Použití vzoru s výběrem hodnot	464
Výpis 31.1:	Šablona initorů standardních balíčků uložená v souboru __init__.py v kořenovém balíčku	468
Výpis 31.2:	Výkonný kód modulu m1_cls v balíčku PKG.sub_pkg.subsub_pkg	468
Výpis 31.3:	Import modulu PKG.sub_pkg.subsub_pkg.m1_cls a jeho opětné načtení po úpravě	469
Výpis 31.4:	Kód modulu PKG.sub_pkg.subsub_pkg.m2_dot bez úvodních a závěrečných tisků	470
Výpis 31.5:	Import modulu PKG.sub_pkg.subsub_pkg.m2_dot	471
Výpis 31.6:	Kód modulu PKG.sub_pkg.subsub_pkg.m3_star, tentokrát včetně úvodních a závěrečných tisků	472
Výpis 31.7:	Průběh m3_star run()	473
Výpis 31.8:	Obsah souboru __init__.py v balíčku PKG.sub_pkg.subsub_pkg	474

Výpis 31.9:	Průběh znovuzavedení aktualizovaného balíčku <code>PKG.sub_pkg.subsub_pkg</code>	474
Výpis 31.10:	Průběh znovuzavedení aktualizovaného balíčku <code>m3_star</code>	475
Výpis 31.11:	Definice modulu <code>mA_importing</code> v balíčku <code>NS.ns_pkg</code> ve standardním zdroji.....	478
Výpis 31.12:	Definice modulu <code>mB_imported</code> v balíčku <code>NS.ns_pkg</code> v externím zdroji.....	478
Výpis 31.13:	Zavedení modulu <code>mA_importing</code>	479
Výpis 31.14:	Definice souboru <code>__init__</code> – initoru balíčku <code>NS.pkg</code> ve stromu <code>71_NS</code>	481
Výpis 31.15:	Import atributu balíčku <code>NS.pkg</code>	481
Výpis 32.1:	Definice modulu <code>m32a_Script</code>	483
Výpis 32.2:	Reakce na spuštění modulu <code>m32a_Script</code> v interaktivním režimu.....	484
Výpis 32.3:	Reakce na spuštění modulu <code>m32a_Script</code> v příkazovém okně Windows.....	484
Výpis 32.4:	Modul <code>m4_run</code> v balíčku <code>PKG.sub_pkg.subsub_pkg</code>	485
Výpis 32.5:	Vytvoření samostatné aplikace pomocí modu <code>winapp</code> a její spuštění.....	486
Výpis 32.6:	Definice modulu <code>__main__</code> v souboru <code>PKG.pyz</code>	486
Výpis 32.7:	Definice funkce <code>main()</code> v modulu <code>m32b_APR</code>	488
Výpis 32.8:	Spouštění skriptu <code>m32b_APR</code>	489
Výpis 33.1:	Definice třídy <code>Prime</code> , její instance jsou iterovatelné objekty a současně iterátory.....	494
Výpis 33.2:	Generátorový výraz a jeho použití	495
Výpis 33.3:	Nekorektní použití generátorového výrazu	496
Výpis 33.4:	Definice a použití generátorové funkce <code>prime_fun()</code>	498
Výpis 33.5:	Definice generátorové funkce <code>gen1()</code> demonstrující vícenásobné použití příkazu <code>yield</code>	499
Výpis 33.6:	Demonstrace prostřednictvím ruční aktivace výrazu <code>yield</code>	502
Výpis 33.7:	Definice generátorové funkce <code>ord_gen()</code> pracující s výrazem <code>yield</code>	504
Výpis 33.8:	Definice funkce <code>orders()</code> využívající možnost ovlivnit generátor	505
Výpis 33.9:	AHA-příklad s definicí generátorových funkcí <code>gen2()</code> a <code>gen21()</code> demonstrujících použití výrazů <code>yield</code> a <code>yield from</code>	507
Výpis 34.1:	Využití dekorátoru <code>total_ordering</code> pro doplnění porovnávacích metod	511
Výpis 34.2:	Definice třídy <code>Zlomek</code> v modulu <code>m34a_Zlomky</code>	515
Výpis 34.3:	Práce s instancemi třídy <code>Zlomek</code>	516
Výpis 34.4:	Definice třídy <code>Indexer</code> přetěžující operátor <code>[]</code>	518
Výpis 34.5:	Přidání instanční metody pro konkrétní instanci	519
Výpis 34.6:	Demonstrace rozdílů v kódu při použití příkazu <code>with</code>	521
Výpis 34.7:	Emulace postupu při zadání příkazu <code>with</code>	522
Výpis 35.1:	Anotace parametrů funkce, její návratové hodnoty a atributů modulu	526
Výpis 35.2:	Zapamatované anotace při okamžitém vyhodnocení platné doposud	528
Výpis 35.3:	Zapamatované anotace při odloženém vyhodnocení plánované od verze 3.11	528
Výpis 35.4:	Co o anotacích prozradí nápověda	529
Výpis 35.5:	Definice a použití přezdívky datových typů	531
Výpis 35.6:	Slučování datových typů	532
Výpis 36.1:	Ukázka použití třídních metod.....	536
Výpis 36.2:	Ukázka definice tovární metody	538
Výpis 36.3:	Definice dekorátoru – třídy <code>traced</code> v modulu <code>m36a_Traced</code>	539
Výpis 36.4:	Použití dekorátoru <code>traced</code>	540
Výpis 36.5:	Definice dekorátorů <code>jedinacka</code> v modulu <code>m36c_Singleton</code>	543
Výpis 36.6:	Použití dekorátoru <code>singleton</code>	544
Výpis 37.1:	Ekvivalent definice vestavěné třídy <code>property</code> v modulu <code>m37a_Property</code>	547
Výpis 37.2:	Definice třídy <code>descriptor</code>	548

Výpis 37.3:	Hrátky s deskriptorem	550
Výpis 37.4:	Postup metody <code>__getattr__()</code> při vyhodnocování atributu instance	552
Výpis 37.5:	Deskriptor pro odložené vyhodnocení	553
Výpis 37.6:	Definice třídy <code>C37c</code> v modulu <code>m37c_Atributy</code>	556
Výpis 37.7:	Ukázky reakcí na přístup k atributům ve třídě <code>C37c</code>	557
Výpis 37.8:	Definice třídy <code>C37d</code> v modulu <code>m37c_Atributy</code>	560
Výpis 37.9:	Práce s atributy instancí třídy <code>C37d</code>	561
Výpis 37.10:	Definice modulu <code>m37d_ModuleProperty</code> s vlastní mateřskou třídou	562
Výpis 37.11:	Import modulu <code>m37d_ModuleProperty</code> a práce s jeho vlastností	563
Výpis 37.12:	Ukázka používání slotů	565
Výpis 38.1:	Použití metody <code>__init_subclass__()</code>	567
Výpis 38.2:	Šablona testu funkce metatřídy v modulu <code>m38a_TestMetaXxx</code>	573
Výpis 38.3:	Import modulu <code>m38a_TestMetaXxx</code>	573
Výpis 38.4:	Definice metatřídy <code>MetaFce</code> definované jako funkce v modulu <code>m38b_MetaFce</code>	574
Výpis 38.5:	Import modulu <code>m38b_MetaFce</code>	575
Výpis 38.6:	Definice metatřídy <code>MetaCls</code> definované jako třída v modulu <code>m38c_MetaCls</code>	575
Výpis 38.7:	Import modulu <code>m38c_MetaCls</code>	576
Výpis 38.8:	Definice metatřídy <code>MetaObj</code> definované v modulu <code>m38d_MetaObj</code> jako objekt	577
Výpis 38.9:	Import modulu <code>m38d_MetaObj</code>	578
Výpis 38.10:	Definice jedináčka pomocí <code>__new__()</code>	579
Výpis 38.11:	Definice jedináčka pomocí metatřídy	580
Výpis 39.1:	Definice a použití jednoduchých korutin	584
Výpis 39.2:	Zabalení korutiny do instance třídy <code>Task</code>	585
Výpis 39.3:	Analýza datových typů	587

Seznam obrázků

Obrázek 1.1: Okno s dokumentací platformy	38
Obrázek 1.2: Okno příkazového řádku Windows se spuštěným interpretem Pythonu	41
Obrázek 1.3: Okno vývojového prostředí IDLE	43
Obrázek 1.4: Výběr požadovaného cílového okna v nabídce Windows	44
Obrázek 4.1: Pokus o výpis rozsáhlejší nápovědy v prostředí IDLE	82
Obrázek 5.1: Indexování jednotlivých znaků textového řetězce – stringu	94
Obrázek 8.1: Syntaktický diagram příkazu <code>assert</code>	128
Obrázek 9.1: Okno světa želvy po provedení zadaných příkazů	138
Obrázek 9.2: Syntaktický diagram možných verzí příkazu <code>import</code>	145
Obrázek 12.1: Syntaktický diagram hlavičky funkce	182
Obrázek 13.1: Význam nastavení Show Code Context v prostředí IDLE	197
Obrázek 14.1: Postup zpracování jednoduchého podmíněného příkazu	213
Obrázek 14.2: Vývojový diagram úplného podmíněného příkazu	214
Obrázek 14.3: Vývojový diagram zobrazující postup vykonávání rozšířeného podmíněného příkazu	215
Obrázek 15.1: Syntaktický diagram příkazu <code>for</code>	229
Obrázek 16.1: Syntaktický diagram příkazu <code>try</code>	245
Obrázek 16.2: Syntaktický diagram příkazu vyhození výjimky	251
Obrázek 17.1: Syntaxe generátorové notace seznamu	260
Obrázek 22.1: Syntaktický diagram formátovacího stringu	324
Obrázek 22.2: Syntaktický diagram nahrazovacího pole	325
Obrázek 22.3: Syntaktický diagram nahrazovaného textu	325
Obrázek 22.4: Syntaktický diagram specifikace formátu	327
Obrázek 23.1: Indexování jednotlivých znaků textového řetězce – stringu	349
Obrázek 25.1: Diamantový problém	385
Obrázek 28.1: Architektura, pro kterou není možné odvodit MRO	429
Obrázek 31.1: Strom složek s balíčky a moduly	467
Obrázek 31.2: Stromy složek s NS-balíčky	480
Obrázek 34.1: Syntaktický diagram příkazu <code>with</code>	520
Obrázek 37.1: Reakce IDLE na zápis tečky za odkazem na objekt	558

Seznam tabulek

Tabulka 5.1: Tabulka priorit operátorů (operátory, u nichž nejsou uvedeny operandy, jsou binární, infixové)	88
Tabulka 6.1: Klíčová slova jazyka Python	98
Tabulka 23.1: Tabulka zařazení jednotlivých probraných kontejnerů	346
Tabulka 23.2: Tabulka různých způsobů vykrajování	349
Tabulka 24.1: Důležité atributy a funkce modulu <code>os</code>	358
Tabulka 24.2: Probrané funkce modulu <code>os.path</code>	359
Tabulka 24.3: Režimy otevírání souboru	366
Tabulka 34.1: Tabulka přetížitelných binárních operátorů	513
Tabulka 34.2: Tabulka přetížitelných unárních operátorů	514
Tabulka 34.3: Tabulka zbylých emulačních funkcí	514
Tabulka 34.4: Tabulka konverzních funkcí	514

Seznam odboček – podšeděných bloků

Odbočka – podšeděný blok.....	33
Přísné a benevolentní programovací jazyky	51
Operátory jako funkční objekty	89
Statické a dynamické typování.....	97
Návrh podle kontraktu.....	128
Fyzické a logické řádky.....	132
Zásobník návratových adres – ZNA.....	222
Hešovatelné objekty	279
Předání argumentu odkazem.....	308
Mělké a hluboké kopie objektů.....	344
Terminologická vsuvka	378
Destruktor versus finalizér	510
Slabé odkazy (weak references).....	564