

knihovna programátora

- Učebnice jazyka i referenční příručka v jednom
- Probírá konstrukce nevyžadující hlubší znalost objektového programování
- Výklad většiny konstrukcí doplňují syntaktické diagramy
- Použití konstrukcí je předvedeno na příkladech
- Rozbor příčin častých chyb a jejich řešení
- Na knihu navazuje další titul věnovaný objektovým konstrukcím



Rudolf Pecinovský

Python 3.14

ALGORITMICKÉ KONSTRUKCE



knihovna programátora

RUDOLF PECINOVSKÝ

Python 3.14

ALGORITMICKÉ KONSTRUKCE

GRADA
Publishing

Rudolf Pecinovský

Python 3.14

Algoritmické konstrukce

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
obchod@grada.cz, www.grada.cz
tel.: +420 234 264 401
jako svou 10344. publikaci

Odpovědný redaktor Petr Somogyi
Fotografie na obálce Depositphotos/mario7
Grafická úprava a sazba Rudolf Pecinovský
Počet stran 496
První vydání, Praha 2025
Vytisky Tiskárny Havlíčkův Brod a.s.

© Grada Publishing, a.s., 2025
Cover Design © Grada Publishing, a. s., 2025
Cover Photo © Depositphotos/mario7

*Upozornění pro čtenáře a uživatele této knihy:
Všechna práva vyhrazena. Žádná část této tištěné či elektronické knihy
nesmí být reprodukována a šířena v papírové, elektronické či jiné podobě
bez předchozího písemného souhlasu nakladatele.
Neoprávněné užití této knihy bude trestně stíháno.
Názvy produktů, firem apod. použité v knize mohou být ochrannými známkami
nebo registrovanými ochrannými známkami příslušných vlastníků.
Automatizovaná analýza textů nebo dat ve smyslu čl. 4 směrnice 2019/790/EU
a použití této knihy k trénování AI jsou bez souhlasu nositele práv zakázány.*

ISBN 978–80–271–8342–5 (ePub)
ISBN 978–80–271–8341–8 (pdf)
ISBN 978–80–271– 6061–7 (print)

*Mé ženě Jarušce a dětem
Štěpánce, Pavlínce, Ivance a Michalovi*

Stručný obsah

Úvod	21
------------	----

Část A Superzáklady	31
----------------------------	-----------

Kapitola 1 Startujeme	32
Kapitola 2 Zadávání jednoduchých hodnot	57
Kapitola 3 Zadávání textů – stringů	73
Kapitola 4 Volání funkcí	90
Kapitola 5 Jednoduché výrazy	109
Kapitola 6 Proměnné, výrazy, příkazy	123
Kapitola 7 Logické hodnoty a operace	145
Kapitola 8 Další jednoduché příkazy	157

Část B Složené příkazy	167
-------------------------------	------------

Kapitola 9 Moduly	168
Kapitola 10 Vytvoření vlastního modulu	185
Kapitola 11 Balíčky	206
Kapitola 12 Definice funkcí	221
Kapitola 13 Parametry, argumenty a lokální proměnné funkcí	234
Kapitola 14 Pokročilé rysy funkcí	255
Kapitola 15 Rozhodování a příkaz with	276
Kapitola 16 Opakování kódu – rekurze a cykly	289
Kapitola 17 Tvorba aplikací	309
Kapitola 18 Ošetřování chyb	323

Část C Kontejnery	347
Kapitola 19 Seznamy.....	348
Kapitola 20 N-tice	367
Kapitola 21 Množiny.....	382
Kapitola 22 Slovníky	395
Kapitola 23 Rozšíření definic funkcí	409
Kapitola 24 Formátování stringů	428
Kapitola 25 Operace s kontejnery.....	453
Kapitola 26 Práce se soubory	464
Literatura.....	487
Rejstřík	489

Podrobný obsah

Úvod	21
Komu kniha není určena	21
Komu je kniha určena	22
Struktura příručky	22
Koncepte výkladu	24
Jazyk identifikátorů	25
Potřebné vybavení	25
Operační systém	25
Doprovodné programy a jejich organizace	25
Použité typografické konvence	27
Odbočka – podšeděný blok	29
Zpětná vazba	29
Část A Superzáklady	31
Kapitola 1 Startujeme	32
1.1 Hlavní součást instalace	32
1.1.1 Platforma	32
1.1.2 Dokumentace	33
1.1.3 PEP	34
1.1.4 Pracovní režimy	34
1.1.5 Překladač versus interpret	35
1.2 Vývojová prostředí	35
1.2.1 PyCharm a IntelliJ IDEA	35
1.2.2 Visual Studio Code	36
1.2.3 Jupyter Notebook a JupyterLab	36
1.2.4 Řádkový editor a IDLE	37
1.3 Spuštění Pythonu	37
1.3.1 Spuštění ve Windows	37
1.3.2 Spuštění v macOS a Linuxu	39
1.3.3 Co je nejdůležitější	39
1.3.4 Spuštění Pythonu v demonstračních programech v knize	39
1.4 Řádkový editor – REPL	39
1.4.1 Komentáře	40
1.4.2 Odsazování	41
1.4.3 Víceřádkové příkazy	41
Fyzické a logické řádky	42
1.4.4 Dvě verze řádkového editoru	42
1.5 Spuštění skriptu napsaného v Pythonu	48
1.5.1 Windows	49
1.6 Prostředí IDLE	49
1.6.1 Spuštění IDLE	49
1.6.2 Základní popis	50
1.6.3 Příkazové okno	51

1.6.4	Editační okno	53
1.6.5	Umístění editovaných souborů	54
1.6.6	Problémy při práci se schránkou	54
1.6.7	Barevné zvýraznění textu	55
1.6.8	Pokročilá nastavení	55
1.7	Shrnutí kapitoly a doprovodné soubory	56
Kapitola 2	Zadávání jednoduchých hodnot	57
2.1	Objekty a objektové programování	58
2.1.1	Koncepce Pythonu	58
2.1.2	Explicitně	59
2.1.3	Objekt, třída, instance, odkaz	59
2.1.4	Kontejnery	61
2.2	Datový typ	62
2.3	Nejdůležitější zvláštnosti Pythonu	63
	Přísné a benevolentní programovací jazyky	63
2.4	Zápis celých a desetinných (reálných) čísel	64
2.4.1	Zpřehlednění dlouhých čísel pomocí znaku podtržení	65
2.4.2	Rozlišení celých a reálných čísel	66
2.5	Komplexní čísla	67
2.6	Počáteční nula u celých čísel	68
2.7	Zadávání čísel v jiných číselných soustavách	68
2.8	Logické hodnoty – platí × neplatí	69
2.9	Nic – None	69
2.10	Výpustka – Ellipsis,	70
2.11	Objekt NotImplemented	70
2.12	Literály	70
2.13	Důležitost přehlednosti	71
2.14	Shrnutí kapitoly a doprovodné soubory	72
Kapitola 3	Zadávání textů – stringů	73
3.1	Stringové literály	73
3.1.1	Víceřádkové stringy	74
3.1.2	Použití funkce <code>print()</code>	76
3.2	Komentáře	76
3.3	Escape sekvence	77
3.3.1	Zobrazení escape-sekvencí funkcí <code>print()</code>	81
3.3.2	Zjištění názvu znaku	82
3.4	Slučování sousedních textových literálů	82
3.5	Prefixy stringových literálů	83
3.6	Stringová interpolace – f-stringy	85
3.6.1	Samodokumentující se výrazy	85
3.6.2	Shrnutí zásad pro práci s f-stringy	85
3.7	Bajtové objekty	87
3.7.1	Bajtové stringy a bajtové literály	87
3.7.2	Třída <code>bytearray</code> – zadávání bajtových polí	88
3.8	Shrnutí kapitoly a doprovodné soubory	89
Kapitola 4	Volání funkcí	90
4.1	Volání volatelného objektu	90
4.1.1	Parametr versus argument	91
4.1.2	Syntaxe volání a návratová hodnota	92
4.1.3	Pořadí vyhodnocování a předávání argumentů	92
4.1.4	Datové × funkční (volatelné) objekty	92
4.2	Vytváření objektů	93
4.3	Vestavěné funkce s jednoduchými argumenty	94

4.3.1	Zápis syntaxe.....	94
4.3.2	Anotace	95
4.3.3	Signatura funkce	96
4.3.4	Přehled vestavěných funkcí s argumenty jednoduchých typů	97
4.4	Získání nápovědy	104
4.4.1	Argument zadán	104
4.4.2	Bez argumentu.....	106
4.5	Rozdělení volání na více řádků	107
4.6	Shrnutí kapitoly a doprovodné soubory	107
Kapitola 5 Jednoduché výrazy		109
5.1	Trocha teorie	109
5.1.1	Výraz	109
5.1.2	Operace	110
5.1.3	Operand	110
5.1.4	Operátor	110
5.1.5	Arita operátorů.....	110
	Operátory jako volatelné objekty – funkce	111
5.1.6	Priorita operátorů	111
5.1.7	Asociativita binárních operátorů	113
5.2	Závorky, čárka jako operátor vytvoření n-tice	114
5.2.1	Pozor na záměnu čárky s desetinnou tečkou	115
5.3	Numerické operace	116
5.3.1	Tři druhy dělení.....	116
5.3.2	Umocňování.....	117
5.3.3	Nekonečna a nesmyslná čísla.....	118
5.4	Operace se stringy	119
5.4.1	Sečítání textů	120
5.4.2	Násobení stringu celým číslem.....	120
5.4.3	Indexace jednotlivých znaků.....	121
5.5	Shrnutí kapitoly a doprovodné soubory	122
Kapitola 6 Proměnné, výrazy, příkazy		123
6.1	Co jsou to proměnné	123
6.1.1	Proměnná versus atribut – kvalifikace.....	124
6.2	Správa paměti	124
	Statické a dynamické typování	125
6.3	Pravidla pro tvorbu identifikátorů	126
6.3.1	Klíčová slova tvrdá a měkká	127
6.3.2	Systémové identifikátory – dunder.....	127
6.4	Zavedení proměnné – přiřazovací příkaz	128
6.5	Výrazy × příkazy, výrazové příkazy	130
6.5.1	Výrazové příkazy	130
6.5.2	Zadání více příkazů na řádku – oddělovací středník	130
6.5.3	Zadání skupiny hodnot.....	131
6.5.4	Proměnné inf a nan	133
6.5.5	Proměnná _	133
6.6	Vnořená volání volatelných objektů	134
6.7	Zjištění typu objektu v proměnné	135
6.8	Uložení funkce do proměnné	136
6.9	Lambda-výrazy	137
6.9.1	Využití při snižování počtu argumentů	138
6.10	Datové a funkční proměnné	139
6.11	Mezery ve výrazech a příkazech	139
6.12	Uložení do proměnné × propojení s názvem	139
6.13	Přiřazovací výraz	140
6.14	Pomocné proměnné	141

6.15 F-stringy – rozšiřující informace	141
6.15.1 Další pravidla	142
6.15.2 Formátování f-stringů	143
6.16 Shrnutí kapitoly a doprovodné soubory	144
Kapitola 7 Logické hodnoty a operace	145
7.1 Konstanty True a False	145
7.2 Převod jiných hodnot na logické	146
7.3 Logické operátory a operace	147
7.3.1 Zkrácené vyhodnocení	147
7.3.2 Pozor na priority	148
7.4 Porovnávání hodnot	149
7.4.1 Porovnání reálných čísel	149
7.4.2 Porovnávání a řazení textů – stringů	150
7.4.3 Zřetěžené porovnávání	150
7.4.4 Porovnávání totožnosti objektů – operátor is	151
7.5 Operace s jednotlivými bity	152
7.6 Bitové posuny	153
7.6.1 Aritmetický × logický posun	154
7.7 Podmíněný výraz	154
7.8 Shrnutí kapitoly a doprovodné soubory	155
Kapitola 8 Další jednoduché příkazy	157
8.1 Přiřazovací příkaz	157
8.2 Rozšířený přiřazovací příkaz	157
8.3 Příkaz del	158
8.4 Příkaz assert	159
Návrh podle kontraktu	160
8.5 Příkaz pass	162
8.6 Složené příkazy a odsazování	163
8.6.1 Výhody a nevýhody koncepce <i>Pythonu</i>	164
8.7 Restart systému	165
8.8 Shrnutí kapitoly a doprovodné soubory	165

Část B Složené příkazy

167

Kapitola 9 Moduly	168
9.1 Základní principy modulů	168
9.1.1 Vše je součástí nějakého modulu	168
9.1.2 Dva názvy objektů	168
9.1.3 Zdrojový soubor	169
9.1.4 Přeložený soubor	169
9.2 Další trocha teorie OOP	170
9.2.1 Atributy	170
9.2.2 Práce s objekty – kvalifikace	170
9.2.3 Další podrobnosti o tvorbě objektů	171
Terminologická vsuvka: co je konstruktor	172
9.3 Příkaz import	173
9.3.1 Čistý import jiného modulu	173
9.3.2 Import modulu pod jiným názvem	175
9.3.3 Přímý import vyjmenovaných objektů	176
9.3.4 Import objektů modulu nezahrnuje import jejich modulu	177
9.3.5 Argumentem příkazů import a from ... import nesmí být výraz	178
9.3.6 Hromadný import všech atributů modulu – hvězdičkový import	178
9.3.7 Systémové identifikátory	180
9.3.8 Syntaktické diagramy příkazu import	180

9.4 Modul jako objekt	181
9.4.1 Modul <code>builtins</code> a zdánlivě neobjektové programování.....	182
9.5 Postup systému při importu modulu	182
9.6 Shrnutí kapitoly a doprovodné soubory.....	183
Kapitola 10 Vytvoření vlastního modulu	185
10.1 Demonstrační modul <code>m10a_Modul</code>	185
10.1.1 Kódová stránka.....	187
10.1.2 Dokumentační komentář a atribut <code>__doc__</code>	187
10.1.3 Kontrolní tisky načítání modulu.....	188
10.1.4 Definice datových atributů	188
10.1.5 Neveřejné atributy	188
10.1.6 Reakce na zadané příkazy	189
10.1.7 Alternativní zveřejnění atributů.....	189
10.2 Modul <code>dbg</code> a kontrolní tisky	190
10.2.1 Co modul <code>dbg</code> není	191
10.3 Průběh importu vytvořeného modulu.....	192
10.3.1 Použitelné názvy.....	193
10.4 Hromadný (hvězdičkový) import	194
10.5 Zveřejnění atributů pomocí <code>__all__</code>	196
10.6 Import jako přiřazovací příkaz – shrnutí.....	198
10.7 Reimport již importovaného modulu	199
10.7.1 Pozor na přímo importované proměnné	201
10.7.2 Specifika funkce <code>importlib.reload()</code>	203
10.7.3 Rozbor chybového hlášení.....	204
10.7.4 Důsledky chybného zavedení modulu	204
10.7.5 Syntaktické chyby	204
10.8 Shrnutí kapitoly a doprovodné soubory.....	205
Kapitola 11 Balíčky.....	206
11.1 Balíčky (packages)	206
11.1.1 Initor standardního balíčku	207
11.1.2 Zobrazování začátku a konce inicializace balíčků a modulů	207
11.1.3 Uspořádání balíčků pro demonstraci	208
11.1.4 Demonstrace importu podbalíčků	210
11.2 Relativní import.....	212
11.2.1 Relativní import z rodičovských balíčků.....	212
11.2.2 Relativní import z podbalíčků	213
11.2.3 Demonstrace relativního importu	213
11.2.4 Cyklický import.....	215
11.3 Co dělat, když interpret na modul nevidí.....	215
11.3.1 Detekce a změna pracovní složky	216
11.4 Kde hledat moduly.....	217
11.4.1 Prohledávané složky	217
11.5 Přidání složky mezi prohledávané	218
11.6 Shrnutí kapitoly a doprovodné soubory.....	219
Kapitola 12 Definice funkcí.....	221
12.1 Povšechně o funkcích.....	221
12.1.1 Funkce je standardní objekt.....	222
12.2 Definice funkce je jen zvláštní přiřazovací příkaz	222
12.3 Definice vlastní funkce.....	222
12.3.1 Jednořádková definice	223
12.3.2 Víceřádková definice.....	224
12.3.3 Interaktivní režim versus zdrojový kód modulu.....	225
12.3.4 Prázdné funkce	226
12.4 Zadávání stringů zabírajících více řádků.....	227

12.4.1 Použití funkce <code>textwrap.dedent()</code>	229
12.5 Definice funkcí v modulu	230
12.6 Kdy se projeví chyby v definici funkce	231
12.7 Shrnutí kapitoly a doprovodné soubory	233
Kapitola 13 Parametry, argumenty a lokální proměnné funkcí	234
13.1 Parametry, argumenty a lokální proměnné	234
13.1.1 Definice	234
13.1.2 Lokální proměnné	235
13.1.3 Volání funkcí s parametry	236
13.1.4 Povinné pojmenované argumenty	237
13.1.5 Povinné poziční argumenty	238
13.1.6 Mix pozičních a pojmenovaných argumentů	239
13.2 Implicitní hodnoty argumentů	240
13.2.1 Ještě jednou porovnání reálných čísel	242
13.3 Konstantnost předdefinovaných hodnot	243
13.4 Funkce s vedlejším efektem	244
13.5 Funkce vracějící hodnotu a příkaz <code>return</code>	244
13.6 Přetěžování funkcí	245
13.6.1 Něco přetížit jde	246
13.7 Anotace	247
13.8 Dekorátory	248
13.9 Pomocné funkce a dekorátory z modulu <code>dbg</code>	249
13.9.1 Základní funkce modulu <code>dbg</code>	249
13.9.2 Přehled nabízených dekorátorů	250
13.10 Shrnutí kapitoly a doprovodné soubory	253
Kapitola 14 Pokročilé rysy funkcí	255
14.1 Vnitřní funkce	255
14.1.1 Odsazování	257
14.1.2 Přístup z okolního kódu	257
14.1.3 Import uvnitř funkce	258
14.2 IDLE a nastavení Show Code Context	258
14.3 Jmenné prostory	259
14.4 Oblast/rozsah platnosti, působnost (scope)	259
14.4.1 Zanoření oblastí platnosti	260
14.5 Lokalita použitých proměnných	261
14.5.1 Volná proměnná	261
14.5.2 Příkaz <code>global</code>	263
14.5.3 Příkaz <code>nonlocal</code>	264
14.6 Vnoření funkce versus vnoření volání funkcí	267
14.7 Zobrazování kontextu	267
14.8 Vnořená volání funkcí	268
14.9 Funkce vyššího řádu	269
14.10 Atributy funkcí	270
14.11 Nelokální proměnné a uzávěry (closures)	271
14.12 Zákaz hvězdičkového importu uvnitř funkcí	272
14.13 Možné řešení cyklického importu	272
14.14 Shrnutí kapitoly a doprovodné soubory	274
Kapitola 15 Rozhodování a příkaz <code>with</code>	276
15.1 Rozhodovací příkazy	276
15.2 Jednoduchý podmíněný příkaz	277
15.3 Úplný podmíněný příkaz	277
15.4 Rozšířený podmíněný příkaz	279
15.5 Přepínač <code>match</code>	279

15.5.1	Trocha terminologie	280
15.5.2	Postup vyhodnocení	281
15.5.3	Sdružování hodnot ve vzorech	281
15.5.4	Klíčové slovo <code>_</code> je jen symbol	282
15.5.5	Další možnosti	282
15.6	Představení knihovny <code>KarelCz7475</code>	283
15.6.1	Práce se světem – dvorkem	283
15.6.2	Zadatelné příkazy	284
15.6.3	Testovatelné podmínky	284
15.6.4	Skrytí aktuálního stavu robota	285
15.7	Příkaz <code>with</code> a správce kontextu	285
15.7.1	Pravidla používání příkazu <code>with</code>	286
15.7.2	Aplikace na příkladu s robotem	286
15.8	Shrnutí kapitoly a doprovodné soubory	288
Kapitola 16 Opakování kódu – rekurze a cykly		289
16.1	Rekurze	289
16.1.1	Nekonečná rekurze	289
16.1.2	Rádně ukončená rekurze	290
	Zásobník návratových adres – ZNA	291
16.1.3	Rekurzivní definice faktoriálu a dodatečná dekorace funkce	291
16.1.4	Nevhodnost použití rekurze pro hluboké volání	293
16.2	Příkaz <code>while</code> – cyklus se vstupní podmínkou	293
16.3	Nekonečný cyklus	294
16.4	Příkaz <code>break</code> – cyklus s podmínkou uprostřed	295
16.5	Cyklus s ukončovací podmínkou	296
16.6	Přiřazení v hlavičce cyklu	296
16.7	Větev <code>else</code>	298
16.8	Příkaz <code>continue</code>	299
16.9	Účel a syntaxe cyklu <code>for</code>	300
16.10	Vyjmenování hodnot parametru cyklu	301
16.11	Využití objektů typu <code>range</code>	302
16.11.1	Důvod použití větve <code>else</code>	302
16.12	Větev <code>else</code>	303
16.13	Použití indexů	304
16.14	Použití stringu jako zdroje	305
16.15	Využití objektů typu <code>enumerate</code>	305
16.16	Vnořování cyklů	306
16.17	Postupné použití několika zdrojů	306
16.18	Shrnutí kapitoly a doprovodné soubory	307
Kapitola 17 Tvorba aplikací		309
17.1	Vytvoření aplikace či knihovny	309
17.2	Přímé spuštění versus <code>import</code> modulu	309
17.2.1	Rozpoznání režimu, v němž byl modul spuštěn	310
17.2.2	Demonstrace rozpoznání způsobu spuštění	310
17.2.3	Spuštění modulu umístěného v balíčku	311
17.3	Argumenty příkazového řádku	312
17.4	Rozsáhlejší aplikace	315
17.4.1	Umístění kódu aplikace	316
17.4.2	Initor balíčku a změny záhlaví modulů balíčku	316
17.4.3	Modul <code>io</code> pro ošetření vstupu	316
17.4.4	Modul <code>game</code> s jádrem hry	318
17.4.5	Řídící modul <code>main</code>	319
17.5	Vytvoření spustitelné aplikace	320

17.5.1 Soubor typu <code>pyz</code>	321
17.5.2 Vytváříme aplikaci „ručně“	321
17.5.3 Alternativy	322
17.6 Shrnutí kapitoly a doprovodné soubory	322
Kapitola 18 Ošetřování chyb	323
18.1 Tři druhy chyb	323
18.1.1 Syntaktické chyby	323
18.1.2 Běhové chyby	324
18.1.3 Logické chyby	325
18.1.4 Varování	325
18.2 Chybové zprávy	325
18.2.1 Syntaktické chyby při práci v interaktivním režimu	326
18.2.2 Syntaktické chyby při překladu importovaného modulu	326
18.2.3 Běhové chyby	327
Odchylky v hlášeném prvotním zdroji	328
18.3 I chyby jsou objekty – výjimky	329
18.4 Rozdělení výjimek	330
18.4.1 Výjimky versus varování	331
18.5 Zachycení a ošetření výjimky	331
18.5.1 Více větví <code>except</code>	332
18.5.2 Větev <code>else</code>	333
18.5.3 Větev <code>finally</code>	333
18.5.4 Syntaktický diagram příkazu <code>try</code>	333
18.5.5 Příklad s kompletní verzí příkazu <code>try</code>	334
18.6 Praktický příklad	336
18.7 Zdánlivé záludnosti větve <code>finally</code>	336
18.8 Vyvolání výjimky	339
18.9 Problematika oddělených tisků	341
18.10 Příkaz <code>assert</code>	342
18.11 Jak vyvolat a zachytit varování	343
18.11.1 Vyvolání varovného hlášení	343
18.11.2 Zachycení varovného hlášení	344
18.11.3 Režimy zachytávání	345
18.12 Shrnutí kapitoly a doprovodné soubory	346

Část C Kontejnery

347

Kapitola 19 Seznamy	348
19.1 Proměnné a neměnné objekty	348
19.2 Základní informace o seznamech	349
19.3 Vytváření seznamů	350
19.3.1 Použití literálu	350
19.3.2 Využití konstruktoru <code>list()</code>	351
19.3.3 Sčítání a násobení	352
19.3.4 Generátorová notace seznamů	353
19.3.5 Tovární funkce a funkce vracející seznam	355
19.4 Modifikace seznamů	356
19.4.1 Metody <code>append()</code> a <code>extend()</code>	356
19.4.2 Rizika práce s odkazy na proměnné objekty	357
19.4.3 Postupné budování seznamu	358
19.4.4 Přičítání jiných zdrojů	359
19.4.5 Indexace prvků seznamu	360
19.4.6 Metody pracující s indexy	361
19.4.7 Metody pracující s celým seznamem	362

19.5	Vícerozměrné seznamy	363
19.6	Souhrnný příklad	364
19.7	Anotace odkazující na seznamy	365
19.8	Shrnutí kapitoly a doprovodné soubory	366
Kapitola 20	N-tice	367
20.1	Základní informace o n-ticích	367
20.2	Vytváření n-tic	368
20.2.1	Vytváření n-tic pomocí literálů	368
20.2.2	Využití konstruktoru <code>tuple()</code>	369
20.2.3	Sčítání a násobení	370
20.2.4	Přičítání n-tic	371
20.2.5	Balení a rozbalování n-tic	372
20.2.6	Prohazování proměnných	372
20.2.7	Hvězdičkové pravidlo	373
20.3	Generátorová notace n-tic	373
20.4	Problematika neměnnosti n-tic	374
	Hešovatelné objekty	375
20.5	Přístup k prvkům n-tic	375
20.6	Sčítání seznamů a n-tic	376
20.7	Proměnné a neměnné prvky n-tice	377
20.8	Pojmenované n-tice	378
20.9	Anotace odkazující na n-tice	380
20.10	Shrnutí kapitoly a doprovodné soubory	381
Kapitola 21	Množiny	382
21.1	Základní informace o množinách	382
21.2	Vytváření množin	382
21.2.1	Vytváření množin pomocí literálů	383
21.2.2	Vytváření množin pomocí konstruktoru <code>set()</code>	383
21.2.3	Vytváření množin prostřednictvím množinových operací	385
21.3	Generátorová notace množin	388
21.4	Zmrazené množiny	388
21.5	Modifikace množin	389
21.5.1	Modifikace pracující s jedním prvkem	390
21.5.2	Množinové operátory a rozšířené operace	391
21.5.3	Porovnávání množin	393
21.6	Anotace odkazující na množiny	394
21.7	Shrnutí kapitoly a doprovodné soubory	394
Kapitola 22	Slovníky	395
22.1	Mapovací objekty a slovníky	395
22.2	Vytváření slovníků	396
22.2.1	Vytváření slovníků pomocí literálů	396
22.2.2	Vytváření slovníků pomocí konstruktoru <code>dict()</code>	397
22.2.3	Ekvivalence slovníků	398
22.2.4	Vytváření slovníků pomocí metody <code>fromkeys()</code>	398
22.3	Generátorová notace slovníků	399
22.4	Operace se slovníkem	401
22.4.1	Práce s hodnotami pomocí „indexace“ klíčem	401
22.4.2	Další metody pro práci s jednotlivými položkami	402
22.4.3	Modifikace slovníku daty ze zadaného zdroje	403
22.4.4	Slovník jako generátor	404
22.5	Pohledy	404
22.5.1	Pohledy jako zdroje dat	405
22.5.2	Operace s pohledy	406
22.6	Anotace odkazující na slovníky	407

22.7 Shrnutí kapitoly a doprovodné soubory	407
Kapitola 23 Rozšíření definic funkcí	409
23.1 Předávání argumentů odkazem a hodnotou	409
23.1.1 Dvě hodnoty odkazu	410
23.1.2 Proč jde o předávání hodnotou	410
23.2 Pomocná funkce <code>gr()</code>	412
23.3 Proměnný počet pozičních argumentů	412
23.3.1 Hvězdičkový parametr	413
23.3.2 Hvězdičkový argument	414
23.4 Proměnný počet pojmenovaných argumentů	416
23.4.1 Dvuhvězdičkový parametr	416
23.4.2 Dvuhvězdičkový argument	416
23.5 Stručný souhrn	417
23.5.1 Podivné chování	418
23.5.2 Použití v definicích literálů	418
23.6 Vestavěné funkce pracující s kontejnery	419
23.7 Funkce modulu <code>dbg</code> pracující s kontejnery	425
23.8 Shrnutí kapitoly a doprovodné soubory	426
Kapitola 24 Formátování stringů	428
24.1 Formátovací operátor <code>%</code>	428
24.2 Pokročilejší metody formátování	430
24.2.1 Metody <code>format()</code> a <code>format_map()</code> versus f-stringy	430
24.2.2 Formátovací string	431
24.2.3 Syntaxe nahrazovacího pole	432
24.2.4 Vyhodnocení nahrazovaného textu	432
24.2.5 Formátování nahrazujícího textu	434
24.2.6 Konverze	435
24.3 Specifikace formátu	436
24.3.1 Počet zabraných pozic	436
24.3.2 Přesnost	438
24.3.3 Typ hodnoty	438
24.3.4 Skupiny číslic	441
24.3.5 Alternativní formát a vedoucí nuly	442
24.3.6 Znaménko	443
24.3.7 Zarovnání a plnění	443
24.3.8 Vnořená nahrazovací pole ve specifikaci formátu	445
24.3.9 Formátování samodokumentujících se nahrazovacích polí	446
24.4 Příklad: Pascalův trojúhelník	446
24.5 Formátování instancí dalších typů	447
24.6 Šablony stringů	448
24.6.1 Instanční metody třídy <code>string.Template</code>	449
24.6.2 Ukázka použití stringových šablon	451
24.7 T-stringy jako šablony stringů	451
24.8 Shrnutí kapitoly a doprovodné soubory	452
Kapitola 25 Operace s kontejnery	453
25.1 Přítomnost prvku v kontejneru	453
25.2 Proměnné objekty jako implicitní hodnoty parametrů	454
25.3 Kopírování	454
Mělké a hluboké kopie objektů	454
25.3.1 Zdánlivé kopie neměnných kontejnerů	455
25.3.2 Alternativní způsob tvorby mělkých kopií	456
25.3.3 Nebezpečí hlubokých kopií	456
25.4 Rozdělení doposud probraných kontejnerů	456
25.5 Řazení prvků posloupnosti	457

25.5.1	reversed(seq)	458
25.5.2	sorted(iterable, *, key=None, reverse=False)	458
25.6	Vykrajování (slicing)	459
25.7	Indexování a vykrajování u rozsahů	460
25.8	Nahrazování hodnot	461
25.9	„Úprava“ neměnných objektů	463
25.10	Shrnutí kapitoly a doprovodné soubory	463
Kapitola 26	Práce se soubory	464
26.1	Soubory: bleskové opakování	464
26.1.1	Soubor, souborový systém, cesta	465
26.1.2	Absolutní a relativní cesta	466
26.1.3	Substituované disky ve Windows	466
26.2	Koncepce vstupu a výstupu dat v <i>Pythonu</i>	466
26.2.1	Starší koncepce souborů v jazycích C nebo Pascal	467
26.2.2	Novější koncepce datových proudů	467
26.2.3	Koncepce Pythonu	467
26.2.4	Shrnutí používané terminologie	468
26.3	Dva způsoby práce se souborovým systémem	469
26.4	Moduly os a os.path	471
26.5	Pracovní složka	472
26.6	Skládání a rozkládání cest	473
26.7	Vytváření a mazání složek	474
26.7.1	Mazání	475
26.8	Získání informací o souborech	476
26.9	Zápis a čtení dat	477
26.9.1	Problematika kódování ve Windows	477
26.9.2	Otevírání souborů a datových proudů	478
26.9.3	Zápis dat, splachování a zavírání proudů a přidružených souborů	480
26.10	Automatické zavírání proudů použitím with	482
26.11	Čtení ze souborů	483
26.12	Shrnutí kapitoly a doprovodné soubory	485
Literatura		487
Rejstřík		489

0 autorovi

Ing. Rudolf Pecinovský, CSc. studoval teoretickou kybernetiku na *Fakultě jaderné a fyzikálně inženýrské ČVUT* a poté technickou kybernetiku na *Fakultě Elektrotechnické ČVUT*, kterou ukončil v roce 1979. Titul CSc. získal v Ústavu teorie informace a automatizace ČSAV v roce 1983. Od počátku 80. let učí a publikuje, přičemž svůj výzkum soustředí především na oblast vstupních kurzů moderního programování a výuku objektivě orientovaného paradigmatu. V současné době učí na *Fakultě jaderné a fyzikálně inženýrské ČVUT* a na *Fakultě informatiky a statistiky Vysoké školy ekonomické v Praze*. Vedle toho učí ještě v řadě kurzů pro začátečníky i profesionální programátory. Doposud mu vyšlo přes 70 knih, které byly přeloženy do pěti jazyků. Většina jeho knih je zaměřena na výuku moderního programování a návrh objektivě orientované architektury.



Poděkování

Vím, že se v českých knížkách většinou neděkuje, ale tvorba knih je spojena s takovými oběťmi řady lidí z mého blízkého i vzdálenějšího okolí, že bych měl velkou újmu na duši, kdybych tak neučinil.

Chtěl bych především nesmírně poděkovat své ženě Jarušce, která byla po celou dobu mojí největší oporou a jejíž nekonečná trpělivost a vstřícnost mi pomohla dokončit knihu v termínu, který se příliš nelišil od toho, jež jsme původně s nakladatelem dohodli, a ne až někdy za rok po něm. Stále marně přemýšlím, kde má schovanou tu svatozář.

Původně jsem se domníval, že s dalším vydáním nebude moc práce. Šeredně jsem se zmýlil, protože veškeré úpravy, jež jsem do knihy zanesl, jsou vykoupeny hodinami studia a experimentování, které rodina s neuvěřitelnou trpělivostí snášela.

Na vylepšování textu nového vydání se podílela řada dalších lidí. Mezi nimi musím poděkovat především těm, kteří si dali práci s odhalováním případných chyb ve vznikajícím rukopisu. Mezi nimi pak především Ludkovi Šťastnému, který po celou přípravu rukopis pročetl a odhaloval v něm pasáže, jež by si zasloužily vylepšit. Neméně velkou zásluhu na současné podobě má i Jirka Kofránek, který mne průběžně upozorňoval na některé problémy s výukou podle běžně používaných postupů. Řadu podnětných myšlenek přinesl i Michal Palas, jenž pracuje v oblasti analýzy a zpracování dat a přispěl tak řadou poznatků z praxe.

V neposlední řadě patří můj dík redakci, především redaktoru Petru Somogyimu, který trpělivě snášel mé neustálé modifikace již zkorigovaného textu, a Radku Matulíkovi, který mne k napsání jednotlivých knih z posledních let vyhecoval a byl pak ochoten týden či dva počkat, když se mi nepodařilo přesně dodržet původně dohodnutý termín.

Úvod

Python je moderní programovací jazyk, který umožňuje velmi jednoduše navrhovat jednoduché programy, ale na druhou stranu nabízí mocné prostředky k tomu, abyste mohli s přiměřeným úsilím navrhovat i programy poměrně rozsáhlé. Je pro něj vyvinuto obrovské množství knihoven, které uživatelům umožňují soustředit se na řešení úkol a nerozptylovat se vývojem nejrůznějších pomocných podprogramů.

Python je v současné době nejlepším jazykem pro ty, kteří se nechtějí živit jako programátoři, ale jejich profese či zájem je nutí jednou za čas něco naprogramovat. Potřebují proto jazyk, který se mohou rychle naučit a v němž budou moci rychle vytvářet jednoduché programy řešící jejich problémy nebo alespoň pomáhající při jejich řešení.

Python je vynikajícím nástrojem i pro ty, kteří vyvíjejí rozsáhlejší programy a ocení jeho obrovské možnosti, které běžné jazyky neposkytují. Oblíbí si ho všichni, kteří nepotřebují pracovat v jazyku plném pravidel zabraňujících častým chybám začínajících programátorů a poskytujícím nepřetržitý dozor přísného překladače kontrolujícího dodržování těchto pravidel.

Pro takový druh uživatelů je určena tato příručka. Určitě ji však přivítají i čtenáři, kteří řeší složitější problémy a potřebují proto znát jazyk do větší hloubky. Najdou zde výklad leckterých konstrukcí, na něž v běžných učebnicích nezbylo místo.

Komu kniha není určena

Dopředu musím upozornit, že tato kniha není koncipována pro naprosté začátečníky, kteří se s programováním teprve seznamují. Ti potřebují poněkud jiný postup výkladu a jiný výběr příkladů. Pro ně je určena kniha *Začínáme programovat v jazyku Python* ([15]), která vedle používání základních konstrukcí učí své čtenáře také řadu zásad moderního programování, jejichž zvládnutí je nutnou podmínkou pro všechny, kteří nehodlají zůstat u malých žákovských programů, ale chtějí se naučit efektivně vyvíjet robustní středně rozsáhlé aplikace, jejichž údržba nepovede uživatele k chrlení nepublikovatelných výroků na adresu jejich autorů.

Kniha, kterou právě čtete, předpokládá alespoň minimální programátorské znalosti a zkušenosti, což jí umožní soustředit se na výklad konstrukcí a rysů jazyka včetně těch, na které v učebnicích pro začátečníky již obvykle nezbyvá místo.

Komu je kniha určena

Je určena pro tři typy uživatelů:

- Autory, jejichž programátorské zkušenosti nejsou příliš hluboké, protože je využívají pro tvorbu jednoduchých programů řešících jejich každodenní problémy. Pro ty jsem výklad konstrukcí jazyka občas proložil výklady základů programování, které mohou ti pokročilejší přeskočit.

U těchto čtenářů bych byl rád, kdyby jim kniha pomohla v postupném prohlubování jejich znalostí a dovedností. Narazí-li někde na obtížnější pasáž, jejíž znalost právě nepotřebují, mohou ji klidně přeskočit a vrátit se k ní, až se jejich zkušenosti prohloubí a budou potřebovat probíranou vlastnost použít.

- Programátory, kteří doposud vyvíjeli v jiném programovacím jazyce a chtějí rozšířit své dovednosti o programování v jazyce *Python*. Pro ty jsou určena různá upozornění na rysy, jimiž se *Python* může odlišovat od toho, na co jsou ze své dosavadní praxe zvyklí.

Z ohlasů na předchozí verze publikace odhaduji, že těchto čtenářů je 60 až 80 %, takže se jim budu snažit vyjít vstříc a vždy včas upozornit na potřebu odlišného přístupu k probírané konstrukci při návrhu programu.

Předpokládám navíc, že tito čtenáři budou chtít tuto knihu používat spíše jako referenční příručku pro situace, kdy si nebudou zcela jisti, jak se některá konstrukce používá nebo jak přesně funguje.

- Uživatelé, kteří již v *Pythonu* programují, ale potřebují znát jazyk do větší hloubky, než jim poskytly absolvované kurzy a prostudované učebnice, nebo se potřebují seznámit s novými konstrukcemi a rysy, které se mezi tím v *Pythonu* objevily.

K dokonalému využití platformy *Python* je však potřeba i znalost základních knihoven. V této příručce vám představím pouze ty nejzákladnější funkce a třídy. Těm, kteří touží po podrobnějším seznámení s knihovnou a jejími možnostmi v oblasti práce s daty, poslouží publikace [\[13\]](#).

Struktura příručky

Tato kniha je prvním dílem dvoudílné příručky. V tomto dílu se čtenář seznámí se vestavěnými jednoduchými datovými typy a základními algoritmickými konstrukcemi jazyka. Dozví se základní informace o práci s objekty a naučí se pracovat s nejdůležitějšími strukturovanými datovými typy.

Druhý díl nazvaný [Python 3.14 – Objektové konstrukce](#) je pak zaměřen na konstrukce podporující objektově orientované programování a na hlubší pochopení některých pokročilých programových konstrukcí.

Příručka se nesnaží probrat rozsáhlou knihovnu, která je součástí standardní konfigurace, nicméně většinu jejích funkcí typický uživatel nevyužije. Seznamuje proto

opravdu jen s jejími nejdůležitějšími součástmi, které použije prakticky každý. O použití některých nadstavbových součástí vyšly nebo vyjdou samostatné publikace, přičemž spektrum těchto navazujících učebnic mohou ovlivňovat sami čtenáři.

Tento přístup umožňuje probrat jazyk do větší hloubky, takže se dostane i na oblasti, které běžné učebnice opomíjejí a očekávají, že si některá témata čtenáři osvojí metodou pokus-omyl.

První díl

První díl se věnuje především algoritmickým konstrukcím a základním datovým strukturám. Je rozdělen do tří částí doprovázených samostatně distribuovanými přílohami:

- První část seznamuje s platformou Python a s vývojovými prostředími, která používám pro definici ukázkových příkladů a demonstraci jejich funkce. Poté vám ukáže, jak zadávat hodnoty jednoduchých datových typů, seznámí vás s jejich použitím ve výrazech a s jednoduchými příkazy, které s takovými daty pracují. Její zvládnutí je podmínkou pro studium dalších částí.
- Druhá část probírá složené příkazy. Postupně probereme definice modulů, funkcí a řady složených příkazů. Po jejím zvládnutí budete schopni vytvářet jednoduché programky, i když se bude jednat spíše o programky na hraní.
- Třetí část rozšiřuje množinu používaných dat o kontejnery, což jsou objekty určené k uchování jiných objektů. Snažil jsem se v ní uvést i pár příkladů, které již nejsou pouze demonstrační, a měly by mít i praktické využití. Po jejím zvládnutí byste měli být schopni vytvářet v *Pythonu* užitečné programy, které budou moci používat jiní.

Druhý díl

Druhý díl se soustředí na konstrukce pro podporu objektově orientovaného programování, které rozebírá do nadstandardní hloubky. Je rozdělený do čtyř částí.

- První část se zaměřuje na výklad základních konstrukcí umožňujících a podporujících objektově orientované programování. Po jejím zvládnutí budete schopni vytvářet v *Pythonu* středně složité programy.
- Druhá část probírá tyto konstrukce do větší hloubky a ukazuje, jak definovat ekvivalenty některých konstrukcí a datových struktur, které se do té doby v příručce jenom používaly.
- Třetí část se pak věnuje pokročilejším objektovým konstrukcím, jejichž znalost umožňuje navrhovat efektivnější architekturu vytvářených programů. Podrobně seznamuje s náročnými tématy jako přetěžování operátorů, správa paměti a použití metatříd.
- Čtvrtá část se pak soustředí na podporu souběžného programování, při němž počítač provádí několik úloh současně. Po stručném seznámení s problematikou

pak postupně představí a na praktických příkladech ukáže nejprve návrh programů pracujících s více vlákny, poté návrh programů pracujících s více procesy, poté představí možnosti asynchronního programování a na závěr seznámí s novinkou verze 3.14, kterou je možnost vytváření subinterpretů.

Samostatnou část knihy tvoří přílohy. Ty ale nejsou součástí tištěného vydání – čtenáři papírových knih je najdou webových stránkách příručky. V elektronických vydáních je ale najdete.

Koncepce výkladu

Snažil jsem se, aby tato kniha mohla sloužit současně jako učebnice jazyka *Python* i jako referenční příručka. Tyto dva druhy publikací si ale ve svých požadavcích poněkud odporují.

- Dobrá učebnice vyžaduje, aby se výklad obešel bez dopředných odkazů. Musí proto maximálně omezit (nejlépe zcela zrušit) používání čehokoliv, co ještě nebylo vysvětleno, a to i za cenu toho, že výklad mnoha témat bude třeba rozdělit do několika částí, aby v něm mohly být vždy použity jen doposud probrané rysy jazyka.

V opačném případě čtenáři vnímají danou konstrukci jako obrázek, který okopírují a používají, aniž by znali význam jeho jednotlivých součástí. (To je ostatně nectnost řady učebnic začínajících oblíbeným příkladem „Hello World“.) Při takovémto přístupu totiž čtenáři občas nechápou důsledky některých obrátů a akcí a příčiny ohlášených chyb, jejichž odstranění je pak stojí nemalé úsilí.

- Naproti tomu referenční příručky by měly umožňovat co nejsnazší a nejrychlejší dohledání potřebných informací. Mohou být proto koncipovány tak, že každé téma je v nich plně probráno na jednom místě se všemi detaily, a to i za cenu dopředných odkazů, protože při výkladu často potřebují použít konstrukci, která teprve bude vysvětlena. Mohou si to dovolit, protože jsou určeny především zkušenějším čtenářům, u nichž se předpokládají základní znalosti a danou referenční příručku používají většinou k tomu, aby si ozřejmili některé detaily.

Jak jsem řekl, chtěl jsem, aby tato kniha mohla sloužit k oběma účelům, tedy jak jako základní příručka, podle které se dá učit, tak jako referenční příručka, kde lze rychle nalézt pasáž zabývající se diskutovaným tématem. Její struktura je proto navržena tak, aby jednotlivé kapitoly probraly dané téma pokud možno komplexně a bez potřeby se k němu vracet, ale aby se v nich na druhou stranu pokud možno neobjevovaly dopředné odkazy i za cenu toho, že bude třeba výklad některých pasáží rozdělit.

Veškerý výklad je prostoupen hojnými AHA-příklady, tedy příklady, jejichž hlavním účelem není vytvoření nějakého zdánlivě užitečného programu, ale kódu, jehož cílem je co nejjednodušeji a co nejsrozumitelněji demonstrovat probíranou konstrukci, aby čtenář pochopil její princip – aby u něj nastal kýžený AHA-efekt.

Vedle nich se občas objeví i nějaký příklad, který by byl v praxi užitečný. Protože se ale nejedná o učebnici programování, ale především o učebnici jazyka, bude těchto praktických příkladů poskrovnu.

Jazyk identifikátorů

Jak jsem řekl, doprovodné programy v první části jsou převážně AHA-příklady vysvětlující probíranou konstrukci. V nich budu pro větší názornost používat české identifikátory. V příkladech, které zavánějí praktickou aplikovatelností, dám – jak bývá v programování zvykem – přednost identifikátorům anglickým.

Potřebné vybavení

Pro úspěšné studium této knihy je vhodné mít instalovanou platformu *Python*. Tu lze stáhnout na adrese <https://www.python.org/downloads/>. Na dané adrese najdete i stručný návod, jak *Python* stáhnout a instalovat.

Kniha je psána pro verzi 3.14, ale drtivá většina příkladů poběží i na verzi 3.10. Protože ale vím, že řada čtenářů pracuje se staršími verzemi (např. proto, že to jejich zaměstnavatel požaduje z důvodu kompatibility), snažím se upozorňovat na všechna vylepšení, která se objevila v novějších verzích počínaje verzí 3.5, abyste byli včas upozorněni na to, co by vám nemuselo na počítači chodit bez vašeho zavinění.

Operační systém

Při práci v *Pythonu* vám může být většinou zcela jedno, nad jakým operačním systémem je instalovaný. V některých výjimečných situacích na tom ale záleží. Protože asi 85 % mých studentů i účastníků mých kurzů používá operační systém *Windows*, jsou mé knihy primárně zaměřeny na ně. Pokusím se ale nezapomínat ani na ten zbytek.

Doprovodné programy a jejich organizace

Text knihy je prostoupen řadou doprovodných programů. Budete-li si je chtít spustit a ověřit jejich funkci, potřebujete je nejprve stáhnout. Najdete je na stránce knihy na adrese¹ http://knihy.pecinovsky.cz/74_Python314bas.

¹ Číslem 74 na počátku poslední složky se nevzrušujte, je to pouze moje interní označení pořadí vytvářené knihy, protože bych v nich jinak bloudil. V textu příručky se s ním setkáte ještě mnohokrát.

Soubory s doprovodnými programy jsou uloženy v ZIP-souboru, jehož název začíná `74_Python_314bas_PGM_`. Za tímto začátkem následuje číslo verze a datum vytvoření. Když tento ZIP-soubor rozbalíte, vytvoří se následující složky:

74_APP Složka určená pro samostatné aplikace, které budeme v průběhu výkladu vytvářet.

74_ILB Složka se spustitelným souborem `Karelcz7475.pyz` obsahujícím knihovnu, se kterou budeme v některých kapitolách pracovat.

74_ILS Složka se zdrojovými kódy knihovny `Karel7475`

74_INP Složka s doprovodnými programy. K její vnitřní struktuře se za chvíli ještě vrátím.

74_IWD Texty výpisů programů v příručce vysazené spolu s čísly řádků, abyste si mohli daný soubor zobrazit vedle knihy např. v textovém editoru a při pročítání popisu daného programu nemuseli listovat v knize, abyste zjistili, jak vypadá řádek s daným číslem, o němž se v textu hovoří. Soubory odpovídají stejnojmenným souborům ze složky `74_INP/lessons`.

Pojďme se nyní podívat na vnitřní strukturu složky `74_INP`. Složka představuje pro doprovodné programy kořenový balíček. V něm je klíčový modul v souboru `dbg.py`, který používám při ladění a k názornému zobrazení průběhu některých procesů. kromě něj zde naleznete pár dalších souborů s příponou `py`, které potřebují být z nejrozumnějších důvodů v kořenové složce.

Kromě toho jsou tu tři podsložky představující balíčky (o balíčcích budeme hovořit v kapitole [11 Balíčky](#) na straně). Každý z nich je schránkou na jeden druh souborů:

lessons Balíček (složka) s doprovodnými programy obsahujícími příkazy zadávané v průběhu kapitol většinou řádkovému interpretu označovanému REPL, občas ale operačnímu systému. Zadání příkazu i odpověď systému jsou pak zobrazeny ve výpisech v knize. Názvy těchto souborů začínají vždy písmenem `m` následovaným dvojmístným číslem kapitoly, dvěma podtržítky a textem naznačujícím téma kapitoly – např. `m01__Startujeme.py`.

modules Balíček se samostatnými moduly, které se vztahují ke konkrétním kapitolám. Názvy zde umístěných souborů začínají vždy písmenem `m` následovaným dvojmístným číslem kapitoly, písmenem určujícím pořadí vzniku souboru v průběhu kapitoly, podtržítkem a textem naznačujícím obsah souboru – například `m10a_Modul.py`.

util Balíček s užitečnými programy, které se občas použijí.

DATA Pomocná složka, kterou budeme používat, až budeme v závěrečné kapitole probírat práci se soubory.

Stažený ZIP-soubor si rozbalte, kam uznáte za vhodné. Jak napovídají adresy souborů v jejich úvodních komentářích, já mám celý ZIP rozbalený do kořenové složky substituovaného disku `R:`, takže složka `R:/74_INP` je kořenovou složkou doprovodných programů.

Soubory s příkazy zadávanými v průběhu kapitoly se vyskytují ve dvou podobách se stejným názvem, ale odlišnou příponou.

- Soubory s příponou **py** jsou zdrojové soubory *Pythonu*, jejichž příkazy je většinou možné přímo zadávat *Pythonu* a které naleznete ve složce **74_INP/lessons**.
- Soubory s příponou **pydoc** obsahují výpisy programů v knize i s uvedenými čísly řádků. Ty obsahují jak příkazy uvedené v souborech ze složky **74_INP/lessons**, tak odpovědi systému. Tyto soubory najdete ve složce **74_IWD** a jak už jsem řekl, mají vám usnadnit sledování rozboru některých programů, abyste při něm nemuseli neustále listovat mezi textem rozboru a rozebíraným výpisem.

Použité typografické konvence

K tomu, abyste se v textu lépe vyznali (a také abyste si vykládanou látku lépe zapamatovali), používám několik prostředků pro odlišení a zvýraznění textu.

Termíny	První výskyt nějakého termínu a další texty, které chci zvýraznit, vysazuji tučně .
<i>Název</i>	Názvy firem a jejich produktů vysazuji <i>kurzivou</i> . Kurzivou vysazuji také názvy kapitol, podkapitol a oddílů, na které v textu odkazují.
Citace	Texty, které si můžete přečíst na displeji, například názvy polí v dialogových oknech či názvy příkazů v nabídkách, vysazuji tučným bezpatkovým písmem .
<u>Odkaz</u>	Celá kniha je prošpikovaná křížovými odkazy na související pasáže. Ne-ní-li odkazovaný objekt (kapitola, obrázek, výpis programu, ...) na stejné stránce nebo na některé ze sousedních stránek, je pro čtenáře tištěné verze doplněn o číslo stránky, na níž se nachází. Čtenářům elektronické verze stačí, když na něj klepnou, a použitý čtecí program by je měl na odkazovaný objekt ihned přenést.
Adresa	Internetové adresy vysazuji obyčejným bezpatkovým písmem.
Program	Identifikátory a další části programů zmíněné v běžném textu vysazuji neproporcionálním písmem , které je v elektronických verzích pro zvýraznění tmavě červené.
Zadání	Ve výpisech konverzace s počítačem budou zadání uživatele vysazena tučně (v elektronických verzích pro zvýraznění modře).
<u>Keyword</u>	Klíčová slova jazyka (anglicky keywords) budou vysezena také tučně i v běžných výpisech programu (v elektronických verzích pro zvýraznění tmavě červeně).
Chyba	Obdobně budou zvýrazněna chybová hlášení. Nebudou podbarvena a v elektronických verzích budou vysazena červeně.

Na několika místech v knize je ukázka komunikace v příkazovém panelu operačního systému. V těchto výpisech zobrazuji pro přehlednost zprávy systému stejně jako chybová hlášení, aby byly jasně odlišeny od následné komunikace s *Pythonem*.

Komentář

Svůj styl budou mít ve výpisech programů i komentáře, které budou vysazeny kurzivou a v elektronických verzích zeleně podbarveny.

Výzva

Výzvy operačního příkazu k zadání příkazu či jeho části budou podbarvené tak, aby byly snadno odlišitelné od zadávaného kódu.

Kromě výše zmíněných částí textu, které považuji za důležité zvýraznit nebo alespoň odlišit od okolního textu, najdete v knize ještě řadu doplňujících poznámek a vysvětlivek. Všechny budou v jednotném rámečku, jenž bude označen ikonou charakterizující druh informace, kterou vám chce poznámka či vysvětlivka předat.



Symbol jin-jang bude uvozovat poznámky, s nimiž se setkáte na počátku každé kapitoly. Zde vám vždy prozradím, co se v dané kapitole naučíte.



Symbol znamení raka označuje poznámky upozorňující na odchylky od *mainstreamových jazyků*, tj. od jazyků *Java*, *C/C++*, *C#*, *Delphi*, *Visual Basic* apod. V této poznámce občas opakuji informace, které jsou v hlavním textu, ale bojím se, že by mohly být přehlédnuty.



Symbol znamení střelce označuje poznámky odkazující na část knihy, kde výklad daného tématu pokračuje.



Obrázek knihy označuje poznámku týkající se používané terminologie. Tato poznámka většinou upozorňuje na další používané termíny označující stejnou skutečnost nebo na konvence, které se k probírané problematice vztahují.



Píšící ruka označuje obyčejnou poznámku, která pouze doplňuje informace z hlavního proudu výkladu o nějakou zajímavost.



Ruka s hrozcím prstem upozorňuje na věci, které byste měli určitě vědět a na něž byste si měli dát pozor, protože jejich zanedbání vás většinou dostane do problémů.



Mračoun vás bude upozorňovat na některé nepříjemnosti a bude vám radit, jak se těmto nástrahám vyhnout (či jak to zařídit, aby vám alespoň pokud možno nevadily).



Brýle označují tzv. „poznámky pro šfouraly“, ve kterých se vás snažím seznámit s některými zajímavými vlastnostmi probírané konstrukce nebo upozorňuji na některé souvislosti, které však nejsou k pochopení látky nezbytné.

Odbočka – podšeděný blok

Občas je potřeba vysvětlit něco, co nezapadá přímo do okolního textu. V takových případech používám podšeděný blok se silnou čarou po straně. Tento podšeděný blok je takovou drobnou odbočkou od výkladu. Nadpis podšeděného bloku pak najdete i v podrobném obsahu mezi nečíslovanými nadpisy.

Při prvním čtení můžete tyto bloky přeskakovat a vracet se k nim až ve chvíli, kdy v textu narazíte na téma probírané v bloku a budete si chtít osvěžit či doplnit svoje znalosti.

Zpětná vazba

Přestože knihu četlo několik lektorů a její přípravě bylo věnováno značné úsilí, nelze vyloučit, že se v ní mohou objevit přehlédnutí, která nemá redaktor šanci zachytit a opravit.

Pokud vám proto bude někde připadat text nepříliš srozumitelný nebo budete mít nějaký dotaz (ať už k vykládané látce či použitému vývojovému prostředí), anebo pokud v knize objevíte nějakou chybu nebo budete mít návrh na nějaké její vylepšení, neostýchejte se poslat svůj dotaz či připomínku na adresu rudolf@pecinovsky.cz jako e-mail s předmětem [74 Python 314bas DOTAZ](#).

Bude-li se dotaz týkat něčeho obecnějšího nebo to bude upozornění na chybu, pokusím se co nejdříve zveřejnit na stránce knihy http://knihy.pecinovsky.cz/74_Python314bas odpověď i pro ostatní čtenáře, kteří by mohli o danou chybu zakopnout, nebo by je mohl obdobný dotaz napadnout za pár dní, anebo jsou natolik ostýchaví, že si netroufnou se sami zeptat.

Musím se ale předem omluvit, že na dotazy neodpovídám okamžitě, protože při svém pracovním vytížení zapínám poštu jen jednou za čas.

Část A

Superzáklady

Tato část seznamuje s naprostými základy jazyka *Python* a s možnostmi práce s jednoduchými daty. Nejprve prozradí, jak instalovat a spustit vestavěné vývojové a výukové prostředí, poté vás naučí zadávat hodnoty různých typů, probere jednoduché aritmetické a logické výrazy včetně základní sady jednoduchých funkcí. Po této předehře se pustí do výkladu proměnných a základních jednoduchých příkazů.

Kapitola 1

Startujeme



Co se v kapitole naučíte

Kapitola vás nejprve seznámí se základními instalovanými součástmi platformy a poté vám představí vývojová prostředí, jež jsou součástí instalace a umožní vám komunikovat s interpretem jazyka *Python*.

1.1 Hlavní součást instalace

V dalším textu budu předpokládat, že máte staženou a instalovanou platformu *Python* podle návodu v pasáži [Potřebné vybavení](#) na straně . Projděme si nyní, co jste instalovali, a ukažme si, co z toho budeme využívat.

1.1.1 Platforma

Nejprve bych vám měl vysvětlit, proč říkám, že jste si stáhli platformu *Python*, když všichni hovoří o jazyku. Jde o to, že samotný jazyk vám není k ničemu do té doby, než získáte nástroje k tomu, abyste jej mohli používat.

Především potřebujete *překladač*, abyste svůj program převedli do podoby, kterou je schopen interpretovat *virtuální stroj*. Současně potřebujete *knihovny*, v nichž jsou připravené podprogramy realizující nejčastější operace, abyste je nemuseli všechny vytvářet sami. Sadu programů potřebných k tomu, aby se váš program plnohodnotně rozběhl, označujeme termínem *platforma*.

Vedle příslušné platformy potřebujete pro úspěšný vývoj programů nějaké vývojové prostředí, ve kterém budete své programy s rozumným úsilím navrhovat, a dokumentaci, v níž najdete potřebné informace a pokud možno i nějaký tutoriál, jenž vás seznámí s naprostými základy.

Python se od ostatních platforem liší mimo jiné také tím, že kromě základního interpretu, překladače, knihoven, frameworků a relativně rozsáhlé dokumentace obsahuje i jednoduché, ale pro začátečníka vcelku použitelné vývojové prostředí, v němž

můžete zadávat své příkazy a vyvíjet programy, a vedle něj i rozumně podrobný tutorial. Tato konstelace proto bývá označována anglickým marketingovým heslem *battery included* (včetně baterií) – když instalujete standardní *Python*, tak nějakou dobu už nebudete nic jiného potřebovat.

Výchozí standardní implementace, kterou si můžete jako platformu stáhnout na stránkách www.python.org, bývá označována jako *CPython*, protože je implementována v jazyku C. Implementace jazyka *Python* vznikla i na jiných platformách. Projděme si alespoň ty nejznámější.

Jython přináší implementaci nad platformou *Java*. V době psaní této učebnice však stále ještě běžela ve verzi 2, která již není od počátku roku 2020 podporována. Vývoj *Jython* 3 je sice v pokročilé fázi, ale ještě nebyla vydána finální stabilní verze.

IronPython přináší implementaci nad platformou .NET a *Mono*.

PyPy je implementace napsaná v *Pythonu* s JIT (Just-In-Time) překladačem. Je pro některé typy úloh, zejména ty, které intenzivně používají procesor a nezdřují se čekáním na jiné zdroje (paměť, disk, síť), výrazně rychlejší než *CPython*. Bohužel zde občas kulhá kompatibilita s některými C rozšířeními a může mít vyšší spotřebu paměti.

MicroPython je lehká implementace jazyka pro mikrokontroléry a různé zabudované systémy. Má nízkou spotřebu paměti, ale na druhou stranu podporuje pouze omezenou část standardní knihovny.

V tomto přehledu nesmíme zapomenout na jednu z poměrně rozšířených platform, kterou je interaktivní platforma *Jupyter Notebook*,². Na této platformě se používá odnož jazyka označovaná *IPython*, jejíž odchylka spočívá především ve vylepšeném interaktivním rozhraní. Tato platforma je používána zejména při analýze dat a v nejrozličnějších vědeckých projektech. Vzhledem k její rozšířenosti je maximální snaha implementovat do ní nové verze jazyka co nejrychleji, takže můžete novou verzi používat během týdnů či několika málo měsíců od jejího vydání.

V této učebnici se soustředím na nativní platformu jazyka *Python*, tedy na platformu, kterou jste podle výše uvedeného návodu instalovali, a to konkrétně na **verzi 3.14**, jež vyšla v říjnu roku 2024.

1.1.2 Dokumentace

Přítomnost dokumentace je nesmírně důležitá, protože *Python* přichází (ostatně jako většina současných platform) s obrovskou nabídkou možností, které si průměrný lidský mozek nedokáže zapamatovat. Proto jistě oceníte možnost rychlého získání potřebných informací jak o samotném jazyku, tak o instalovaných knihovnách. Navíc zde najdete i několik rad a doporučení.

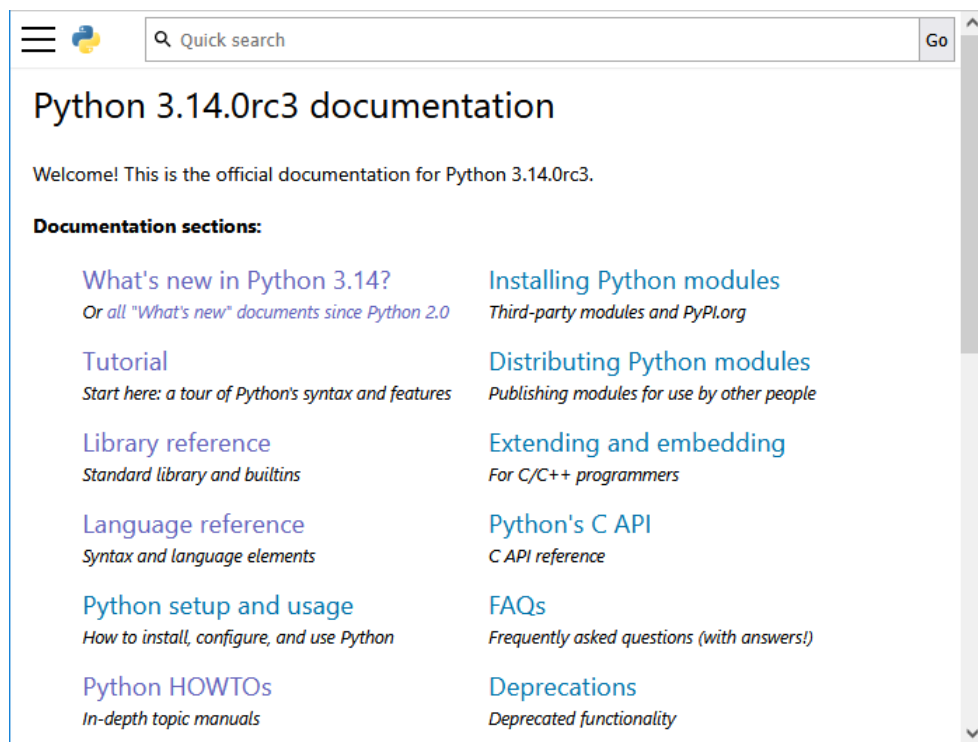
Pokud jste si v rámci instalace *Pythonu* instalovali i jeho dokumentaci (což vřele doporučuji), najdete ji v podsložce **Doc/html** složky, do níž jste instalovali *Python*. Z toho jste si jistě odvodili, že dokumentace je ve formátu HTML a můžete ji otevřít v libovolném prohlížeči.

² Název je zkratkou z názvů tří primárně zabudovaných jazyků: **Julia**, **Python** a **R**.

Jak se můžete přesvědčit na obrázku [1.1](#), hlavní stránka slouží jako taková křižovatka s přehledem oblastí, o nichž se můžete informovat. Kromě toho je nad nadpisem vstupní pole, do nějž můžete zadat svůj požadavek, a zabudované skripty vám pomohou najít vhodnou odpověď.

1.1.3 PEP

Dokumentace se velmi často odvolává na nejrůznější dokumenty, které jsou označované vždy zkratkou PEP a číslem. PEP je zkratka z anglického *Python Enhancement Proposal* (doporučení pro vylepšení *Pythonu*). Jsou to dokumenty poskytující informace komunitě *Pythonu* nebo navrhuující nové funkce, procesy nebo prostředí. Jejich přehled najdete na <https://www.python.org/dev/peps/>.



Obrázek 1.1:

Okno s dokumentací platformy ve verzi [3.14](#) Release candidate 3

1.1.4 Pracovní režimy

Při vývoji programů v *Pythonu* pracujete v jednom ze dvou režimů:

- V interaktivním režimu zadáváte příkazy, na něž systém okamžitě zareaguje, zadaný příkaz provede a na následujících řádcích zobrazí svoji odpověď.

- V editačním režimu vytváříte v nějakém textovém editoru zdrojový kód programu, který následně buď přímo, anebo v interaktivním režimu spustíte.

Při práci v *Pythonu* s uživatelem vždy komunikuje interpret. Ten však zadané příkazy hned neprovádí, ale nechá si je od překladače nejprve převést do interního kódu, který je pak možné interpretovat mnohem efektivněji než původní zdrojový text.

Při interpretaci interního kódu se navíc interpret může rozhodnout, že se některé části budou provádět opakovaně a že bude proto výhodné si je nechat přeložit stranou přímo do strojového kódu příslušného procesoru, a ten pak nechat provést maximální rychlostí.

1.1.5 Překladač versus interpret

Překladač a interpret *Pythonu* spolu těsně spolupracují, takže je někdy těžké rozlišit, kdo z nich je zodpovědný za příslušnou reakci (a přiznejme si, programátora to většinou ani nezajímá). Nebudu proto mezi nimi v dalším textu většinou rozlišovat a nebude-li evidentní, kdo z nich je za něco zodpovědný, budu pro ně používat hromadné označení *Python*.

1.2 Vývojová prostředí

Když se rozhodneme vyvíjet programy, měli bychom si nejprve rozmyslet, jaké budeme používat vývojové prostředí, tj. sadu nástrojů usnadňujících vývoj. Teoreticky je sice možné používat běžný textový editor a komunikovat se systémem prostřednictvím příkazového řádku, ale naprostá většina programátorů dává přednost komfortu vývojového prostředí.

Pro *Python* existuje řada integrovaných vývojových prostředí (Integrated Development Environment – IDE), která výrazně usnadňují práci, a to zejména na rozsáhlých projektech. Pojďme si představit nejpoužívanější vývojová prostředí používaná při vývoji programů v jazyce *Python*. Pokud byste chtěli vyzkoušet i některá další, zajímavý přehled najdete v anglické wikipedii pod heslem [*Comparison of integrated development environments*](#).

1.2.1 PyCharm a IntelliJ IDEA

Pravděpodobně nejrozšířenějším prostředím mezi profesionálními programátory je prostředí *PyCharm*, které vyvíjí pražská firma *JetBrains*. Standardní verze je placená, ale firma nabízí i *Community Edition*, která je pro výuku a soukromé použití zdarma a pro středně složité aplikace dostačuje.

Toto prostředí je postaveno nad platformou *IntelliJ IDEA*, která je základem stejnojmenného nejrozšířenějšího prostředí pro tvorbu programů v *Javě*, ale hojně se používá

pro vývoj programů v řadě dalších programovacích jazyků. Rozšíření o podporu *Pythonu* lze do prostředí *IntelliJ IDEA* nahrát i jako plugin. Proto mu zřejmě dají přednost programátoři, kteří již toto prostředí využívají pro tvorbu programů v jiném programovacím jazyce.

Obě verze zahrnují i podporu umělé inteligence, i když každá v různé síle. Základní nabídku inteligentního návrhu a generování kódu však poskytují obě.

Prostředí *PyCharm* můžete stáhnout na <https://www.jetbrains.com/pycharm/download>. Plugin do *IntelliJ IDEA*, který ocení všichni, kdo dané prostředí již používají, najdete na adrese <https://www.jetbrains.com/idea/download>.

1.2.2 Visual Studio Code

Prostředí *Visual Studio Code* nebo zkráceně *VS Code* vyvíjí *Microsoft* jako open-source. Toto IDE zřejmě upřednostní programátoři, kteří v něm již programují v jazyku *JavaScript* nebo v některém programovacím jazyku pro platformu .NET. Prostředí má nepřeberné množství pluginů pro nejrůznější jazyky včetně *Pythonu*, nicméně to primárně není vývojové prostředí, ale editor kódu umožňující zefektivnění práce prostřednictvím řady pluginů.

Mnozí jistě ocení, že do něj *Microsoft* začlenil svoji umělou inteligenci *Copilot* a umožnil její používání zdarma

Prostředí můžete stáhnout na adrese <https://code.visualstudio.com/>.

1.2.3 Jupyter Notebook a JupyterLab

Project Jupyter je nezisková organizace zabývající se vývojem open-source softwaru, otevřených standardů a služeb pro interaktivní výpočetní techniku. Její název vznikl jako akronym složený z názvů tří primárně podporovaných jazyků: *Julia*, *Python* a *R*.

Jupyter Notebook je webové prostředí pro vytváření dokumentů – poznámkových bloků (notebooků) *Jupyter*. *JupyterLab* je uživatelské rozhraní nové generace. Seznámit se s nimi můžete na adrese <https://jupyter.org/try>.

Jak už jsem řekl, toto prostředí je oblíbené při analýze dat a práci na různých vědeckých projektech, kdy uživatel ocení jeho interaktivitu a možnost prokládat programy běžným textem. Oceníte jej zejména v situaci, kdy potřebujete vytvářet jednoduché programy spouštějící kód definovaný v některé z rozšířených knihoven a potřebujete co nejjednodušeji sestavit různé přehledy a grafy. Je proto vhodný pro prototypování a experimentování.

Pro mnohé je ale důležité, že neposkytuje prakticky žádné nástroje pro refaktoriaci kódu a nehodí se pro vývoj netriviálních aplikací. Je vhodný pro výuku základních programových konstrukcí, ale na druhou stranu je naprosto nevhodný pro výuku vývoje netriviálních aplikací. Proto jej ve svých učebnicích a příručkách nepoužívám.

1.2.4 Řádkový editor a IDLE

Při používání nejrůznějších propracovaných vývojových prostředí bychom se však zejména na počátku zbytečně rozptylovali vstřebáváním základních pravidel jejich ovládání. Zvládnout dobře nějaké dokonalé vývojové prostředí totiž bývá složitější než zvládnout základy programovacího jazyka. Nemáte-li proto zkušenost s vývojovým prostředím, které podporuje i *Python*, má smysl si mezi prostředím začít vybírat až v okamžiku, kdy zvládnete základy jazyka a nejdůležitějších součástí standardní knihovny.

V této příručce se oněmi propracovanými vývojovými prostředími zabývat nebudu, protože demonstrační programy budou jednoduché, takže bychom jejich možnosti nevyužili. Využijeme naopak toho, že standardní instalace nabízí dvě základní vývojová prostředí:

- Řádkový editor (bývá označován zkratkou REPL) se spustí tehdy, spustíme-li *Python*, aniž bychom mu zadali program, který má spustit, nebo nějaký nestandardní požadavek.
- Prostředí IDLE je interaktivní vývojové prostředí s jednoduchým editorem, které je celé napsané v *Pythonu* a je součástí standardní knihovny (pokud jste při instalaci *Pythonu* jeho začlenění neodmítli), takže je můžeme spustit jako program v *Pythonu*.

Prostředí IDLE nabízí rozumný kompromis mezi jednoduchostí ovládání (a s ní spojenou rychlostí osvojení) a sadou nabízených funkcí. Jeho nejdůležitější výhodou je však to, že je součástí standardní instalace a nenutí vás proto instalovat cokoli dalšího. (Za chvíli vám obě základní prostředí představím.)

Výběr vývojového prostředí, které budete používat, je však zcela na vás, protože zdrojový kód bude nezávisle na použitém vývojovém prostředí shodný a shodné, nebo alespoň velmi podobné by měly být i odpovědi v interaktivním režimu. Jak si ukážeme ve výpisech [1.1](#) na straně a [1.5](#) na straně , bude se lišit nejvýše jejich formát.

1.3 Spuštění Pythonu

Příkaz potřebný ke spuštění *Pythonu* závisí na používaném operačním systému. Proberu proto samostatně pravidla pro *Windows* a pravidla pro prostředí založená na *Linuxu* (včetně *MacOS*).

1.3.1 Spouštění ve Windows

Ve *Windows* existují dva běžné způsoby, jak spustit *Python*. Rozdíl mezi nimi spočívá v tom, jak jsou oba příkazy implementovány a jaké verze *Pythonu* mohou spustit.

Pomocí příkazu **py**

Pokud jste při instalaci zaškrtnuli položku **py launcher** (viz příloha [A](#),³ obrázek [A.2](#)), můžete spustit speciální zavadač – *Python Launcher*, což je nástroj, který je umístěn do složky `C:\Windows\`, takže je dostupný bez nutnosti upravovat proměnné prostředí **PATH**. *Python Launcher* detekuje, které verze *Pythonu* jsou na vašem počítači nainstalovány, a dokáže automaticky delegovat spouštěcí příkaz na správnou verzi.

Ve výchozím nastavení použije nejnovější verzi *Pythonu* nainstalovanou na vašem počítači. Když jsem na svém počítači instaloval vývojové verze *Pythonu* (*alfa*, *beta*, *release candidate*), spouštěl příkaz **py** zadaný bez argumentů právě tyto verze, protože jsem zmiňovanou volbu v instalačním okně zaškrtnl. Když jsem chtěl spustit některou z minulých verzí, musel jsem zadat její číslo jako argument – např. **-3.13**, případně jiné obdobné číslo odpovídající verzi, kterou jsem chtěl spustit:

```
py -3.13
```

Pomocí příkazu **python**

Tento příkaz odkazuje na spustitelný soubor *Pythonu*, který byl vybrán jako výchozí během instalace *Pythonu*. Cesta k této verzi je uložena v proměnné prostředí **PATH**, takže je přímo dostupná.

Aby tomu tak bylo, musíte při instalaci zaškrtnout v okně **Setup** volbu **Add python.exe to PATH**. Protože jsem však při instalaci vývojových verzí tuto volbu nezaškrtnl (viz příloha [A](#), obrázek [A.1](#)), tak se při zadání příkazu **Python** spouštěla verze, jejíž cesta byla v proměnné **PATH** uvedena.

Na druhou stranu ale musíte při zařazení další verze hlídat, aby byla minulá verze ze seznamu v této proměnné odstraněna.

Pomocí příkazu **pyw**, resp. **pythonw**

Tyto programy nejsou určeny pro spouštění samotného *Pythonu*, ale pro spouštění programů používajících grafické uživatelské rozhraní (GUI). Při použití těchto verzí příkazu se neotevřít konzolové okno, protože se předpokládá, že program komunikuje s uživatelem prostřednictvím vlastních oken. Případné tisky na standardní nebo na standardní chybový výstup jsou proto ztraceny (samozřejmě pokud je program nepřesměruje jinam).

Zadáním některého z uvedených příkazů v adresním řádku Průzkumníka

Málo lidí ví o tom, že ve *Windows* je možné zadávat příkazy tak, že se zadají v adresním řádku *Průzkumníka*. Tento způsob se uplatní zejména tehdy, potřebujete-li zadat spouštěnému programu nějaké argumenty a nechce se vám pro daný účel definovat zástupce.

Abych neprovokoval uživatele jiných systémů, že je tu zbytečně velká část textu věnována *Windows*, umístil jsem příslušný výklad do přílohy [A Instalace a konfigurace Pythonu pod Windows](#).

³ Připomínám, že čtenáři papírové verze si musejí stáhnout přílohy z webové stránky knihy.

1.3.2 Spouštění v macOS a Linuxu

V těchto systémech se ke spouštění *Pythonu* standardně používá příkaz `python3`. Někde lze použít i samotné `python`, ale to na řadě systémů nefunguje, anebo spouští *Python 2.x*.

Některé distribuce dokonce vyžadují zadání úplné verze – např. `python3.14`. Co platí pro váš operační systém, si musíte vyzkoušet.

1.3.3 Co je nejdůležitější

Nezávisle na použitém operačním systému je třeba, abyste příkaz spouštějící interpret *Pythonu* zadávali ve složce (adresáři), v níž budete mít umístěné spouštěné skripty. Studenti na to často zapomínají, a pak se diví, proč *Python* tvrdí, že daný skript neexistuje.

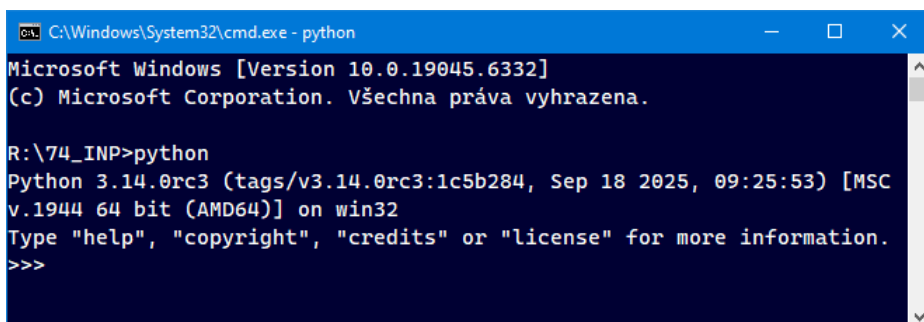
1.3.4 Spouštění Pythonu v demonstračních programech v knize

Já budu v této knize používat v demonstračních programech spouštění prostřednictvím zavaděče `py` (je to nejkratší). Vy pak nahraďte tento příkaz tím, který bude platit na vašem systému.

1.4 Řádkový editor – REPL

Všechna vývojová prostředí včetně tak jednoduchého, jakým je IDLE, poskytují okna či panely pro komunikaci s interpretem v interaktivním režimu. Představím vám proto základy komunikace s řádkovým editorem v okně příkazového řádku. Tento editor se aktivuje, spustíte-li *Python* bez argumentů. Tyto základy pak všechna zmíněná okna či panely vývojových prostředí nějakým způsobem přebírají.

Demonstraci takového spuštění ve *Windows* si můžete prohlédnout na obrázku [1.2](#).



```
C:\Windows\System32\cmd.exe - python
Microsoft Windows [Version 10.0.19045.6332]
(c) Microsoft Corporation. Všechna práva vyhrazena.

R:\74_INP>python
Python 3.14.0rc3 (tags/v3.14.0rc3:1c5b284, Sep 18 2025, 09:25:53) [MSC
v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
;>>>
```

Obrázek 1.2:

Okno příkazového řádku Windows se spuštěným řádkovým editorem (REPL) *Pythonu*

Řádkový editor bývá často označován zkratkou REPL. Tato zkratka se používá pro označení editorů z rodiny interaktivních programů a vychází z počátečních písmen anglického *Read-Evaluate-Print-Loop*, což bychom mohli přeložit do češtiny jako *cyklus přečti, vyhodnoť, vytiskni*. Touto zkratkou budu v dalším textu označovat tento program i já.

Po spuštění *Pythonu* bez argumentů se spustí REPL, který zobrazí úvodní řádky oznamující spuštění interpretu následované řádkem s výzvou (anglicky prompt) k zadání příkazů, jejichž prostřednictvím získáte další informace nebo spustíte zadanou akci.

Základní výzva je tvořená čtveřicí znaků „**>>>**“ (tři většítka následovaná mezerou). Když za ni něco napíšete a stisknete ENTER, interpret zadaný řádek zpracuje a zdá-li se mu, že jste zadali celý příkaz, vypíše na další řádek výsledek.

Zadáváte-li příkaz, který zabírá několik řádků, *Python* zahajuje každý další řádek **pokračovací výzvou** tvořenou čtveřicí znaků „**...**“ (tři tečky následované mezerou).

Výsledek vypisuje od začátku řádku, kdežto všechny řádky příkazů jsou uvozené počáteční nebo pokračovací výzvou. Výzvy budu ve výpisech programů podbarvovat jinou barvou než zbytek programu, abyste co nejsnadněji odlišili zadávaný kód od výzev programu.

Podoba zobrazeného textu trochu závisí na tom, používáte-li verzi 3.13 či mladší, anebo některou ze starších verzí. Od verze 3.13 totiž začal *Python* zobrazovat výzvy jinou barvou, aby se daly snadno odlišit od zadávaného textu.

1.4.1 Komentáře

I při maximální snaze o srozumitelnost zapsaného kódu se občas stane, že si u některé části nemůžete vzpomenout, proč jste ji do kódu zahrnuli, k čemu slouží a jak přesně se má používat. Jednou z cest, jak program učinit pochopitelnějším jiným programátorům (a po čase i sobě), je doplnit jej vysvětlujícími komentáři.

Text začínající znakem **#** (mříž) a pokračující až do konce řádku vnímá *Python* jako komentář a ignoruje jej. Jedinou výjimkou je výskyt znaku **#** uvnitř stringu (slangový výraz pro zadávaný text – podrobnosti viz kapitolu [3 Zadávání textů – stringů](#) na straně).

Řada programátorů komentáře ignoruje a nastavuje svá vývojová prostředí tak, aby byly komentáře co nejméně nápadné a při pročitání kódu je neobtěžovaly. Protože většinou vytvářím výukové programy, jejichž čtenáři dodatečné informace v komentářích často ocení, mám ve svých prostředích naopak nastaveno, že komentáře jsou podbarveny, aby se tím co nejlépe odlišily od okolního kódu.

Bude tomu tak i ve výpisech komunikace s prostředím a v definicích částí programu v této knize. To umožňuje snadno číst kód a ignorovat komentáře, anebo naopak se soustředit na vysvětlovací komentáře ve chvíli, kdy nám něco není jasné.

Ke komentářům se ještě vrátím v podkapitole [3.2 Komentáře](#) na straně .

1.4.2 Odsazování

Abych mohl vysvětlovat další speciality *Pythonu*, musím vás nejprve seznámit s některými rysy jazyka.

V *Pythonu* je (na rozdíl od většiny ostatních programovacích jazyků) důležité, jak moc daný řádek kódu odsadíte od levého kraje. Odsadíte-li jej méně nebo více, než je v danou chvíli potřeba, *Python* se vzbouří a ohlásí syntaktickou chybu.

Příkaz zadávaný za výzvou musí vždy začínat hned za výzvou bez jakýchkoliv úvodních mezer či tabulátorů. Ve výpisu 1.1 je na řádku 1 zadán součet tří čísel.

Pokud bychom však před naše zadání vložili byť jedinou mezeru, jak je naznačeno na řádku 3, *Python* ohlásí syntaktickou chybu *unexpected indent* (nečekané odsazení).

Výpis 1.1: *Komunikace s interpretem spuštěným v konzolovém okně*

```
1 >>> 1+2+3          # Jednoduchý výraz, jehož hodnotu chceme spočítat
2 6
3 >>> 0              # CHYBA! Řádek začíná mezerou
4 File "<unknown>", line 1
5     0
6 IndentationError: unexpected indent
7 >>>
```

1.4.3 Víceřádkové příkazy

Nevejde-li se nám celé zadání na řádek, musíme interpret včas varovat: buď tím, že otevřeme závorku, anebo tím, že na konec řádku vložíme znak \ (zpětné lomítko). Po stisku klávesy ENTER pak interpret přečte řádek, zjistí, že ještě není ukončen (otevřená závorka není uzavřena nebo řádek končí znakem \), a editor připraví další řádek, v němž vypíše pokračovací výzvu tvořenou třemi tečkami a mezerou.

Když při čtení řádku pozná, že zadání skončilo (před ukončením řádku byla případná závorka uzavřena a řádek nekončí zpětným lomítkem), tak celý několikařádkový blok textu přeloží a zpracuje a na následujícím řádku (u delšího výsledku na následujících řádcích) vypíše případný výsledek.

Končí-li řádek komentářem, tak se koncová zpětná lomítka neuplatní, protože od znaku # až do konce řádku je *Python* ignoruje, a to včetně případného závěrečného zpětného lomítka.

Celý koncept víceřádkových příkazů je založen na principu fyzických a logických řádků popsaném v podšeděném okně [Fyzické a logické řádky](#). Kód uplatňující popsaná pravidla si můžete prohlédnout ve výpisu 1.2.

Zde si na řádku 9 všimněte, že zpětné lomítko zadané na konci komentáře se neuplatní (*Python* nezobrazil pokračovací výzvu, ale hned vypsál výsledek), protože (jak jsme si řekli) všechny znaky od znaku # do konce řádku ignoruje a s nimi i závěrečné zpětné lomítko.

Výpis 1.2: Dva způsoby zadání pokračovacích řádků

```

1 >>> (45          # První řádek víceřádkového příkazu
2 ...     + 67      # Druhý fyzický řádek je pokračování logického
3 ...     + 89      # Pokračovacích řádků může být libovolný počet
4 ...     ) - 100   # Poslední z fyzických řádků tvořících jeden logický
5 101
6 >>> 45 \
7 ...     + 67\
8 ...     + 89\
9 ...     ) - 100   # Zpětné lomítko v komentáři Python ignoruje \
10 101
11 >>>

```

Fyzické a logické řádky

Syntaxe *Pythonu* rozlišuje fyzické a logické řádky.

- Fyzický řádek je řádek textu končící znakem konce řádku.
- Logický řádek může sestávat z několika fyzických řádků, přičemž pokračování logického řádku vzniknou v těchto případech:
 - Předchozí fyzický řádek končí znakem `\` (zpětné lomítko), který není součástí komentáře ani stringu.
Teoreticky by za zpětným lomítkem měl bezprostředně následovat konec řádku, ale pokud se mezi lomítko a konec řádku vloudí nějaké mezery, *Python* je bude ignorovat.
 - Na některém z předchozích řádků byla otevřena závorka, která ještě nebyla uzavřena. Za takovouto „závorku“ se považuje i trojice apostrofů či uvozovek uzavírající víceřádkový string (viz pasáž [3.1.1 Víceřádkové stringy](#) na straně).

Python sloučí úvodní fyzický řádek se všemi jeho pokračovacími řádky do jednoho logického řádku. Pravidla pro odsazování platí pouze pro logické řádky, přičemž počáteční odsazení pokračovacích řádků *Python* ignoruje.

1.4.4 Dvě verze řádkového editoru

Autoři verze 3.13 modifikovali řádkový editor (REPL) tak, aby usnadňoval některé operace a aby se současně výpis zpřehlednil. Pro výklad některých funkcionalit tohoto editoru je ale potřeba znát základy *Pythonu*, proto budu často uvádět dopředné odkazy pro ty, kteří na *Python* přecházejí z jiného programovacího jazyka.

Barevné rozlišení části textu

Nový editor používá barvy (samozřejmě pokud to operační systém u příkazového řádku podporuje). Výzvu nyní tiskne jinou barvou než následující text a odpovědi

Pythonu. Svou vlastní barvu mají i výpisy zásobníku návratových adres (viz podšeděný rámeček [Zásobník návratových adres – ZNA](#) na straně) po vyvolání výjimky (viz podkapitulu [18.3 I chyby jsou objekty – výjimky](#) na straně).

Není to sice plnohodnotné zvýrazňování syntaxe, ale i taková drobnost pomůže orientaci v zadávaném kódu.

Nepříjemné je, že na černém pozadí, které používá většina programátorů, jsou barvy výzev a chybových hlášení špatně vidět. V době psaní tohoto textu nebylo možné nastavit vlastní. Bylo však možné barvy vypnout, a to tak, že jste před spuštěním *Pythonu* nastavili systémovou proměnnou `PYTHON_COLORS` na nulu.

Ve *Windows* k tomu musíte ve standardním příkazovém řádku zadat před vlastním spuštěním *Pythonu* příkaz

```
set PYTHON_COLORS=0
```

používáte-li *PowerShell*, tak

```
$env:PYTHON_COLORS = 0
```

v *Linuxu* a *macOS* pak příkaz

```
export PYTHON_COLORS=0
```

Tím nastavíte systémovou proměnnou `PYTHON_COLORS`, jejíž hodnotu *Python* před spuštěním inovovaného editoru kontroluje.

Práce se složenými příkazy – řádkový versus příkazový režim editoru

Nejdůležitější změnou je to, že nyní můžete vkládat a editovat text po blocích (přesněji řečeno po příkazech), takže v případě víceřádkových příkazů včetně složených (ty bývají většinou víceřádkové – viz podkapitulu [8.6 Složené příkazy a odsazování](#) na straně), můžete postupně vložit celý příkaz a dokud jej nedokončíte, můžete jej editovat včetně slučování a rozdělování řádků.

Navíc při zadávání příkazů řádkový editor sám rozpoznává složené příkazy a automaticky vloží na začátek dalšího řádku odpovídající počet mezer.

Když chcete upravit některý ze dříve zadaných příkazů, tak se šipkou nahoru a dolů nepřesouváte po jednotlivých fyzických řádcích, jak tomu bývá v konzolových oknech zvykem, ale opět po celých příkazech, které pak můžete opravit a zadat znovu. Tím se alespoň částečně přibližujete k možnostem prostředí *IDLE*.

Tento nový režim práce bychom mohli nazvat **příkazový režim** (vstup se zpracovává po celých příkazech), kdežto režim, v němž pracoval editor ve starších verzích *Pythonu*, bychom mohli označit jako **řádkový režim**, protože vstup se zpracovával po řádcích a chyba ve víceřádkovém příkazu se dala opravit pouze tak, že se tento příkaz začal zadávat znovu řádek za řádkem.

Vkládání kódu ze schránky v příkazovém režimu – vkládací výzva

Automatické odsazování se, bohužel, uplatní i při vkládání textu ze schránky. Máte-li proto příkazy v těle složeného příkazu odsazené, budou odsazené dvakrát. Editor si toto odsazení zapamatuje, takže další příkaz již bude odsazen třikrát – viz výpis [1.3](#).

Bohužel, stejně se chová i při vkládání kódu ze schránky, chceme-li vyzkoušet kus kódu zkopírovaný přes schránku z vašeho editoru. Výsledek pak vypadá pro nás nehezky a pro *Python* je v řadě případů nepřijatelný a chrlí chybová hlášení.

Záchranu poskytuje stisk klávesy F3 (aby to fungovalo, tak řádek, ve kterém tisknete klávesu F3, musí být v danou chvíli prázdný), která přepíná režim zpracování vstupu a současně ihned změni výzvu na „(paste)“ (uzávorkovaný text následovaný mezerou). Pak můžete vložit požadovaný text ze schránky.

Protože je výzva „(paste)“ určena pro vkládání kódu ze schránky, budu jí říkat **vládací výzva**.

Problémy při vkládání textu ze schránky jsem se snažil ukázat a vysvětlit ve výpisu [1.3](#), kde jsem ze schránky vkládal první výraz (řádky [1-4](#)) z výpisu [1.2](#) na straně .

Musíte ale vzít v úvahu, že řádky, kde komentář říká, že jsem stiskl klávesu F3, byly při stisku klávesy prázdné a uvedený komentář jsem tam vložil (je jedno, jestli ze schránky či přímo) až poté, co editor změnil podobu výzvy.

Výpis 1.3: Špatně odsazovaný kód vkládaný ze schránky

```

1 >>> (45      # První řádek víceřádkového příkazu
2 ...      + 67  # Druhý fyzický řádek je pokračování logického
3 ...      + 89  # Pokračovacích řádků může být libovolný počet
4 ...      ) - 100 # Poslední z fyzických řádků tvořících jeden logický
5 101
6 (paste) # Změna výzvy po stisku klávesy F3
7 (paste) (45      # První řádek víceřádkového příkazu
8 (paste)      + 67  # Druhý fyzický řádek je pokračování logického
9 (paste)      + 89  # Pokračovacích řádků může být libovolný počet
10 ...      ) - 100 # Poslední z fyzických řádků tvořících jeden logický
11 ...
12 101
13 >>> # Stisk klávesy na předminulém řádku změnil dvě po sobě jdoucí výzvy
14 (paste) # Znovu změna výzvy po stisku klávesy F3
15 (paste) (45      # První řádek víceřádkového příkazu
16 (paste)      + 67  # Druhý fyzický řádek je pokračování logického
17 (paste)      + 89  # Pokračovacích řádků může být libovolný počet
18 (paste)      ) - 100 # Poslední z fyzických řádků tvořících jeden logický
19 ...
20 ...
21 101
22 >>> # Stisk F3 na předminulém řádku znovu změnil dvě po sobě jdoucí výzvy
23 >>>
```

Na řádcích [1-4](#) je vidět zobrazení vloženého kódu totožného s kódem prvních čtyř řádků zadaných ve výpisu [1.2](#) na straně .

Na řádcích [2-4](#) je vidět, že editor na začátku řádku zopakoval odsazení z minulého řádku, a poté vložil příslušný řádek ze schránky. Tím se ale k odsazení vloženému editorem přidalo odsazení ze zdrojového textu.

Řádek [1](#) nebyl odsazený, takže řádek [2](#) je odsazený správně. Na řádku [3](#) se ale ke čtyřem mezerám vloženým editorem přidaly čtyři mezery ze zkopírovaného řádku.

Obdobná situace je na řádku 4, kde se k osmi mezerám vloženým editorem přidaly čtyři mezery ze zkopírovaného řádku.

Zobrazená podoba zadaného kódu se nám možná nelíbila, ale neobsahovala žádnou chybu. *Python* proto spočítal zadaný výraz a zobrazil jej na řádku 5, načež na řádku 6 zobrazil standardní výzvu „>>>“ (není vidět, změnila se).

Tato výzva se po stisku klávesy F3 změnila na vkládací výzvu „(paste)“, která je na řádku 6 ve výpisu zobrazena a která editor přepnula do vkládacího režimu.

Za tuto výzvu jsem zapsal komentář na řádku 6 a za něj znovu vložil stejný kód jako výše. Na řádcích 7–10 se nyní zobrazil kód správně odsazený. Po vložení posledního řádku se zobrazil prázdný řádek 11, na jehož začátku se stále zobrazovala vkládací výzva „(paste)“. REPL očekával zadání dalšího kódu, ať už ručně, nebo ze schránky.

Stiskl jsem znovu F3. REPL z toho odvodil, že zadávání příkazu skončilo, nahradil vkládací výzvy „(paste)“ na řádcích 10 a 11 pokračovacími výzvami „...“ a čekal, co budu zadávat dál.

Protože je však pokračovací výzva o 3 znaky kratší, tak byl kód na řádku 10 zdánlivě posunutý. Říkám zdánlivě, protože kód na řádku 19 začíná čtyřmi mezerami, stejně jako kódy na dvou řádcích předchozích. Pro *Python* tedy posunuté nejsou. Jsou posunuté pouze pro nás, protože řádek 10 začíná o čtyři znaky dříve než řádky předchozí.

Jak jsem řekl, REPL čekal, co budu zadávat dál. Stiskl jsem ENTER, REPL předal celý zadaný blok *Pythonu*, který jej vyhodnotil, REPL ho na řádku 12 vypsal a na následujícím řádku (13) zobrazil opět standardní výzvu „>>>“.

Zde jsem vložil komentář upozorňující na to, že stisk klávesy F3 na řádku 11 změnil dvě po sobě jdoucí výzvy – tu na řádku, na němž jsem stiskl F3 (řádek 11), a také tu na řádku předchozím.

Na řádku 14 jsem znovu stiskl klávesu F3, čímž se výzva znovu změnila na pokračovací výzvu „(paste)“ a nastavil se vkládací režim. Zapsal jsem komentář oznamující změnu výzvy a poté jsem ze schránky vložil stejný kód jako minule. Když ale po vložení kódu skončil kurzor na řádku 19, stiskl jsem ještě jednou ENTER a na řádku 20 stiskl F3.

Po změně dvou posledních výzev jsem znovu stiskl ENTER, abych oznámil konec zadání. REPL znovu předal zadaný blok *Pythonu*, jenž jej vyhodnotil, REPL vytiskl na řádku 21 výsledek a zobrazil na řádku 22 standardní výzvu „>>>“.

Opět jsem vložil komentář upozorňující na změnu dvou výzev.

Když se nyní podíváte na řádky 19–20 se změněnými výzvami, zjistíte následující: protože jsou oba předchozí řádky prázdné, tak v nich nebylo co posouvat a estetický dojem z programu to uživateli neruší (snad až na ten „zbytečný“ prázdný řádek).

Tisk hodnoty pouze posledního výrazového příkazu

Je-li na řádku zadáno více výrazových příkazů, tiskne nový REPL pouze hodnotu posledního z nich. Totéž platí o situaci, kdy se vyskytne více výrazových příkazů ve vkládacím režimu. Po skončení vkládacího režimu se vytiskne pouze poslední z nich. Vše jsem se pokusil demonstrovat ve výpisu [1.4](#).

Výpis 1.4: *Více výrazových příkazů na řádku či ve vkládacím bloku*

```
1 >>> 1; 2; 3
2 3
3 (paste) 11
4 (paste) 22
5 ... 33
6 ...
7 33
8 (paste) print(111)
9 (paste) print(222)
10 ... print(333)
11 ...
12 111
13 222
14 333
15 >>>
```

Na řádku 1 jsem zadal tři výrazové příkazy. Jak vidíte na řádku 2, nový editor vytiskl pouze poslední z nich (ten předchozí tiskne postupně všechny tři hodnoty, každou na samostatný řádek).

Na řádku 3 jsem stiskl klávesu F3. Tím se režim práce přepnul do vkládacího a standardní výzva byla nahrazena vkládací výzvou. Postupně jsem zadal tři hodnoty, na každý řádek jednu.

Na řádku 5 jsem znovu stiskl klávesu F3. Tím jsem přepnul zpět do standardního režimu, čímž se výzva na řádcích 5-6 změnila na pokračovací. Po stisku ENTER se na řádku 7 vytiskla hodnota naposledy zadaného výrazového příkazu a na následujícím řádku 8 standardní výzva.

Na řádku 8 jsem opět stiskl klávesu F3 a opět tak přepnul režim práce do vkládacího. Standardní výzva byla nahrazena vkládací výzvou. Postupně jsem zadal tři příkazy `print`, na každý řádek jeden.

Na řádku 11 jsem znovu stiskl klávesu F3. Tím jsem přepnul zpět do standardního režimu, čímž se výzva na řádcích 10-11 změnila zpět na pokračovací. Po stisku ENTER se na řádku 12-14 postupně vytiskly hodnoty tištěné jednotlivými příkazy a na následujícím řádku 15 se opět zobrazila standardní výzva.

Jak odmítnout práci v příkazovém režimu řádkového editoru

To, jestli bude výše popsané vylepšení opravdu sloužit k zefektivnění vaší práce, záleží na tom, v jakém režimu s řádkovým editorem pracujete.

Pokud do něj příkazy pro svoje experimenty s kódem vkládáte ručně a strukturovaný příkaz do něj ze schránky vkládáte spíše výjimečně, mohl by vám tento nový režim práce řádkového editoru pomoci zvýšit efektivitu.

Pokud naopak jako já připravujete své programy převážně v nějakém editoru a většinou je zkopírujete ze schránky do editoru, abyste vyzkoušeli funkčnost daného kódu, a pokud tento kód potřebuje opravu, opravíte ji v editoru a znovu zkopírujete přes schránku, bude vás nutnost přepínání mezi dvěma režimy brzy obtěžovat.

Samozřejmě, že IDE, které používám (přiznám se, je to *PyCharm*), umožňuje spouštět testovací programy mnohem efektivněji, ale problém je v tom, že do spolupráce s vámi občas vkládá zbytečně moc umělé inteligence, která je nakonec kontraproduktivní.

Již několikrát se mi stalo, že jsem nějaký svůj doprovodný program chtěl spustit v IDE, ale to mi začalo tvrdit, že v programu je nějaká mně neznámá chyba. Když jsem pak též program bez jediné změny spustil v prostoduchém řádkovém editoru, prošel bez problémů. Nepodařilo se mi přijít na to, co je příčinou tohoto nesouladu.

Zřejmě jsem nebyl sám, protože dokumentace vysvětluje, že když mi vylepšený řádkový editor nebude vyhovovat, mohu pustit standardní řádkový editor převzatý z minulých verzí, který pracuje pouze v řádkovém režimu. Ve *Windows* k tomu musíte zadat před spuštěním *Pythonu* z příkazového řádku příkaz

```
set PYTHON_BASIC_REPL=1
```

používáte-li *PowerShell*, tak

```
$env:PYTHON_BASIC_REPL = 1
```

v *Linuxu* a *macOS* pak příkaz

```
export PYTHON_BASIC_REPL=1
```

Tím vytvoříte systémovou proměnnou, kterou *Python* před spuštěním inovovaného editoru kontroluje, aby zjistila, který editor má spustit. Spustíte-li starší verzi editoru, bude barevné zvýraznění automaticky vypnuto, protože je předchozí verze nenabízela.

Měl bych ještě upozornit, že na nastavené hodnotě nezáleží, *Python* pouze kontroluje, zda taková systémová proměnná existuje. Pokud byste se chtěli vrátit k vylepšenému REPL, tak musíte danou proměnnou odstranit. Ve standardním příkazovém řádku *Windows* tak učiníte příkazem

```
set PYTHON_BASIC_REPL=
```

v *PowerShell* příkazem

```
Remove-Item Env:PYTHON_BASIC_REPL
```

a v *Linuxu* a *macOS* příkazem

```
unset PYTHON_BASIC_REPL
```



Editor použitý pro výpisy

Vzhledem ke svému výše popsanému režimu práce budu v příštích výpisech zaznamenávat své seance s řádkovým editorem převzatým z minulých verzí, tj. budu mít nastavenou systémovou proměnnou `PYTHON_BASIC_REPL=1`, protože mi pak zobrazené výsledky připadají přehlednější.

1.5 Spuštění skriptu napsaného v Pythonu

Možnosti spuštění skriptů zapsaných v *Pythonu* bohužel silně závisí na operačním systému, v němž pracujete. Jedno ale mají všechny operační systémy společné: ve všech systémech lze skript napsaný v jazyku *Python* spustit tak, že z příkazového řádku zavoláte *Python* a předáte mu název souboru se skriptem jako argumentem.

Můžete si to hned vyzkoušet. V sadě doprovodných programů najdete ve složce **74_INP** soubor **m01a_script.py**. Otevřete okno příkazového řádku, v něm otevřete tuto složku a zadejte v závislosti na operačním systému odpovídající z příkazů:

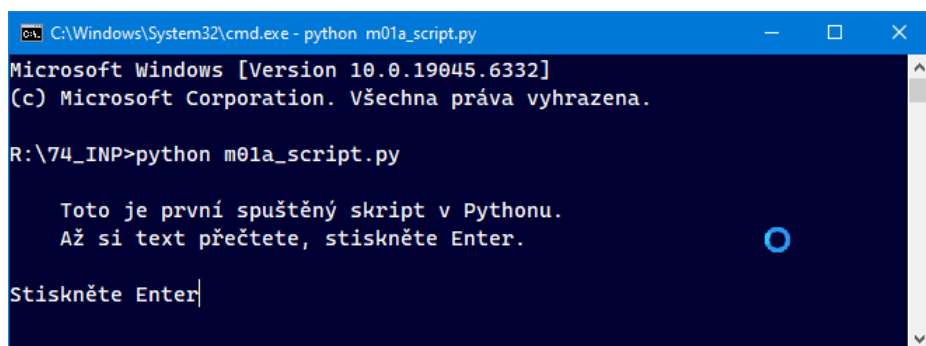
```
py m01a_script.py
python m01a_script.py
python3 py m01a_script.py
```

První dva by měly fungovat ve *Windows*, třetí na *Linuxu* a *macOS*. Po zadání příkazu by se měl vypsát text

```
Toto je první spuštěný skript v Pythonu.
Až si text přečtete, stiskněte Enter.
```

Stiskněte Enter

obdobně, jako je tomu na obrázku [1.3](#).



Obrázek 1.3:

Text zobrazený skriptem **m01a_script**

Když si text přečtete a stisknete ENTER, řízení se vrátí operačnímu systému a vypíše se další výzva – ve výše uvedeném případě výzva **R:\74_INP>**.

Takto by to mělo chodit na všech operačních systémech. Když jsem hovořil s uživateli pracujícími na *Linuxu* nebo *macOS*, tak mi všichni shodně tvrdili, že jiný postup ani nepoužívají. Že je sice možné modifikovat nastavení tak, aby šel daný skript spustit poklepáním, ale že se z jejich pohledu vynaložená námaha nevyplatí.

1.5.1 Windows

Uživatelé operačního systému *Windows*, kterých bývá mezi čtenáři mých knih tak 85 %, jsou však zhyčkaní a nejraději by skripty spouštěli pouhým poklepáním.

Mají-li *Python* správně nainstalován, mělo by opravdu stačit poklepat na daný soubor. Tím se automaticky spustí *Python*, otevře konzolové okno, v němž vypíše pouze hlášku programu se závěrečnou výzvou ke stisku tlačítka ENTER. Po jeho stisku se okno zavře a systém se vrátí do původního stavu.

Nevýhodou tohoto řešení je, že skriptu nelze zadávat žádné argumenty. Používáte-li některé argumenty často, můžete definovat příslušného zástupce. Pokud argumenty střídáte a nechce se vám pro každou jejich kombinaci definovat nového zástupce, musí se skript po svém spuštění na jejich hodnoty doptat.

1.6 Prostředí IDLE

Jak už jsem naznačil, prostředí IDLE je pro úvodní kroky asi nejvhodnější, protože je za prvé součástí základní instalace, takže už je nemusíte instalovat, a za druhé je ze všech nejjednodušší. Navíc je pro návrh jednodušších programů, jakými bude většina doprovodných programů, naprosto postačující.

Prostředí bylo pojmenováno podle Erika Idlea, jednoho ze zakládajících členů skupiny *Monty Python*, podle níž byl pojmenován celý jazyk. Později ale bylo rozhodnuto, že mu bude přidělen profesionálnější název. Nepřejmenovali jej, jen název jinak interpretují – nyní je oficiálně zkratkou z anglického *Integrated Development and Learning Environment* (Integrované vývojové a učební prostředí).

Předpokládám, že většina z vás bude používat nějaké dokonalejší IDE. Nicméně základní funkcionalitu, kterou můžeme u libovolného IDE předpokládat, lze na IDLE snadno demonstrovat, a na vás už pak zbyde jenom zjistit, jak se tato funkcionalita ve vašem IDE aktivuje.

Pojďme si proto IDLE ve stručnosti představit.

1.6.1 Spuštění IDLE

Zdrojový kód prostředí IDLE je ve složce `#/Lib/idlelib/`, kde symbol `#` zastupuje složku, kam jste instalovali *Python*. Najdete jej v souboru `idle.py`, používáte-li *Windows*, tak také v souboru `idle.pyw`.

IDLE spustíte např. tak, že výše popsaným způsobem spustíte *Python*, kterému předáte argument `-m idlelib.idle`, tedy například ve *Windows*:

```
pyw -m idlelib.idle
```

v *Linuxu* a *macOS* pak příkaz

```
python3 -m idlelib.idle
```

Nezávisle na použitém operačním systému je třeba, abyste příkaz spouštějící IDLE zadávali ve složce, v níž budete mít umístěné spouštěné skripty. Můžete si k tomu vytvořit dávkový soubor, v němž nastavíte příslušnou složku a zadáte popsany příkaz. Tento soubor pak můžete spouštět odkudkoliv.

Alternativa pro Windows

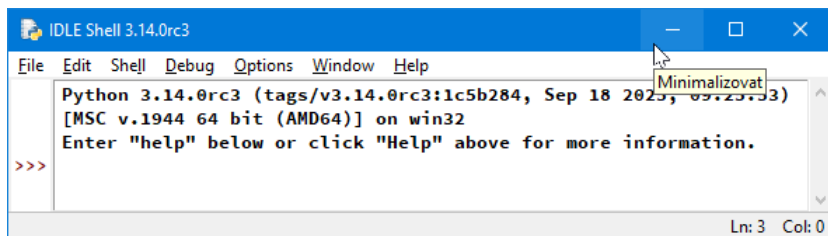
Patříte-li mezi čtenáře pracující ve *Windows*, můžete zkusit alternativní možnosti:

- Můžete zadat příslušný příkaz do adresního řádku *Průzkumníka* (nejdříve se samozřejmě musíte přesunout do příslušné složky).
- Můžete si vytvořit zástupce způsobem popsaným v příloze v podkapitole [B.2 Nastavování zástupce spouštějícího IDLE](#) (připomínám, že čtenáři tištěné verze příručky najdou soubor s přílohami na webové stránce knihy).

Můžete k tomu využít můj soubor `!IDLE_Python_314.lnk`. Tento zástupce je ale nastaven pro můj počítač, takže si jej budete muset otevřít a upravit podle uspořádání na vašem počítači. Vše je popsáno ve zmíněné podkapitole.

1.6.2 Základní popis

Prostředí IDLE (viz obrázek [1.4](#)) je multiokenní prostředí, jehož okna mohou pracovat v jednom ze dvou režimů:



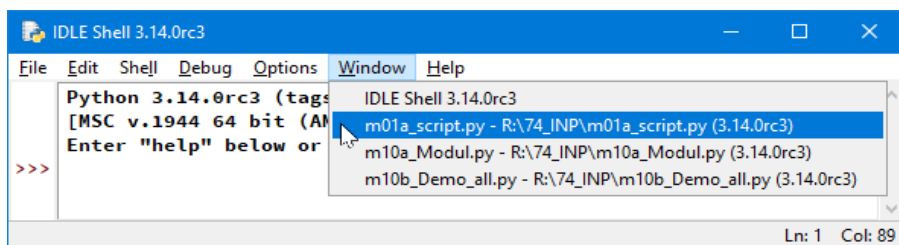
Obrázek 1.4:
Okno vývojového prostředí IDLE

- Jedno z oken může pracovat v příkazovém, interaktivním režimu, v němž uživatel komunikuje přímo se systémem, tj. s interpretem jazyka *Python*. V dalším textu jej budu označovat jako **příkazové okno**.
- Ostatní okna mohou pracovat pouze v editačním režimu, v němž editujete zdrojové či datové soubory, které pak v příkazovém okně použijete. Budu je proto označovat jako **editační okna**.

Režim aktuálního okna poznáte podle titulkové lišty. Příkazové okno v ní má uveden text **IDLE Shell**, následovaný verzí programu. Editací okno v ní uvádí název otevřeného souboru následovaný pomlčkou, úplnou cestou k danému souboru a v kulatých závorkách uvedenou verzi *Pythonu*.

Režim otevřeného okna ovlivňuje i hlavní nabídku (nabídkovou lištu). Příkazové okno má jako třetí položku nabídku **Shell** a jako čtvrtou položku nabídku **Debug** (viz obrázek 1.5), zatímco editační okno má místo nich nabídky **Format** a **Run**.

Mezi aktuálně otevřenými okny se přepínáte buď jejich aktivací na příkazovém panelu operačního systému (jednotlivá okna se zde vydávají za samostatné aplikace), anebo výběrem požadovaného cílového okna v nabídce **Window** (viz obrázek 1.5).



Obrázek 1.5:

*Výběr požadovaného cílového okna v nabídce **Windows***

Při prvním otevření bude mít okno takovou velikost, aby se v něm zobrazilo 40 řádků po 80 znacích. Zadáním příkazu **Zoom Height** nebo stiskem klávesové zkratky **ALT+2** zvětšíte výšku okna na velikost displeje, opětovným zadáním tohoto příkazu je vrátíte do výchozí velikosti, a to i v případě, kdy mělo před zvětšením jinou velikost.

Dáváte-li přednost jiné výchozí velikosti okna, můžete ji nastavit v dialogovém okně **Settings** vyvolaném zadáním příkazu **Option → Configure IDLE**. Podrobněji již možnosti nastavení tohoto programu prozatím vysvětlovat nebudu; zájemci si jistě poradí sami, i když třeba s trochou experimentování. Jenom bych upozornil, že některá nastavení (mezi nimi právě výchozí velikost okna) se uplatní až po zavření a následném spuštění programu.

Protože vím, že většina programátorů dává přednost tmavé barvě pozadí a světlému písmu, neboť mnohem méně oslňuje, tak bych zájemce navedl v nastavovacím okně na kartu **Highlights**, kde si v pravé části mohou nastavit tmavé téma a v případě potřeby je i upravit na téma vlastní.

1.6.3 Příkazové okno

Nenastavíte-li ve výše zmíněném okně **Settings** spuštění v editačním okně, tak se po spuštění aplikace otevře příkazové okno (viz obrázek 1.4), v němž se zobrazí úvodní řádek definující verzi *Pythonu* a použitý procesor následovaný řádkem s nápovědou a řádkem s výzvou k zadání příkazů, jejichž prostřednictvím získáte další informace nebo spustíte zadanou akci.

Až do verze 3.9 byla výzva standardní součástí zobrazovaného textu – stejně jako je tomu u editoru v konzolovém okně (viz obrázek 1.1). Na rozdíl od příkazového panelu se v příkazovém okně nevypisovala pokračovací výzva, což občas způsobovalo problémy, o nichž jsem hovořil v předchozích verzích této příručky.

Od verze 3.10 je proto pro výzvy vyhrazen vlevo speciální panel. Vedle standardní výzvy se v něm zobrazuje i pokračovací výzva. Od zobrazení v konzolovém okně se zobrazení v příkazovém okně liší tím, že odsazené jsou nyní i odpovědi systému na vaše příkazy, protože komunikace s uživatelem probíhá v pravém panelu. Levý panel s výzvami je pouze informační.

Ve výpisu [1.5](#) si můžete prohlédnout zopakovanou sadu příkazů z výpisů [1.1](#) na stránce (včetně komentářů probraných před tím v pasáži [1.4.1 Komentáře](#)) a prvního příkazu z výpisu [1.2](#) na straně zobrazenou přibližně tak, jak by ji zobrazilo IDLE. Můžete tak porovnat, jak se tyto dva způsoby záznamu komunikace v interaktivním režimu liší.

Výpis 1.5: Podoba výpisu v prostředí IDLE

```

1 >>> 1+2+3      # Jednoduchý výraz, jehož hodnotu chceme spočítat
2          6
3 >>> 0          # CHYBA! Řádek začíná mezerou
4 ...
5 SyntaxError: unexpected indent
6 >>> (45
7 ...      + 67
8 ...      + 89
9 ...      ) - 100
10      101
11 >>>
```

Z mého pohledu se s IDLE pracuje většinou lépe, ale na druhou stranu mi připadá, že zobrazení řádkového editoru je informativnější. Navíc v IDLE nelze zadat sadu příkazů ze schránky, protože je ochotno zpracovat vždy jen jeden příkaz, kdežto řádkový editor je schopen zpracovat celou sérii příkazů zadaných ze schránky. Proto budu ve zbytku příručky používat zobrazení řádkového editoru.

Restart interaktivního režimu

V interaktivním režimu je občas výhodné smazat všechny výsledky pokusů a začít znovu. Ukázky ve výpisech jsou většinou koncipovány tak, že žádný restart nevyžadují a mnohé na sebe dokonce navazují. Pokud by někdy byl potřeba restart, vždy na to výslovně upozorním.

V IDLE se restart spouští příkazem **Restart Shell** v nabídce **Shell**, anebo stiskem klávesové zkratky **CTRL+F6**.

Po zadání tohoto příkazu se za text vloží řádek s čarou z rovnítek přerušenou uprostřed textem **RESTART: Shell**. Od tohoto okamžiku se interpret chová stejně, jako kdybyste jej právě spustili.

```
===== RESTART: Shell =====
```

V **řádkovém editoru** není plynulý restart typu IDLE možný, protože interpret běží jako aplikace pod operačním systémem. Musíte proto nejprve zadat příkaz

```
exit()
```