

Část E

Přílohy

Tato část není součástí tištěné knihy, aby zbytečně nezvyšovala její cenu. Je však součástí elektronických verzí a čtenáři tištěných knih si ji mohou stáhnout.

Příloha A	Instalace Pythonu pod Windows	400
A.1	Vlastní instalace	400
A.1.1	Uživatelská nastavení	401
A.1.2	Instalace	402
Příloha B	Konfigurace ve Windows	404
B.1	Definice substituovaných disků	404
B.2	Nastavování zástupce spouštějícího IDLE	405
Příloha C	Použité funkce ze standardní knihovny	407
C.1	Zabudované funkce	407
C.2	Speciální metody.....	411
C.3	Metody třídy <code>dir</code>	411
C.4	Metody posloupností – <code>Sequence</code>	411
C.5	Metody třídy <code>list</code>	412
C.6	Metody třídy <code>str</code>	412
Příloha D	Konvence pro psaní programů v Pythonu	413
D.1	Uspořádání kódu.....	413
D.2	Jmenné konvence	414
D.3	Dokumentační komentáře (PEP 257).....	415

Příloha A

Instalace Pythonu pod Windows

Jak bylo řečeno v pasáži [1.3.2 Instalace Pythonu](#) na straně [40](#), instalační soubory najdete na stránce <https://www.python.org/downloads/>, kde si zadáte, pro který operační systém chcete *Python* instalovat. Poté si stáhnete instalační soubor a spustíte jej.

Ve zbytku této přílohy najdete stručný popis instalace pro nejrozšířenější operační systémy *Windows*, který používá 92 % čtenářů mých knih.

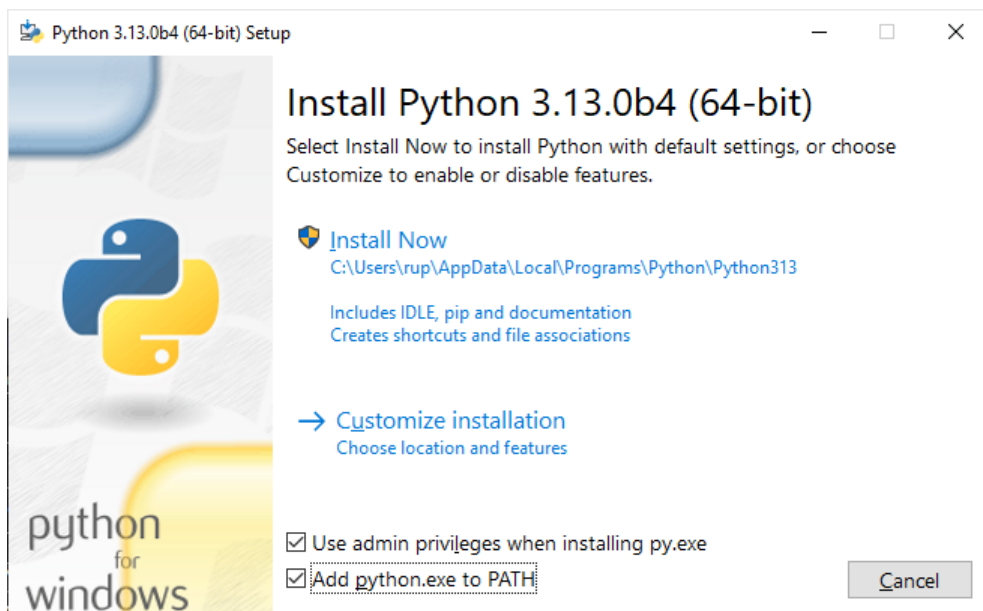
A.1 Vlastní instalace

Při instalaci na systémech *Windows* se vás instalační program nejprve zeptá, jestli chcete spustit instalaci s implicitním nastavením, anebo budete chtít její nastavení ovlivnit (viz obrázek [A.1](#)).

Pod heslem **Install now**, jehož zadáním zvolíte implicitní instalaci, se dozvíte složku, do níž bude *Python* instalován, a to, že součástí instalace bude vývojové prostředí *IDLE* (a s ním samozřejmě grafická knihovna), program **pip** a dokumentace. Současně se vytvoří zástupci a nastaví se spouštění interpretu při poklepání na soubor s příponou z definované sady.

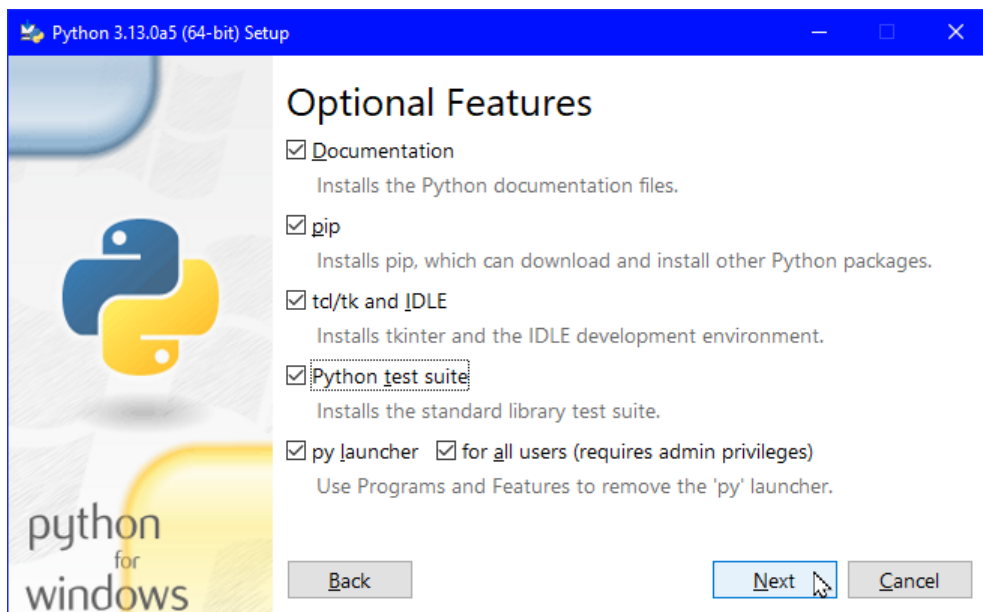
Jste-li opravdoví začátečníci, asi budete volit tuto možnost. Budete-li chtít některé z výše uvedených nastavení změnit, zvolte **Customize installation**, po níž se otevře okno s možnostmi podrobnějšího nastavení.

Před zadáním této volby byste ale měli zaškrtnout políčko **Add python.exe to PATH**, aby se cesta k programu dostala do seznamu prohledávaných cest, a mohli jste pak zadávat samotný název programu bez příslušné cesty k němu.



Obrázek A.1:
Úvodní instalační okno pro Windows

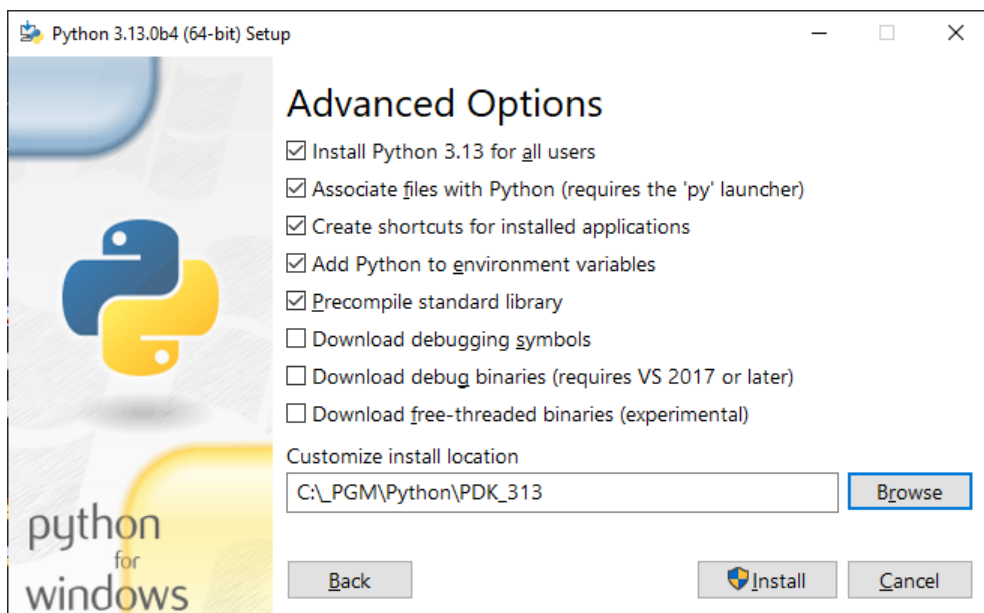
A.1.1 Uživatelská nastavení



Obrázek A.2:
Okno pro nastavení instalovaných součástí

Jak jsem řekl, po aktivaci uživatelského nastavení se otevře okno **Optional Features** (viz obrázek [A.2](#)), v němž budete moci zadat, které ze součástí budete, resp. nebudete chtít instalovat.

Po stisku tlačítka **Next** se otevře okno **Advanced options**, v němž můžete nastavit ještě některé pokročilé volby a především pak můžete v poli **Customize install location** zadat, kam chcete danou verzi *Pythonu* instalovat (viz obrázek [A.3](#)).



Obrázek A.3:

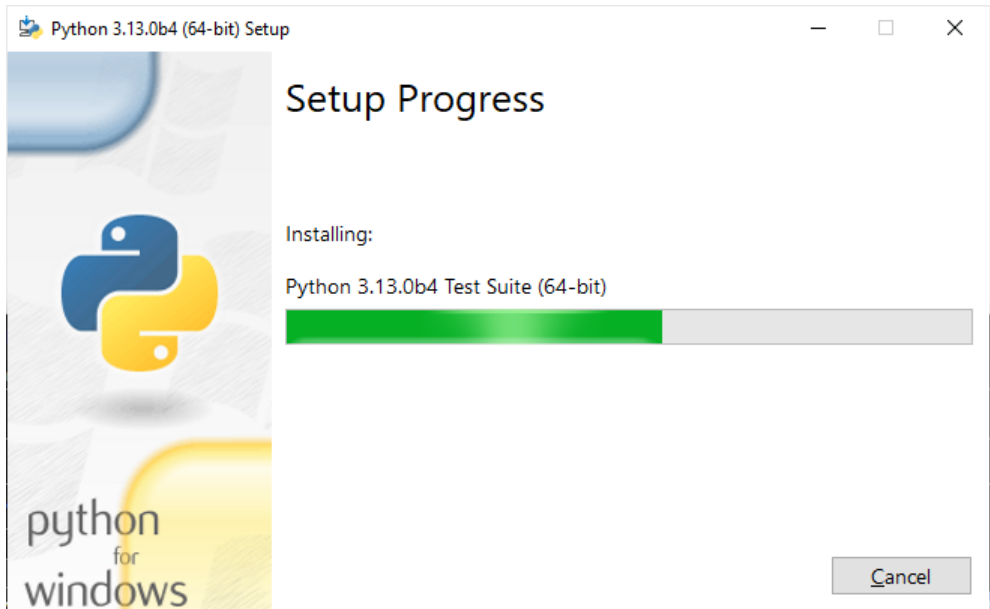
Okno pro nastavení pokročilejších voleb a umístění programu

Po nastavení všech voleb a případného vlastního umístění stisknete **Install**.

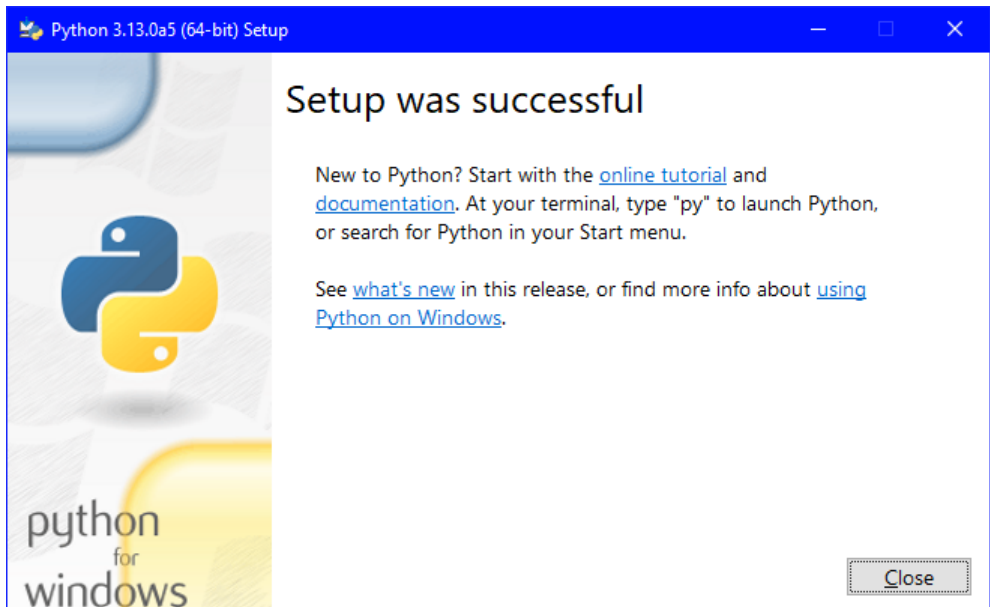
A.1.2 Instalace

Po spuštění instalace na obrazovce vše zhasne (používáte-li více displejů, zhasnou všechny) a systém otevře jediné okno, v němž se vás zeptá, jestli chcete spuštěné aplikaci povolit, aby prováděla změny ve vašem počítači. Samozřejmě jí to povolíte (jinak byste nic nenaistalovali), načež se na všechny displeje vrátí původní obsah a na hlavním se objeví okno **Setup Progress** (via obrázek [A.4](#)), které vás informuje o postupu instalace.

Po úspěšné instalaci se otevře okno **Setup was successful** (viz obrázek [A.5](#)), v němž vás instalační program informuje, že se vše podařilo, a nabídne odkazy na tutoriál, dokumentaci, informace o novinkách instalované verze a na stránku o rozšířených možnostech, určenou pro zkušenější uživatele s některými nestandardními nároky.



Obrázek A.4:
Indikace postupu instalace



Obrázek A.5:
Indikace postupu instalace

Příloha B

Konfigurace ve Windows



Co se v kapitole naučíte

Tato příloha vás seznámí s některými operacemi, které je vhodné provést při instalaci doprovodných programů v prostředí *Windows*.

B.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 **P:**, je vložit do složky, kterou budete chtít substituovat jako disk **P:**, 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 obyčejné 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 P: /d  
SUBST P: .
```

První příkaz má za úkol zrušit případnou doposud nastavenou substituci disku **P:** (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 **P:**.

Soubor umístíte do složky, z níž budete chtít vytvořit 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 **P:**, budete ve skutečnosti pracovat s obsahem substituované složky. A naopak: cokoliv uděláte s obsahem substituované složky, uděláte zároveň s obsahem disku **P:**.

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 substituovat 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.

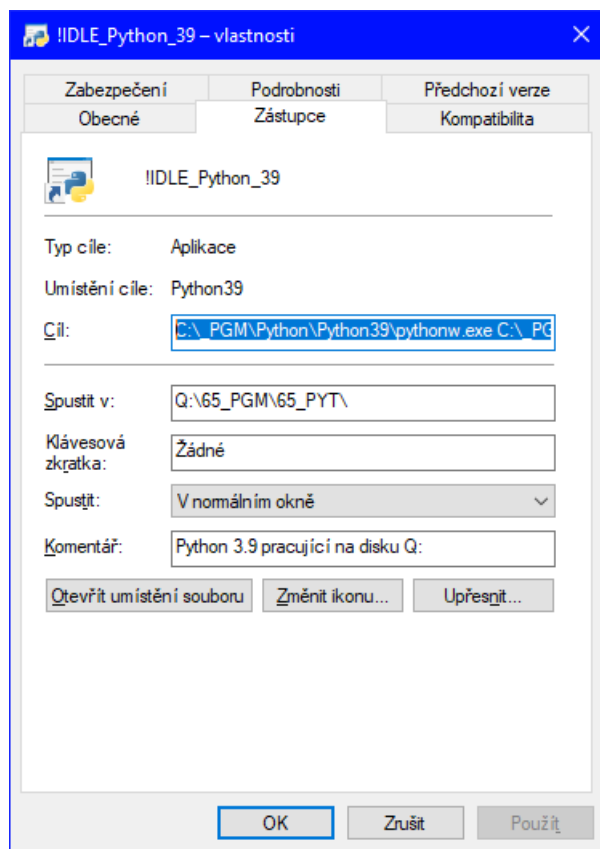
B.2 Nastavování zástupce spouštějícího IDLE

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

1. Otevřete ve svém oblíbeném správci souborů (budu předpokládat, že jím je *Průzkumník* nebo *Total Commander*) složku s doprovodnými programy **65_PYT**.
2. Klepněte na soubor **!IDLE_Python_39.lnk** (používáte-li *Průzkumník*, ten příponu zástupců, tj. *lnk*, nezobrazuje).
3. Stiskněte **ALT+ENTER**. Otevře se okno z obrázku [B.1](#).
4. V poli **Cíl** je zadáno:

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

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



Obrázek B.1:
Okno zástupce spouštějícího IDLE

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

5. 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.
6. V poli **Spustit v** je zadána cesta ke složce se zdrojovými soubory doprovodných skriptů. Tu si také upravte podle svého počítače.
7. Když máte vše korektně upraveno, uložte upraveného zástupce stiskem **OK**. Od této chvíle můžete IDLE spouštět poklepáním na daného zástupce.

Příloha C

Použité funkce ze standardní knihovny

C.1 Zabudované funkce

Ležaté hranaté závorky v následujících hlavičkách funkcí označují nepovinné součásti.

abs(x)

Vrátí absolutní hodnotu zadaného čísla (řádky 1 až 4).

```
>>> abs(-3.14)
3.14
>>> abs(1+1j)      #Absolutní hodnotou komplexního čísla je jeho velikost
1.4142135623730951
```

bool(x)

Vrátí logickou hodnotu, na kterou se daná hodnota v případě potřeby převede – viz např. výpis [10.1](#) na straně [146](#).

dict(x)

Vytvoří slovník z prvků dodaných generátorem zadaným v argumentu.

divmod(x, y, /)

Očekává jako argumenty dvě (ne komplexní) čísla a vrátí dvojici hodnot obsahující jejich celočíselný podíl a zbytek po dělení. Pro celočíselné argumenty platí, že výsledek je $(x//y, x\%y)$. Pro reálné argumenty je to trochu složitější – podrobnosti najdete v dokumentaci.

enumerate(iterable, start=0)

Očekává jako argument zdroj sekvenčních dat. Vrátí instanci třídy `enumerate` generující posloupnost dvojic, v nichž je nultým prvkem klíč a prvním prvkem příslušná hodnota.

eval(výraz[, globals[, locals]])

Vyhodnotí zadaný výraz a vrátí spočtenou hodnotu. Při vyhodnocení může používat zadané globální a lokální proměnné, které však musejí být zadány prostřednictvím slovníku.

float(x=0, /)

Vrátí reálné číslo vytvořené ze zadaného čísla či stringu. Je-li argumentem string, požaduje, aby vyhovoval pravidlům pro zápis reálného čísla, na něž jej převede. Jsou ale povoleny pouze úvodní a závěrečné bílé znaky.

frozenset(iterable/)

Vrátí nový objekt typu `frozenset`, volitelně s prvky převzatými z `iterable`. O typu `frozenset` jsme hovořili v podkapitole [12.3 Druhy kontejnerů](#) na straně [176](#).

hash(obj, /)

Vrátí heš-hodnotu objektu (má-li ji), což je celé číslo, které daný objekt do jisté míry charakterizuje. Podrobněji jsme o heš-kódu hovořili v pasáži [12.2.1 Hešovatelné objekty](#) na straně [175](#).

help(objekt/)

Funkce slouží k získání nápovědy. Protože je ale výklad rozsáhlejší, je mu věnována samostatná podkapitola [5.4 Získání nápovědy – dokumentace](#) na straně [84](#).

chr(i, /)

Očekává celé číslo a vrátí string tvořený znakem s daným kódem. Je to vlastně inverzní funkce k funkci `ord()`.

```
>>> chr(0x20ac) #Aby se znak zobrazil, musí být definován v použitém fontu
'€'
>>> chr(0x1D120) #Houslový klíč není v použitém fontu => zobrazil se jako kód
'\U0001d120'
```

id(object, /)

Vrátí celé číslo sloužící k identifikaci zadaného objektu. Toto číslo můžeme chápat jako virtuální adresu objektu. Objekty se stejným `id` považuje systém za totožné. Rozdílné objekty musí být na různých místech paměti, a proto musejí mít i rozdílná `id`.

input(/výzva/)

Může být volána buď bez argumentů, anebo s argumentem představujícím text výzvy. Vypíše na standardní výstup zadanou výzvu (nebyla-li zadána, nevypíše nic) a čeká, až za ni uživatel запиše to, k čemu byl vyzván, a stiskne ENTER. Vrátí uživatelem zadaný text.

```
>>> input("Zadej číslo: ")
Zadej číslo: 123
'123'
```

int(x=0, base=10, /)

Prvním argumentem je objekt, který lze převést na celé číslo. Je-li argumentem reálné číslo, vrátí celé číslo představující jeho celou část. Je-li argumentem string, požaduje, aby vyhovoval pravidlům pro zápis celého čísla v desítkové soustavě, na něž jej převede. Přitom jsou povoleny pouze úvodní a závěrečné bílé znaky. Není-li zadán argument, vrátí nulu.

```
>>> int(3.9) #Při převodu se nezaokrouhluje, ale usekává se desetinná část
3
>>> int(-3.9) #Stejně je tomu i při převodu záporného čísla
-3
```

Je-li zadán druhý argument, pak v prvním argumentu očekává string reprezentující celé číslo zadané v číselné soustavě definované druhým argumentem, který musí být z rozsahu $2 \leq \text{base} \leq 36$. V zadaném čísle jsou povolena zpráhledňovací podtržítka. Číselné soustavy se základem větším než 10 používají v roli dalších číslic písmena A-Z, resp. a-z. Velikost použitých písmen přitom nehraje roli.

len(obj, /)

Očekává objekt s více prvky a vrací počet těchto prvků. Je-li argumentem string, vrátí počet jeho znaků.

```
>>> len("")
0
>>> len('12345')
5
```

list(x)

Vytvoří seznam z prvků dodaných generátorem zadaným v argumentu. Podrobněji viz [12.4.2 Vytvoření prostřednictvím konstruktorů](#) na straně 178.

object()

Vytvoří novou instanci třídy `object`.

ord(c)

Při zadání řetězce představujícího jeden znak vrátí celé číslo představující kód zadaného znaku v sadě *Unicode*. Například `ord('a')` vrátí celé číslo 97 a `ord('€')` (znak eura) vrátí 8364. Je to inverzní funkce k funkci `chr()`.

print(argumenty)

Očekává seznam čárkami oddělených výrazů, jejichž hodnoty má vytisknout na standardní výstup. Podrobnosti viz [9.5 Funkce print\(\) a její parametry](#) na straně [138](#).

range(stop)**range(start, stop[, step])**

Očekává celočíselné argumenty a vrátí objekt, který se používá jako zdroj posloupnosti celých čísel začínající číslem `start`, pokračující čísla zvětšujícími se postupně o `step` a končící posledním číslem menším (při záporné hodnotě `step` větším) než `stop`.

Není-li zadán argument `step`, rostou čísla po jedné. Volání `range(stop)` je ekvivalentní volání `range(0, stop)`, respektive `range(0, stop, 1)`.

reload(/modul/); přesněji importlib.reload(/modul/)

Znovu načte zadaný modul. Podrobněji viz [6.7 Oprava načteného modulu](#) na straně [98](#).

repr(x)

Vrátí string představující systémový podpis svého argumentu.

set(x)

Vytvoří množinu prvků dodaných generátorem zadaným v argumentu. Podrobněji viz [12.4.2 Vytvoření prostřednictvím konstruktorů](#) na straně [178](#).

str(object='')

Převede zadaný argument na text představující jeho uživatelský podpis.

super(/type/, object-or-type//)

Vytvoří instanci třídy `super` schopnou delegovat požadavek na přístup k zadanému atributu na správného rodiče. Použijete-li bezparametrickou verzi, dodá potřebné argumenty překladač. Volání `super()` je pak ekvivalentní s voláním

```
super(type(self), self)
```

tuple(x)

Vytvoří n-tici prvků dodaných generátorem zadaným v argumentu. Podrobněji viz [12.4.2 Vytvoření prostřednictvím konstruktorů](#) na straně [178](#).

type(object)

Vrátí typ zadaného argumentu.

C.2 Speciální metody

`__init__()`

Initor instance zodpovídající za její inicializaci.

`__repr__()`

Vrátí systémový podpis své instance. Podrobnosti viz [17.1.10 Metody `repr` \(\) a `str` \(\)](#) na straně [230](#).

`__str__()`

Vrátí uživatelský podpis své instance. Podrobnosti viz [17.1.10 Metody `repr` \(\) a `str` \(\)](#) na straně [230](#).

C.3 Metody třídy `dir`

`fromkeys(iterable[, value]) -> dict`

Vytvoří slovník se zadanými klíči a jednotně inicializovanými hodnotami.

C.4 Metody posloupností – `Sequence`

Z probraných datových typů patří mezi posloupnosti instance tříd `list`, `tuple` a `str`, tj. seznamy, n-tice a stringy.

`index(x [, i [, j]]) -> int`

Vrátí index prvního výskytu argumentu `x` v zadané posloupnosti. Je-li zadán argument `i`, začne se hledat od tohoto indexu, je-li zadán argument `j`, přestane se hledat před tímto indexem. Pokud hledanou položku v zadané oblasti nenajde, vyvolá výjimku `ValueError`.

`sorted(x) -> list`

Vrátí nově vytvořený seznam, obsahující seřazená data dodaná zdrojem předaným jako argument. Podrobněji je funkce rozebrána v publikacích [\[14\]](#) a [\[15\]](#).

C.5 Metody třídy `list`

`append(x) -> None`

Přidá na konec seznamu svůj argument.

`sort(*, key=None, reverse=False) -> None`

Seřadí prvky v seznamu podle jejich velikosti. Prvky ale musejí být vzájemně porovnatelné. Příklad použití je na řádce 4 ve výpisu [26.2](#) na straně [331](#).

C.6 Metody třídy `str`

`lower() -> str`

Vrátí kopii svého stringu převedenou na malá písmena podle standardu *Unicode*. Příklad použití je na řádcích [18-19](#) ve výpisu [25.4](#) na straně [324](#).

`split(sep=None, maxsplit=-1) -> list[str, ...]`

Vrátí seznam stringů, z nichž je tvořen oslovený string. Tyto stringy jsou v něm odděleny stringem zadaným v argumentu `sep`. Není-li tento argument zadán, považuje se za oddělovač posloupnost bílých znaků.

Je-li zadán argument `maxsplit`, bude vrácen seznam o nejvýše `maxsplit+1` prvcích. Příklad použití je na řádce 6 ve výpisu [26.4](#) na straně [332](#).

`upper() -> str`

Vrátí kopii svého stringu převedenou na velká písmena podle standardu *Unicode*.

`strip([chars]) -> str`

Vrátí kopii svého stringu s odstraněnými úvodními a závěrečnými znaky vyskytujícími se v zadaném argumentu. Není-li zadán argument, odstraní úvodní a závěrečné bílé znaky. Příklad použití je na řádcích [15-16](#) ve výpisu [25.4](#) na straně [324](#).

Příloha D

Konvence pro psaní programů v Pythonu

Oficiální konvence pro psaní kódu jsou včetně demonstračních ukázek podrobně popsány v dokumentu [PEP 8](https://www.python.org/dev/peps/pep-0008/)⁴¹ na adrese <https://www.python.org/dev/peps/pep-0008/>. Konvence pro psaní dokumentačních komentářů jsou v dokumentu [PEP 257](https://www.python.org/dev/peps/pep-0257/), který najdete na adrese <https://www.python.org/dev/peps/pep-0257/>. 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.

D.1 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
```

⁴¹ 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/>.

```
+ druhá_proměnná)
```

- Definice tříd a samostatných funkcí oddělujte dvěma řádky, definice metod uvnitř třídy oddělujte 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 v seznamu prvků závěrečnou čárku – může usnadnit přidání dalšího prvku – např.

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

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

D.2 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), `o` (velké o) nebo `I` (velké i). Některé fonty neumí odlišit znaky `l-l-I` (jedna – L – I) a `o-o` (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.

D.3 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 z něj exportované funkce a třídy.
- 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 [9.2](#) na straně [131](#). 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.

Část F

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	418
Seznam obrázků	423
Seznam tabulek	425
Seznam odboček – podšeděných bloků	426

Seznam výpisů programů

Výpis 1.1:	Komunikace s interpretem spuštěným v konzolovém okně	45
Výpis 2.1:	Komunikace s interpretem spuštěným v konzolovém okně	54
Výpis 2.2:	Komunikace s interpretem v příkazovém okně IDLE	54
Výpis 3.1:	Počáteční mezery a komentáře	55
Výpis 3.2:	Možnosti zápisu celých čísel	56
Výpis 3.3:	Zadávání a zobrazování reálných čísel	57
Výpis 3.4:	Zápis jednořádkového textu	58
Výpis 3.5:	Ve stringu představuje znak # sám sebe	59
Výpis 3.6:	Zápis víceřádkového textu	59
Výpis 3.7:	Definice a použití proměnných	64
Výpis 3.8:	Současné přiřazení skupiny hodnot proměnným	64
Výpis 3.9:	Nebezpečné změny hodnot	67
Výpis 4.1:	Dvě možnosti přiřazení více hodnot	70
Výpis 4.2:	Použití funkcí	71
Výpis 4.3:	Hodnota None a její zobrazení	72
Výpis 4.4:	Reakce na použití objektu ... (Ellipsis)	74
Výpis 4.5:	Použití formátovaných stringů	75
Výpis 4.6:	Více příkazů versus více výrazů na jednom řádku	76
Výpis 4.7:	Složené přiřazovací operátory	78
Výpis 4.8:	Ukázka použití funkce input() a implicitní proměnné	78
Výpis 5.1:	Typy objektů, s nimiž jsme se doposud seznámili	81
Výpis 5.2:	Demonstrace práce s atributy	82
Výpis 5.3:	Získání nápovědy k zadané funkci	84
Výpis 5.4:	Zadání argumentu nápovědy pomocí stringu	86
Výpis 5.5:	Přepnutí do režimu nápovědy a zpět	87
Výpis 6.1:	Importování modulů	91
Výpis 6.2:	Uložení odkazu na modul pod jiným názvem	93
Výpis 6.3:	Import zadaných objektů ze zadaného modulu	94
Výpis 6.4:	Modul m06a_module_demo sloužící k demonstraci chování Pythonu při zavádění modulu	96
Výpis 6.5:	Import a použití modulu m06a_module_demo	98
Výpis 6.6:	Upravené řádky v modulu m06a_module_demo	99
Výpis 6.7:	Načtení upravené verze modulu m06a_module_demo	99
Výpis 7.1:	Initor balíčku modules.m07a_pkg	104
Výpis 7.2:	Initor balíčku modules.m07a_pkg.sub_pkg	105
Výpis 7.3:	Modul modules.m07a_pkg.module_1	105
Výpis 7.4:	Modul modules.m07a_pkg.sub_pkg.module_2	105
Výpis 7.5:	Import modulu modules.m07a_pkg.sub_pkg.module_2	106
Výpis 7.6:	Zjištění aktuální pracovní složky a její změna	107

Výpis 7.7:	Přímé spuštění modulu <code>modules.m07a_pkg.module_1</code> z příkazového řádku	108
Výpis 8.1:	Začlenění archivu <code>Karelcz73.pyz</code> mezi prohledávané	115
Výpis 8.2:	Vytvoření světa s definovanými počty značek a umístění robota	118
Výpis 8.3:	Ovládání vytvořeného robota	120
Výpis 8.4:	Použití některých zjišťovacích funkcí	122
Výpis 8.5:	Demonstrace vlivu skrývání	123
Výpis 8.6:	Ukázka použití příkazu <code>with</code>	126
Výpis 9.1:	Možné podoby definice prázdné funkce	130
Výpis 9.2:	Získávání nápovědy	131
Výpis 9.3:	Definice funkce <code>demo()</code> a její použití	132
Výpis 9.4:	Definice funkce <code>mocnina()</code>	134
Výpis 9.5:	Definice a použití funkce <code>counter()</code> tisknoucí, kolikrát byla zavolána	135
Výpis 9.6:	Definice funkce <code>pozdrav()</code>	135
Výpis 9.7:	Definice funkce <code>div_mod()</code>	136
Výpis 9.8:	Definice a použití funkce <code>bmí</code>	136
Výpis 9.9:	Používání pozičních a pojmenovaných argumentů	137
Výpis 9.10:	Definice funkce demonstrující použití anotací	141
Výpis 9.11:	Definice a test funkcí <code>turn_right()</code> a <code>step_left()</code>	142
Výpis 9.12:	Definice a použití lambda-výrazu	143
Výpis 10.1:	Převod libovolné hodnoty na logickou	146
Výpis 10.2:	Porovnání hodnot versus porovnání objektů	148
Výpis 10.3:	Chování logických operátorů	150
Výpis 10.4:	Funkce <code>free_before()</code> zjišťující, zda je před robotem volno	150
Výpis 10.5:	Podmíněný výraz a jeho použití	151
Výpis 10.6:	Vnořený podmíněný výraz	152
Výpis 10.7:	Jednoduchý podmíněný příkaz a jeho použití	153
Výpis 10.8:	Definice a test funkce <code>are_2_markers()</code> využívající zanořování bloků	154
Výpis 10.9:	Ukázka použití úplného podmíněného příkazu	155
Výpis 10.10:	Ukázka použití rozšířeného podmíněného příkazu	156
Výpis 10.11:	Ukázka použití přepínacího příkazu (přepínače)	157
Výpis 11.1:	Definice funkce <code>to_wall()</code> využívající cyklus <code>while</code>	160
Výpis 11.2:	Definice funkce <code>find_treasure()</code> hledající s robotem poklad ze značek	161
Výpis 11.3:	Definice funkce <code>markers_2_wall()</code> využívající cyklus s podmínkou uprostřed	163
Výpis 11.4:	Definice funkce <code>goto_next_marker()</code> demonstrující použití cyklu s koncovou podmínkou	164
Výpis 11.5:	String použitý jako objekt a jako zdroj hodnot	165
Výpis 11.6:	Posloupnosti reprezentované objekty typu <code>range</code>	166
Výpis 11.7:	Posloupnosti reprezentované objekty typu <code>enumerate</code>	166
Výpis 11.8:	Definice funkce <code>put_n()</code> demonstrující použití jednoduchého cyklu <code>for</code>	167
Výpis 11.9:	Definice funkce <code>put_n_beside()</code> demonstrující možnost zadání potřebných funkcí pomocí proměnných	167
Výpis 11.10:	Definice a spuštění funkce <code>test_put_n()</code> testující obě výše definované funkce	168
Výpis 11.11:	Definice funkce <code>sum3xC()</code> demonstrující použití příkazu <code>continue</code>	169
Výpis 11.12:	Definice funkce <code>factorial()</code> demonstrující použití rekurze	171
Výpis 11.13:	Definice funkce <code>put_markers_at_wall()</code> řešené pomocí rekurze	171
Výpis 12.1:	Vytváření kontejnerů prostřednictvím literálů	178
Výpis 12.2:	Vytváření kontejnerů prostřednictvím konstruktorů	179
Výpis 12.3:	Vytváření kontejnerů prostřednictvím generátorové notace	181

Výpis 12.4:	Funkce <code>mocniny()</code> vracející generátor pro tvorbu kontejnerů.....	182
Výpis 13.1:	Vytvoření pomocných kontejnerů pro následující demonstrace.....	185
Výpis 13.2:	Získání prvku z kontejneru pomocí indexu.....	185
Výpis 13.3:	Procházení kontejnerem pomocí příkazu <code>for</code>	187
Výpis 13.4:	Cyklus <code>for</code> s více parametry.....	188
Výpis 13.5:	Pohledy a jejich reflexe změn v „pozorovaném“ slovníku	189
Výpis 13.6:	Test přítomnosti prvku v kontejneru	191
Výpis 13.7:	Přidání a odebrání prvků kontejnerů	191
Výpis 13.8:	Definice a použití hvězdičkového parametru	193
Výpis 13.9:	Použití hvězdičkového argumentu	194
Výpis 13.10:	Definice a použití dvouhvězdičkového parametru.....	195
Výpis 13.11:	Použití dvouhvězdičkového argumentu	196
Výpis 14.1:	Nástin syntaxe příkazů pro zachycení a ošetření výjimky.....	201
Výpis 14.2:	Definice modulu <code>m14a_demo_vyjimky</code> sloužícího k demonstraci zachycení a ošetření výjimky	202
Výpis 14.3:	Záznam importu modulu <code>m14a_demo_vyjimky</code> a volání jeho funkcí.....	204
Výpis 14.4:	Definice funkce <code>factorial()</code> demonstrující použití dekorátoru <code>dbg.prSEd()</code> a kontrolních tisků	208
Výpis 15.1:	Definice modulu <code>m15a_script</code> v balíčku <code>modules</code>	211
Výpis 15.2:	Reakce na import modulu <code>modules.m15a_script</code> v interaktivním režimu	212
Výpis 15.3:	Reakce na spuštění modulu <code>modules.m15a_script</code> v příkazovém okně Windows.....	212
Výpis 15.4:	Definice těla modulu <code>solution</code>	213
Výpis 15.5:	Vytvoření samostatné aplikace pomocí modulu <code>zipapp</code> a její spuštění.....	215
Výpis 15.6:	Definice modulu <code>__main__</code> v souboru <code>Demo_application.pyz</code>	216
Výpis 17.1:	Definice prázdné třídy	226
Výpis 17.2:	Definice demonstrační třídy <code>DemoClass</code>	227
Výpis 17.3:	Vytvoření instancí třídy <code>DemoClass</code> a práce s nimi	232
Výpis 17.4:	Definice třídy <code>Fract</code> v modulu <code>m17b_Fract</code> v balíčku <code>modules</code>	234
Výpis 17.5:	Ukázka práce s instancemi třídy <code>Fraction</code>	234
Výpis 17.6:	Výsek z definice modulu <code>karelcz</code> s převodem instančních metod robota Karla na funkce.....	235
Výpis 17.7:	Ovládání robota Karla pomocí funkcí a instančních metod.....	236
Výpis 17.8:	Přidání nových instančních metod a jejich použití	237
Výpis 19.1:	Definice tříd <code>Matka</code> a <code>Dcera</code> v modulu <code>m19a_MatkaDcera</code>	249
Výpis 19.2:	Záznam vytvoření instancí tříd <code>Matka</code> a <code>Dcera</code> a volání jejich metod	250
Výpis 19.3:	Definice třídy <code>KarelRCD</code> v modulu <code>m19b_KarelRCD</code>	251
Výpis 19.4:	Definice tříd <code>Přítel</code> a <code>VnučkaDP</code> v modulu <code>m19c_PřítelVnučka</code>	254
Výpis 19.5:	Záznam vytvoření instancí tříd <code>VnučkaDP</code> a <code>VnučkaPD</code> z modulu <code>m19c_PřítelVnučka</code>	255
Výpis 20.1:	AHA-příklad s definicí hierarchie abstraktních a konkrétních tříd	261
Výpis 20.2:	Práce s výše definovanými třídami	261
Výpis 20.3:	Definice protokolu <code>IKarel</code> v initoru balíčku <code>karelcz</code>	264
Výpis 20.4:	Definice třídy <code>KarelC</code> implementující protokol <code>IKarel</code> a dekorující robota schopností měnit barvu	265
Výpis 20.5:	Test funkcionalit třídy <code>KarelC</code> a jejich instancí.....	266
Výpis 21.1:	Viditelnost globálních a třídních atributů	270
Výpis 21.2:	Použití příkazu <code>global</code>	271

Výpis 21.3:	Chyba při volání funkce <code>fceg()</code> s odkomentovaným příkazem na řádce 13	272
Výpis 21.4:	Definice těla modulu <code>m21a_underscore</code> označující neveřejné atributy počátečním podtržítkem	274
Výpis 21.5:	Definice těla modulu <code>m21b_all</code> uvádějící názvy veřejných atributů v posloupnosti <code>__all__</code>	274
Výpis 21.6:	Záznam konverzace s interpretem, při níž byly importovány oba moduly	275
Výpis 21.7:	Zadání zjišťovací vlastnosti pomocí dekorátoru	277
Výpis 21.8:	Prvky standardních n-tic nemají jména	279
Výpis 21.9:	Vytváření pojmenovaných n-tic	280
Výpis 21.10:	Definice třídy <code>Direction4</code> z balíčku <code>karelcz</code>	281
Výpis 23.1:	Aktuální podoba definice modulu <code>world</code>	306
Výpis 24.1:	Definice kroků scénáře v modulu <code>tests</code>	313
Výpis 24.2:	Podpis instancí třídy <code>ScenarioStep</code> definované podle výpisu 24.1	314
Výpis 24.3:	Změna systémového podpisu instancí třídy <code>ScenarioStep</code>	315
Výpis 24.4:	Podpis kroku scénáře po změně metody <code>__repr__()</code>	315
Výpis 24.5:	Definice počátku n-tice reprezentující „šťastný scénář“ vytvářené hry	316
Výpis 24.6:	Definice funkce <code>simulate_simple()</code> v modulu <code>tests</code>	317
Výpis 24.7:	Spuštění metody <code>simulate_simple()</code>	317
Výpis 24.8:	Definice funkce <code>simulate_with_state()</code> v modulu <code>tests</code>	318
Výpis 24.9:	Začátek simulace vypsání metodou <code>simulate_with_state()</code>	318
Výpis 25.1:	Prozatímní definice funkce <code>execute_command()</code> v modulu <code>__init__</code>	321
Výpis 25.2:	Předběžná definice testu funkce hry podle scénáře	321
Výpis 25.3:	Upravená definice třídy <code>Place</code> v modulu <code>world</code>	323
Výpis 25.4:	Definice funkce <code>test_by_scenario()</code> v modulu <code>tests</code>	324
Výpis 25.5:	Definice metody <code>ERROR()</code> v modulu <code>tests</code>	326
Výpis 25.6:	Tělo modulu <code>m25a_test_game</code> spouštějícího test hry	327
Výpis 25.7:	Tisky vypisované po spuštění modulu <code>m25a_test_game</code>	327
Výpis 26.1:	Definice upravené metody <code>execute_command()</code> v modulu <code>__init__</code>	330
Výpis 26.2:	Definice funkce <code>execute_command()</code> v modulu <code>actions</code>	331
Výpis 26.3:	Definice metody <code>_execute_empty_command()</code> v modulu <code>actions</code>	332
Výpis 26.4:	Prozatímní definice funkce <code>_execute_standard_command()</code> v modulu <code>actions</code>	332
Výpis 26.5:	Import modulu <code>m25a_test_game</code> v interaktivním režimu	333
Výpis 26.6:	Definice třídy <code>ANamed</code> v modulu <code>world</code>	335
Výpis 26.7:	Upravená definice třídy <code>Place</code>	337
Výpis 26.8:	Definice globálního slovníku <code>NAME_2_PLACE</code>	338
Výpis 26.9:	Definice instanční metody <code>initialize()</code> třídy <code>Place</code>	339
Výpis 26.10:	Definice funkce <code>initialize()</code> v modulu <code>world</code>	339
Výpis 26.11:	Aktualizovaná definice třídy <code>BAG</code> v modulu <code>world</code>	340
Výpis 26.12:	Zpráva testu <code>m25a_test_game</code> bez úvodních informací o průběhu importů	341
Výpis 27.1:	Nová definice funkce <code>_execute_standard_command()</code> v modulu <code>actions</code>	343
Výpis 27.2:	Definice třídy <code>Action</code> v modulu <code>actions</code>	344
Výpis 27.3:	Zástupné definice funkcí realizujících kód akcí v modulu <code>actions</code>	345
Výpis 27.4:	Definice globálního slovníku <code>NAME_2_ACTION</code> v modulu <code>actions</code>	345
Výpis 27.5:	Závěr zprávy o průběhu testu	345
Výpis 27.6:	Definice třídy <code>ItemContainer</code> v modulu <code>world</code>	347
Výpis 27.7:	Upravená definice initoru třídy <code>ANamed</code> v modulu <code>world</code>	347

Výpis 27.8:	Upravené definice třídy <code>Place</code> v modulu <code>world</code>	348
Výpis 27.9:	Upravená definice třídy <code>Bag</code> v modulu <code>world</code>	349
Výpis 27.10:	Upravená definice funkce <code>_take()</code> v modulu <code>action</code>	349
Výpis 27.11:	Konec zprávy testu po definici funkce <code>_take()</code>	350
Výpis 27.12:	Upravená definice funkce <code>_goto()</code> v modulu <code>action</code>	350
Výpis 27.13:	Konec zprávy testu po definici funkce <code>_goto()</code>	350
Výpis 27.14:	Upravená definice funkce <code>_put()</code> v modulu <code>action</code>	351
Výpis 27.15:	Konec zprávy testu po definici funkce <code>_put()</code>	351
Výpis 28.1:	Krok testující špatně zadané odstartování hry	355
Výpis 28.2:	Začátek definice chybového scénáře v modulu <code>tests</code>	355
Výpis 28.3:	Definice těla modulu <code>m28a_2_tests</code> sloužícího jako nový spouštěč testů.....	356
Výpis 28.4:	Konec zprávy testu po spuštění (nebo importu) modulu <code>m28a_2_tests</code>	356
Výpis 28.5:	Upravená definice třídy <code>Item</code> v modulu <code>world</code>	360
Výpis 28.6:	Definice metody <code>initialize()</code> ve třídě <code>Bag</code>	362
Výpis 28.7:	Výsledná podoba funkce <code>_take()</code> v modulu <code>actions</code>	362
Výpis 28.8:	Definice funkce <code>help</code> v modulu <code>actions</code>	363
Výpis 29.1:	Definice funkce <code>run()</code> v modulu <code>__init__</code>	365
Výpis 29.2:	Spuštění hry v interaktivním režimu	366
Výpis 29.3:	Definice funkce <code>multirun()</code> v modulu <code>__init__</code>	367
Výpis 29.4:	Definice funkce <code>current_state()</code> v modulu <code>__init__</code>	368
Výpis 29.5:	Upravená definice funkce <code>execute_command()</code> v modulu <code>__init__</code>	369
Výpis 29.6:	Definice funkcí <code>main()</code> a <code>help()</code> v modulu <code>__init__</code>	369
Výpis 30.1:	Definice protokolu <code>IUI</code> v modulu <code>__init__</code>	373
Výpis 30.2:	Upravené definice funkcí <code>run()</code> a <code>multirun()</code> v modulu <code>__init__</code>	373
Výpis 30.3:	Definice třídy <code>Console</code> v modulu <code>interfaces</code>	374
Výpis 30.4:	Aktivace třídy <code>Console</code> v modulu <code>__init__</code>	375
Výpis 30.5:	Definice třídy <code>PrimitiveGUI</code> v modulu <code>interfaces</code>	379
Výpis 30.6:	Doplnění metody <code>main()</code> v modulu <code>__init__</code>	380
Výpis 30.7:	Spuštění hry v grafickém režimu	381
Výpis 30.8:	Definice modulu <code>__main__</code> v kořenovém balíčku vytvářené aplikace	381
Výpis 31.1:	Definice předpon a přípon v názvech textových konstant.....	385
Výpis 31.2:	Začátek šťastného scénáře v balíčku <code>v1h_texts</code>	386

Seznam obrázků

Obrázek 1.1: Okno otevřené po poklepaní na skript <code>m01a_script.py</code>	42
Obrázek 1.2: Spouštění skriptu <code>m01a_script.py</code> v konzolovém okně	42
Obrázek 1.3: Okno s dokumentací platformy	43
Obrázek 1.4: Spuštění interpretu z příkazového řádku Windows	44
Obrázek 2.1: Okno průzkumníka Windows se zadáním příkazu spouštějícího IDLE	47
Obrázek 2.2: Okno vývojového prostředí IDLE se zadáním z výpisu 1.1	49
Obrázek 2.3: Okno vývojového prostředí IDLE	49
Obrázek 5.1: Pokus o výpis rozsáhlejší nápovědy v prostředí IDLE	85
Obrázek 6.1: Okno světa želvy po provedení zadaných příkazů	92
Obrázek 6.2: Okno světa želvy po provedení zadaných příkazů	95
Obrázek 7.1: Demonstrační hierarchie balíčků	104
Obrázek 8.1: Závěrečné podoba světa robota Karla vytvořeného programem z výpisu 8.2	118
Obrázek 8.2: Svět robota po provedení akcí z výpisu 8.3	121
Obrázek 8.3: Okno s chybovým hlášením po provedení akcí z výpisu 8.3	121
Obrázek 10.1: Postup zpracování jednoduchého podmíněného příkazu	152
Obrázek 10.2: Vývojový diagram úplného podmíněného příkazu	155
Obrázek 10.3: Vývojový diagram zobrazující postup vykonávání rozšířeného podmíněného příkazu	155
Obrázek 11.1: Bludiště, v němž robot hledá poklad	162
Obrázek 13.1: Indexování jednotlivých znaků textového řetězce – stringu	190
Obrázek 15.1: Sestava na disku	214
Obrázek 19.1: Diamantový problém	253
Obrázek 22.1: Prvky používané při tvorbě UML diagramů	294
Obrázek 23.1: Diagram tříd aktuálního stavu aplikace zkopírovaného do balíčku <code>game_ver.v1a_basic_architecture</code>	307
Obrázek 24.1: Diagram tříd aktuálního stavu aplikace zkopírovaného do balíčku <code>game_ver.v1a_basic_architecture</code>	319
Obrázek 25.1: Diagram tříd aktuálního stavu aplikace zkopírovaného do balíčku <code>game_ver.v1b_test</code>	328
Obrázek 26.1: Diagram tříd aktuálního stavu aplikace zkopírovaného do balíčku <code>game_ver.v1c_world</code>	341
Obrázek 27.1: Diagram tříd aktuálního stavu aplikace zkopírovaného do balíčku <code>game_ver.v1d_actions</code>	352
Obrázek 28.1: Diagram tříd aktuálního stavu aplikace zkopírovaného do balíčku <code>game_ver.v1e_robust</code>	364
Obrázek 29.1: Diagram tříd aktuálního stavu aplikace zkopírovaného do balíčku <code>game_ver.v1f_application</code>	371
Obrázek 30.1: Dialogové okno s výzvou k zadání údajů	378
Obrázek 30.2: Upozornění na chybné zadání číselné hodnoty	378

Obrázek 30.3: Diagram tříd aktuálního stavu aplikace zkopírovaného do balíčku

game_ver.v1g_gui.....382

Seznam tabulek

Tabulka 3.1: Klíčová slova jazyka Python.....	62
Tabulka 13.1: Tabulka různých způsobů vykrajování.....	190
Tabulka 22.1: Porovnání metody shora dolů a zdola nahoru	292

Seznam odboček – podšeděných bloků

Odbočka – podšeděný blok	30
Historie robota Karla	114
Zvláštnosti programových kontejnerů	174
Co to je h-objekt.....	298
Návrhový vzor Fasáda.....	376