

Začínáme programovat v jazyku PYTHON

3., rozšířené a aktualizované vydání

- » Nepředpokládá žádné předchozí znalosti programování
- » Výklad je postaven na vybudování jednoduché aplikace a průběžném seznamování s potřebnými konstrukcemi
- » Neomezuje se na výuku kódování v *Pythonu*, ale učí, jak program navrhnout a postupně vyvinout a rozchodit
- » Učí čtenáře programovat podle moderních zásad a metodik



edice
začínáme s ...

Začínáme programovat v jazyku

PYTHON

3., rozšířené a aktualizované vydání

RUDOLF PECINOVSKÝ

GRADA Publishing



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**.

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.

Rudolf Pecinovský

Začínáme programovat v jazyku Python

Třetí, rozšířené a aktualizované vydání

Vydala Grada Publishing, a.s.

U Průhonu 22, Praha 7

obchod@grada.cz, www.grada.cz

tel.: +420 234 264 401

jako svou 9364. publikaci

Odpovědný redaktor: Petr Somogyi

Fotografie na obálce Depositphotos/novotnyfi

Grafická úprava a sazba Rudolf Pecinovský

Počet stran 400

Třetí, rozšířené a aktualizované vydání, Praha 2024

Vytiskly Tiskárny Havlíčkův Brod, a.s.

© Grada Publishing, a.s., 2024

Cover Design © Grada Publishing, a. s., 2024

Cover Photo © Depositphotos/novotnyfi

Názvy produktů, firem apod. použité v knize mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

ISBN 978-80-271-7447-8 (ePub)

ISBN 978-80-271-7446-1 (pdf)

ISBN 978-80-271-5467-8 (print)

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

Stručný obsah

O autorovi	21
Poděkování	22
Úvod	23
Část A Superzáklady	31
Kapitola 1 Předehra	32
Kapitola 2 Vývojová prostředí	46
Kapitola 3 Zadávání hodnot a proměnné	55
Kapitola 4 Výrazy, příkazy a používání funkcí	70
Kapitola 5 Práce s objekty a nápověda	80
Kapitola 6 Moduly	88
Kapitola 7 Balíčky	101
Část B Začínáme programovat	111
Kapitola 8 Knihovny, robot Karel a příkaz with	112
Kapitola 9 Definice funkcí	128
Kapitola 10 Rozhodování	145
Kapitola 11 Opakování kódu, cykly	159
Kapitola 12 Kontejnery	174
Kapitola 13 Práce s kontejnery	184
Kapitola 14 Ošetřování chyb	199
Kapitola 15 Vytváříme aplikace	210
Část C Základy OOP	217
Kapitola 16 Úvod do OOP	218
Kapitola 17 Definice třídy	225
Kapitola 18 Dědění	239
Kapitola 19 Definice dceřiné třídy	248
Kapitola 20 Zobecňování, protokoly a abstraktní třídy	258
Kapitola 21 Prohlubujeme znalosti OOP	269

Část D	Vývoj aplikace	283
Kapitola 22	Zásady objektové architektury.....	284
Kapitola 23	Návrh základní architektury	296
Kapitola 24	Testovací scénář	308
Kapitola 25	Definice testu a jeho spuštění	320
Kapitola 26	Spuštění hry a její svět	329
Kapitola 27	Definice akcí	342
Kapitola 28	Děláme aplikaci robustní	353
Kapitola 29	Dokončení aplikace	365
Kapitola 30	Volitelné uživatelské rozhraní	372
Kapitola 31	Další možnosti	383
Literatura.....		392
Rejstřík		394
Část E	Přílohy	399
Příloha A	Instalace <i>Pythonu</i> pod Windows.....	400
Příloha B	Konfigurace ve Windows	404
Příloha C	Použité funkce ze standardní knihovny.....	407
Příloha D	Konvence pro psaní programů v <i>Pythonu</i>	413
Část F	Seznamy	417
Seznam výpisů programů.....		418
Seznam obrázků		423
Seznam tabulek		425
Seznam odboček – podšeděných bloků		426

Podrobný obsah

O autorovi	21
Poděkování	22
Úvod	23
Pro koho kniha není	23
Komu kniha určena je	23
Požadované znalosti	24
Jak číst tuto knihu	25
Uspořádání výkladu	25
První část: A Superzáklady	26
Druhá část: B Začínáme programovat	26
Třetí část: C Základy OOP	26
Nezávislost kapitol v předchozích částech	26
Čtvrtá část: D Vývoj aplikace	27
Pátá část: E Přílohy	27
Šestá část: F Seznamy	27
Potřebné vybavení	27
Doprovodné programy	28
Použité typografické konvence	28
Odbočka – podšeděný blok	30
Zpětná vazba	30
Část A Superzáklady	31
Kapitola 1 Předehra	32
1.1 Hardware a software	32
1.1.1 První počítače	32
1.1.2 Co je to program	33
1.1.3 Syntaxe – sémantika – paradigma	33
1.1.4 Změny přístupu k tvorbě programů	34
Vyšší programovací jazyky	34
Modulární programování	35
Objektově orientované programování	35
Funkcionální paradigma	35
Objektově funkcionální paradigma	35
1.1.5 Důležitost čitelnosti programu	36
1.2 Překladače, interprety, platformy	37
1.2.1 Operační systém	37
1.2.2 Platforma	37
1.2.3 Programovací jazyky	38
Překládaný program	38
Interpretovaný program	38
Porovnání	39
Hybridně zpracovávaný program	39

	Jazyk versus způsob zpracování	39
1.3	Platforma Python	40
1.3.1	Součásti standardní instalace	40
1.3.2	Instalace Pythonu	40
1.3.3	Spuštění	41
1.3.4	Skripty	41
1.3.5	Dokumentace	42
1.4	Řádkový interpret	43
1.4.1	Odsazování	44
1.5	Shrnutí a soubory pro opakování	45
Kapitola 2	Vývojové prostředí	46
2.1	Vývojové prostředí	46
2.2	Prostředí IDLE	47
2.2.1	Spuštění prostředí IDLE	47
2.2.2	Základní popis	48
2.2.3	Příkazové okno	49
2.2.4	Opětne zadání dříve zadaných příkazů	50
2.2.5	Restart interaktivního systému	50
2.2.6	Uložení záznamu seance	50
2.2.7	Editační okno	51
2.2.8	Umístění editovaných souborů	51
2.2.9	Pokročilá nastavení	52
	Použité písmo	52
	Barevné zvýraznění textu	52
	Další nastavení	53
2.3	IDLE versus řádkový interpret	53
2.3.1	Zobrazování výpisů programů	53
2.4	Shrnutí a soubory pro opakování	54
Kapitola 3	Zadávání hodnot a proměnné	55
3.1	Počáteční mezery	55
3.1.1	Komentáře	55
3.2	Celá čísla	56
3.3	Reálná čísla	57
3.4	Další možné zápisy čísel	58
3.5	Textové řetězce – stringy	58
3.5.1	Znak # ve stringu	59
3.5.2	Víceřádkové stringy	59
3.5.3	Escape sekvence	60
3.5.4	Bílé znaky	61
3.6	Proměnné a přiřazovací příkaz	61
3.6.1	Identifikátor	61
3.6.2	Konvence pro podobu identifikátorů	62
	Jazyk identifikátorů	63
3.6.3	Definice a použití proměnné, přiřazovací příkaz	63
3.6.4	Zadání skupiny hodnot	64
3.6.5	N-tice hodnot	65
3.7	Hodnota versus odkaz na hodnotu	65
3.7.1	Halda a správa paměti	66
	Odkaz je jako telefonní číslo	66
3.7.2	Terminologie	66
3.7.3	Nebezpečné změny hodnot	67
3.8	Literály	68
3.9	DRY – bez kopií	68
3.10	Shrnutí a soubory pro opakování	69
Kapitola 4	Výrazy, příkazy a používání funkcí	70

4.1	Více příkazů na řádku	70
4.2	Volání funkcí	71
4.2.1	Návratová hodnota funkce	71
4.2.2	Příklady funkcí	71
4.2.3	Parametr versus argument	72
4.3	Hodnota None	72
4.3.1	Podrobnosti o volání funkcí	73
4.4	Objekt vypustka –	73
4.5	Základní aritmetické operace	74
4.6	Formátovací stringy – f-stringy	74
	Závěrečné rovnítko	75
4.7	Výrazy, příkazy, výrazové příkazy	75
4.7.1	Proměnná	75
4.7.2	Přiřazovací výraz	76
4.7.3	Více příkazů na řádku versus více výrazů na řádku	76
4.8	Složený přiřazovací příkaz	77
4.9	Zadání údajů z klávesnice	78
4.10	Shrnutí a soubory pro opakování	79
Kapitola 5	Práce s objekty a nápověda	80
5.1	Vše je objekt	80
5.2	Třída – instance – typ	81
5.3	Atributy objektů a jejich kvalifikace	81
5.3.1	Vytváření objektů	82
5.3.2	Konstrukce = alokace + inicializace	83
	Alokátor	83
	Initor	83
5.3.3	Tovární funkce	83
5.4	Získání nápovědy – dokumentace	84
5.4.1	Argument zadán	84
	Řádkový interpret	84
	IDLE	85
	Nápověda k některým operátorům a konstrukcím jazyka	85
5.4.2	Bez argumentu	85
5.5	Shrnutí a soubory pro opakování	86
Kapitola 6	Moduly	88
6.1	Moduly – základní informace	88
6.1.1	Vše je součástí nějakého modulu	88
6.1.2	Dva názvy objektů	89
6.1.3	Zdrojový soubor	89
6.1.4	Přeložený soubor	90
6.2	Příkaz import	90
6.2.1	Import je jen jiný druh přiřazení	90
6.2.2	Čistý import jiného modulu	91
6.3	Import modulu pod jiným názvem	92
6.3.1	Opakovaný import nic nenačítá	93
6.4	Přímý import	94
6.4.1	Import vyjmenovaných atributů	94
6.4.2	Hromadný import všech atributů	95
6.5	Vytvoření vlastního modulu	95
6.5.1	Název modulu	96
6.5.2	Kódová stránka	96
6.5.3	Dokumentační komentář	97
6.5.4	Import ladícího modulu a kontrolní tisky	97
6.5.5	Zadané příkazy	97
6.6	Práce s vytvořeným modulem	98

6.6.1	Proměnná s odkazem na objekt modulu	98
6.7	Oprava načteného modulu	98
6.8	Opětovné načtení opraveného modulu	99
6.8.1	Co opětovné načtení nezmění	100
6.9	Shrnutí a soubory pro opakování	100
Kapitola 7	Balíčky	101
7.1	Balíčky	101
7.1.1	Trocha terminologie	101
7.1.2	Název modulu – balíčku	102
7.2	Initor balíčku	102
7.2.1	Neinicializované balíčky	102
7.3	Relativní import.....	102
7.4	Umístění modulů s doprovodnými programy	103
7.5	Demonstrační hierarchie balíčků	104
7.6	Co dělat, když interpret na modul nevidí	107
7.6.1	Detekce pracovní složky	107
7.7	Spouštění modulu v balíčku	108
7.8	Shrnutí a soubory pro opakování	109

Část B Začínáme programovat **111**

Kapitola 8	Knihovny, robot Karel a příkaz with	112
8.1	Knihovny	112
8.1.1	Knihovna <code>KarelCz73</code>	113
8.1.2	Rozbalení knihovny do zdrojového kódu	113
	Historie robota Karla	114
8.1.3	Použití nerozbalené knihovny	115
8.1.4	Začlenění knihovny do systému	116
8.1.5	Shrnutí	116
8.2	Signatura funkcí	116
8.3	Robot Karel a jeho svět	116
8.3.1	Vytvoření světa	117
8.3.2	Vytvoření robota	119
	Pozice	119
	Směr natočení	119
	Barva	119
8.3.3	Výchozí sada akcí	119
8.3.4	Testy	121
8.3.5	Zrychlování	122
8.4	Jednoduché a složené příkazy	123
8.4.1	Fyzické a logické řádky	124
8.4.2	Složené příkazy a odsazování	124
8.4.3	Rozdílný způsob práce v interaktivním režimu	125
	Řádkový interpret	125
	IDLE	125
8.5	Příkaz <code>with</code>	125
8.6	Ukončení práce s daným světem robotů	127
8.7	Shrnutí a soubory pro opakování	127
Kapitola 9	Definice funkcí	128
9.1	Terminologie	128
9.1.1	Volatelné objekty	129
9.1.2	Definice funkce	129
9.1.3	Definice prázdné funkce	129
9.1.4	Dokumentační komentář	130

9.1.5	Funkce je objekt, na nějž odkazuje proměnná	132
9.2	Lokální proměnné	133
9.2.1	Funkci můžeme definovat její vlastní atributy	134
9.3	Funkce s návratovou hodnotou	135
9.3.1	Současné vrácení více hodnot	136
9.4	Funkce s parametry	136
9.4.1	Zadávání argumentů	137
9.4.2	Inicializované parametry a implicitní hodnoty argumentů	137
9.4.3	Povinně poziční a povinně pojmenované argumenty	138
9.5	Funkce <code>print()</code> a její parametry	138
9.6	Datový typ	139
	Statické typování	139
	Dynamické typování	140
9.7	Anotace	140
9.7.1	Anotace jsou nepovinné, ale doporučované	141
9.8	Demonstrační příklady	141
9.9	Lambda-výrazy	142
9.9.1	Příklad	143
9.10	Shrnutí a soubory pro opakování	143
9.10.1	Příklady k procvičení	144
	<code>def turn_about(k: Karel) -> Karel:</code>	144
	<code>def step_back(k:Karel) -> Karel:</code>	144
	<code>def step_left(k:Karel) -> Karel: def step_right(k:Karel) -> Karel:</code>	144
	<code>def jump_left(k:Karel) -> Karel: def jump_right(k:Karel) -> Karel:</code>	144
	<code>lambda c2f c:</code>	144
Kapitola 10	Rozhodování	145
10.1	Logické hodnoty	145
10.2	Terminologie výrazů	146
	Operace	146
	Operátor	146
	Operand	146
	Arita operátorů	146
	Priorita operátorů	147
10.3	Porovnávání hodnot	147
10.3.1	Porovnání reálných čísel	147
10.3.2	Zřetězené porovnávání	148
10.3.3	Porovnávání textů	148
10.3.4	Porovnávání totožnosti objektů	148
10.4	Logické operátory a operace	149
10.5	Podmíněný výraz	151
10.6	Podmíněný příkaz	152
10.6.1	Jednoduchý podmíněný příkaz	152
	Opatrný krok	152
10.6.2	Vnořování složených příkazů	153
	Testování	154
10.6.3	Větev <code>else</code> – úplný podmíněný příkaz	155
10.6.4	Rozhodování s více větvemi: rozšířený podmíněný příkaz	155
10.7	Přepínač – příkaz <code>match ... case</code>	156
10.8	Shrnutí a soubory pro opakování	157
10.8.1	Příklady k procvičení	158
	<code>def is_north(k:Karel) -> bool: def is_west(k:Karel) -> bool: def</code>	158
	<code>is_south(k:Karel) -> bool:</code>	158
	<code>def is_turned_to(k:Karel, dir4:Direction4) -> bool:</code>	158
	<code>def is_wall_left(k:Karel) -> bool: def is_wall_right(k:Karel) -></code>	158
	<code>bool:</code>	158

	<code>def clever_step(k:Karel) -> Karel:</code>	158
	<code>def knp_1() -> None</code>	158
Kapitola 11	Opakování kódu, cykly	159
11.1	Předehra	159
11.2	Příkaz while – cyklus se vstupní podmínkou	159
11.2.1	Zanořování cyklů	161
11.3	Nekonečný cyklus	162
11.4	Příkaz break – cyklus s podmínkou uprostřed	163
11.5	Cyklus s koncovou podmínkou	164
11.6	Zdroje hodnot	164
11.6.1	Rozbalovací hvězdička	165
11.6.2	Stringy	165
11.6.3	Objekt typu range	165
11.6.4	Objekt typu enumerate	166
11.7	Příkaz for – cyklus s parametrem	166
11.7.1	Proměnná odkazující na funkci	167
11.7.2	Definice testu	168
11.8	Příkaz continue	169
11.9	Rekurze	170
11.10	Shrnutí a soubory pro opakování	172
11.10.1	Příklady k procvičení	172
	<code>def count_markers(k:Karel) -> int:</code>	172
	<code>def turn_to(k:Karel, dir4:Direction4) -> Karel:</code>	172
	<code>def position(k:Karel) -> tuple[int, int]:</code>	173
	<code>def world_size(k:Karel) -> tuple[int, int]:</code>	173
	<code>def knp2() -> None:</code>	173
Kapitola 12	Kontejnery	174
	Zvláštnosti programových kontejnerů	174
12.1	Kontejnery	174
12.2	Proměnné, neměnné a hešovatelné objekty	175
12.2.1	Hešovatelné objekty	175
12.3	Druhy kontejnerů	176
12.4	Vytváření kontejnerů	177
12.4.1	Vytváření prostřednictvím literálů	177
12.4.2	Vytváření prostřednictvím konstruktorů	178
12.4.3	Vytváření prostřednictvím generátorové notace	181
12.4.4	Generátory lze použít jen jednou	182
12.5	Shrnutí a soubory pro opakování	183
12.5.1	Příklady k procvičení	183
	<code>lambda mocniny :</code>	183
	<code>def multi_table() -> tuple[tuple[int, ...]]:</code>	183
	<code>def hex2dec() -> tuple[tuple[int, ...]]:</code>	183
	<code>def multi_triangle() -> tuple[tuple[int, ...]]:</code>	183
Kapitola 13	Práce s kontejnery	184
13.1	Funkce versus metoda	184
13.2	Vytvoření výchozí sady pomocných kontejnerů	184
13.3	Získání prvku z posloupnosti a slovníků	185
13.4	Procházení kontejnerů – cyklus for	186
13.4.1	Procházení posloupnostmi	187
	Cyklus s více parametry	187
13.4.2	Procházení slovníků – pohledy	188
	<code>keys()</code>	188
	<code>items()</code>	188

	values()	188
	13.4.3 Konvence pro názvy slovníků	189
13.5	Vykrajování (slicing) posloupností	189
13.6	Další možnosti práce s jednotlivými prvky	190
	13.6.1 Test přítomnosti prvku	190
	13.6.2 Přidání a odebrání prvku	191
	Množina	191
	Seznam	192
	Slovník	192
13.7	Funkce s proměnným počtem argumentů	192
	13.7.1 Hvězdičkový parametr	192
	13.7.2 Hvězdičkový argument	194
	13.7.3 Dvuhvězdičkový parametr	195
	13.7.4 Dvuhvězdičkový argument	196
13.8	Jmenné prostory	197
13.9	Shrnutí a soubory pro opakování	197
	13.9.1 Příklady k procvičení	197
	def copy_2_top(k:Karel) -> None:	198
	def histogram(values:list[int]) -> None:	198
	def num_histogram(number:int) -> tuple[int]:	198
Kapitola 14	Ošetřování chyb	199
14.1	Tři druhy chyb	199
	14.1.1 Syntaktické chyby	199
	14.1.2 Běhové chyby	200
	14.1.3 Logické chyby	200
14.2	Reakce na vznik běhových chyb	200
14.3	Zachycení a ošetření výjimky	201
	14.3.1 Průchod programu bloky try ... except ... finally	201
14.4	Demonstrační příklad	202
14.5	Chování programu za běhu	203
14.6	Analýza chybové zprávy	205
14.7	Ladění a kontrolní tisky – modul dbg	205
	14.7.1 Služby modulu dbg	206
	DBG	206
	start_pkg(level: int, name: str, doc: str = '', new_line=True, end_char='␣') -> None	206
	start_mod(level: int, name: str, txt:str='') -> None	206
	prSEd(level:int=1, print_args=False, print_res=False, msg:str= '', wait=False)	207
	prIN(level:int, msg:str='') -> None	207
	prDict(d=None, dict=False, syst=False, msg='', mod=False)	207
	prSeq(seq, prn=True) -> None str	208
	14.7.2 Příklad	208
14.8	Shrnutí a soubory pro opakování	209
	14.8.1 Příklady k procvičení	209
Kapitola 15	Vytváříme aplikace	210
15.1	Spuštění versus import zadaného skriptu	210
	15.1.1 Rozpoznání režimu, v němž byl modul zaveden	210
	15.1.2 Ukázka	211
	Spuštění v interaktivním režimu	211
	Spuštění v příkazovém panelu <i>Windows</i>	212
15.2	Argumenty příkazového řádku	212
15.3	Rozsáhlejší aplikace	213
15.4	Vytvoření spustitelné aplikace	214

15.4.1	Soubor typu <code>pyz</code>	215
15.5	Shrnutí a soubory pro opakování	216
15.5.1	Příklady k procvičení	216

Část C Základy OOP 217

Kapitola 16	Úvod do OOP	218
16.1	Proč se učit objektové paradigma	218
16.1.1	Kdy se OOP začíná vyplácet	219
16.2	Základní princip OOP	220
16.2.1	Objekty a jejich atributy	220
16.2.2	Specifika atributů Pythonu	221
16.2.3	Práce s objekty – kvalifikace	222
16.2.4	Zprávy × metody	222
16.2.5	Metody versus funkce	223
16.3	Třídy a jejich instance	223
16.3.1	Třída	223
16.3.2	Instance	224
16.3.3	Vytváření instancí – konstruktor, alokátor, initor	224
16.4	Shrnutí a soubory pro opakování	224
Kapitola 17	Definice třídy	225
17.1	Definice třídy a jejich atributů	225
17.1.1	Definice prázdné třídy	225
17.1.2	Demonstrační definice standardní třídy	226
17.1.3	Dokumentační komentář	226
17.1.4	Výkonné a výrazové příkazy	227
17.1.5	Datové atributy	228
17.1.6	Instanční metody	228
17.1.7	Statické metody	229
17.1.8	Dekorátory	229
17.1.9	Initor a instanční datové atributy	230
17.1.10	Metody <code>__repr__()</code> a <code>__str__()</code>	230
17.1.11	Speciální identifikátory – dunder	231
17.1.12	Definice třídy je obyčejný příkaz	231
17.1.13	Práce s vytvořenou třídou a jejími instancemi	232
17.2	Definice třídy <code>Fract</code>	233
17.3	Práce s existující třídou a jejími atributy	235
17.3.1	Ještě jednou funkce versus metody	235
	Účel návratové hodnoty s odkazem na instanci – zřetězení volání	236
17.3.2	Děláme Karla chytřejšího	236
17.4	Shrnutí a soubory pro opakování	237
17.4.1	Příklady k procvičení	238
	<code>Turtle2</code>	238
	<code>MultiTurtle</code>	238
Kapitola 18	Dědění	239
18.1	Rozhraní versus implementace	239
18.1.1	Signatura versus kontrakt	240
18.1.2	API	240
18.1.3	PINI	240
18.2	Základní terminologie dědění	241
18.2.1	Dědění versus dědičnost	241
18.3	Tři druhy dědění	241
18.3.1	Přirozené (nativní) dědění	241
18.3.2	Dědění rozhraní	242
18.3.3	Dědění implementace	242

18.3.4	LSP – substituční princip Liskovové.....	243
18.4	Dva způsoby dědění	244
18.4.1	Statické a dynamické typování.....	244
18.4.2	Jmenovité dědění (nominal subtyping)	244
	Hierarchie jmenovitého dědění.....	244
18.4.3	Strukturální dědění a kachní typování	245
18.5	Polymorfismus.....	245
18.6	Virtuální metody a jejich přebíjení.....	246
18.7	Rodičovský podobjekt.....	246
18.8	Initory v procesu dědění.....	247
18.9	Shrnutí a soubory pro opakování.....	247
Kapitola 19	Definice dceřiné třídy	248
19.1	Definice rodičovské a dceřiné třídy	248
19.1.1	Použití vytvořených tříd.....	250
19.2	Karel, který zná svoji polohu	251
19.2.1	Použití neprobraných informací.....	252
19.3	Násobné dědění a diamantový problém	253
19.3.1	Návrh třídy s více bezprostředními rodiči.....	253
19.3.2	Pořadí prohledávání – MRO.....	254
19.3.3	Použití definovaných tříd.....	255
19.3.4	Důležité upozornění pro násobné dědění.....	256
19.3.5	Příklad.....	256
19.4	Shrnutí a soubory pro opakování.....	257
19.4.1	Příklady k procvičení.....	257
	Karel přesouvající se dvojkroky.....	257
Kapitola 20	Zobecňování, protokoly a abstraktní třídy	258
20.1	Zobecňování	258
20.2	Standardní třídy	258
20.3	Abstraktní třídy	259
20.3.1	Terminologie.....	259
20.3.2	Koncepce Pythonu.....	259
20.3.3	Shrnutí.....	260
	Návrhový vzor Šablonová metoda.....	262
20.4	Protokoly	262
20.4.1	Třída Protocol	263
20.4.2	Příklad	263
20.4.3	Návrhový vzor Dekorátor.....	264
	KarelC – robot, který umí změnit svoji barvu.....	265
	Test vytvořeného robota.....	266
20.5	Shrnutí a soubory pro opakování.....	267
20.5.1	Příklady k procvičení.....	267
	Třída Karel2J	267
	Třída Otik	267
Kapitola 21	Prohlubujeme znalosti OOP	269
21.1	Použití nelokálních proměnných	269
21.1.1	Příkaz <code>global</code>	270
21.1.2	Příkaz <code>nonlocal</code>	273
21.2	Neveřejné atributy	273
21.2.1	Atribut <code>__all__</code> modulů.....	274
21.2.2	Nápověda versus hvězdičkový import.....	274
21.3	Atributy × vlastnosti	276
21.3.1	Zadávání a používání vlastností.....	276
	Zadání.....	277
	Použití.....	278

	Pokus o změnu hodnoty vlastnosti nedefinující nastavovací metodu	278
	21.3.2 Vlastnost je atributem třídy	278
21.4	Pojmenované n-tice	279
21.5	Výčtové typy	280
21.6	Shrnutí a soubory pro opakování	282
	21.6.1 Příklady k procvičení	282

Část D Vývoj aplikace 283

Kapitola 22	Zásady objektové architektury	284
22.1	Předmluva	284
22.2	Architektura	285
22.3	Hlavní zásady návrhu	285
	22.3.1 Přípravenost na změny	286
	22.3.2 CRIDP – maximální přehlednost	286
	22.3.3 KISS – maximální jednoduchost	286
	22.3.4 YAGNI – žádné zbytečnosti	286
	22.3.5 SoC – jediný zodpovědný	287
	22.3.6 SRP – jediná zodpovědnost	287
22.4	Návrhové vzory	287
22.5	Antivzory	289
22.6	Návrh programu	289
	1. Účastníci	290
	2. Schopnosti	290
	3. Vlastnosti	290
	4. Kódování	290
22.7	Druhy vytvářených sólo-objektů	290
22.8	Dva způsoby návrhu	291
	22.8.1 Návrh shora dolů	291
	22.8.2 Návrh zdola nahoru	292
	22.8.3 Porovnání	292
22.9	UML – diagram tříd	293
22.10	Důležitost čitelnosti programu	295
Kapitola 23	Návrh základní architektury	296
23.1	Proč právě textová konverzační hra	296
23.2	Záznamy průběhu vývoje v této učebnici	297
23.3.	Koncepce vyvíjené aplikace	297
	Co to je h-objekt	298
23.4	Zadání	299
23.5	Účastníci – objekty vystupující ve hře	301
	Aplikace, Hra – game	301
	Svět – world	301
	Prostor – place	301
	Název – name	301
	Příkaz – Akce – action	302
	Přechod	302
	H-objekt – item	302
	Hráč	303
	Batoň – bag	303
	Úkol, cíl	303
	Množství, kapacita – capacity	304
	Spuštění, ukončení – start, end	304
	Nápověda, přehled – help	304
23.6	Správci skupin objektů	304

23.6.1	Správci v naší aplikaci	304
23.7	Vytvoření zárodku budoucí aplikace	305
23.8	Shrnutí a soubory pro opakování a UML diagram.....	307
Kapitola 24	Testovací scénář	308
24.1	Jak testovat	308
24.1.1	Programování řízené testy	308
24.1.2	Jednotkové, integrační a regresní testy	309
24.1.3	Možnosti testování naší hry	310
24.2	Scénáře	310
24.2.1	Modul <code>tests</code>	311
24.3	Kroky definující stav hry.....	311
24.4	Definice třídy <code>ScenarioStep</code>	312
24.4.1	Systemový a uživatelský podpis	314
24.5	Definice šťastného scénáře	316
24.6	Simulace běhu hry	317
24.6.1	Jednoduchá simulace.....	317
24.6.2	Podrobnější simulace	318
24.7	Shrnutí a soubory pro opakování	319
Kapitola 25	Definice testu a jeho spuštění	320
25.1	Jak budeme testovat	320
25.1.1	Zadání příkazu hře	320
25.1.2	Odpověď a pozice	321
25.1.3	Jak testovat stav hry.....	321
	Odpověď hry.....	321
	Prostor	322
	Sousedé aktuálního prostoru.....	322
	H-objekty v prostoru	322
	Nová definice initoru prostoru.....	323
	H-objekty v batohu.....	323
25.2	Vlastní test hry	323
25.2.1	Kam s ním?	323
25.2.2	Průběh testu	325
	Zpracování příkazu.....	325
25.2.3	Společné zásady a pomocné funkce testů stavu	325
	Převod porovnávaných textů na malá písmena	325
	Společná chybová funkce <code>ERROR()</code>	325
	Funkce <code>compare_sources()</code>	326
25.2.4	Test stavu.....	326
25.3	Spouštíme test.....	327
25.4	Další postup	328
25.5	Shrnutí a soubory pro opakování	328
Kapitola 26	Spuštění hry a její svět	329
26.1	Tři druhy objektů	329
26.2	Delegování zodpovědnosti	330
26.3	Informace, zda hra běží	330
26.4	Funkce <code>execute_command()</code> v modulu <code>actions</code>	331
26.4.1	Definice má být krátká	331
26.4.2	Funkce <code>execute_empty_command()</code>	331
26.4.3	Funkce <code>_execute_standard_command()</code>	332
26.5	Spuštění testu	332
	Načítání modulů	333
	Spuštění testu.....	334
26.6	Pojmenované objekty.....	334
26.6.1	Úvahy o abstraktních třídách v Pythonu	334

26.6.2	Definice třídy ANamed	335
26.6.3	Úvahy nad vlastnostmi	336
26.7	Vytváříme prostory	337
26.7.1	Slovník prostorů	337
26.7.2	Inicializace prostoru	338
26.7.3	Kdy inicializovat – hierarchie inicializací	339
26.7.4	Aktualizace batohu	340
26.8	Spuštění testu	340
26.9	Shrnutí a soubory pro opakování	341
Kapitola 27	Definice akcí	342
27.1	Funkce <code>_execute_standard_command()</code>	342
27.2	Třída <code>Action</code>	343
27.3	Definice slovníku <code>NAME_2_ACTION</code> a test	344
27.4	Manipulace s <code>h</code> -objekty	346
27.4.1	Definice třídy <code>ItemContainer</code>	346
27.4.2	Ošetření násobného dědění	346
	Úprava initoru třídy <code>ANamed</code>	347
	Úprava třídy <code>Place</code>	348
27.4.3	Modifikace třídy <code>Bag</code>	348
27.4.4	Redefinice funkce <code>_take()</code>	349
27.4.5	Kontrolní test	349
27.5	Přesun mezi prostory	349
27.6	Pokládání předmětu	351
27.7	Shrnutí a soubory pro opakování	352
Kapitola 28	Děláme aplikaci robustní	353
28.1	Nesplněné body zadání	353
28.1.1	Potřeba nového scénáře	354
28.1.2	Společný startovní krok	354
28.2	Chybový scénář	354
28.2.1	Co vše se má zkontrolovat	354
28.2.2	Reakce na pokus o nekorektní spuštění	355
28.2.3	Chybový scénář <code>MISTAKE_SCENARIO</code>	356
28.3	Nový spouštěč testů	356
28.4	Oprava nekorektního spuštění	357
	Hra zůstala aktivní	357
	Vyhodnocuje se stav neaktivní hry	357
	Špatná reakce na prázdný příkaz uprostřed hry	358
28.5	Problémy s argumenty standardních akcí	359
28.6	Přesun <code>h</code>-objektů z prostoru do batohu	359
28.6.1	Nová definice třídy <code>Item</code>	360
28.6.2	Předpona může mít širší význam	360
28.6.3	Oprava definice prostorů	361
28.6.4	Úprava funkce <code>_take()</code>	361
28.7	Nápověda	363
28.7.1	Úprava testovací funkce	363
28.8	Shrnutí a soubory pro opakování	364
Kapitola 29	Dokončení aplikace	365
29.1	Jednoduché textové uživatelské rozhraní	365
29.2	Odstranění kontrolních tisků	366
29.3	Možnost opakovaného spuštění	367
29.4	Rozšíření výstupu	368
29.5	Argumenty příkazového řádku a nápověda	369
29.6	Vytvoření spustitelné aplikace	370

29.7	Shrnutí a soubory pro opakování	370
Kapitola 30	Volitelné uživatelské rozhraní	372
30.1	Předehra	372
30.1.1	Změna architektury	372
30.1.2	Protokol <code>IUI</code>	373
30.1.3	Definice a umístění komunikačních objektů – třída <code>Console</code>	374
30.2	Vytvoření primitivního GUI	375
30.2.1	Modalita dialogových oken	375
30.2.2	Knihovna <code>tkinter</code>	375
	Návrhový vzor Fasáda	376
30.2.3	Modul <code>tkinter.messagebox</code>	377
	Parametr <code>**options</code>	377
30.2.4	Modul <code>tkinter.simpledialog</code>	377
30.2.5	Rodičovské okno	378
30.2.6	Schování okna	379
30.2.7	Třída <code>PrimitiveGUI</code>	379
30.2.8	Doplnění metody <code>main()</code> v modulu <code>__init__</code>	380
30.2.9	Spuštění aplikace v interaktivním režimu	380
30.3	Shrnutí a soubory pro opakování	381
Kapitola 31	Další možnosti	383
31.1	Další vylepšování	383
31.2	Převod literálů na konstanty	383
31.2.1	Magické hodnoty	384
31.2.2	Způsoby definice textových konstant	384
31.2.3	Modifikovaný šťastný scénář	385
31.3	Několik dalších námětů	386
	Převod pod kvalitní grafické uživatelské rozhraní	386
	Zdokonalení <code>h</code> -objektů	387
	<code>H</code> -objekty – prostory	387
	Rozšiřování sady příkazů	387
	Rozhovor	387
31.4	Tipy pro učitele – hromadně zadávaná úloha	388
31.4.1	Zadání	388
31.4.2	Návrh architektury a představení společného API	389
31.4.3	Vývoj společného základu	389
31.4.4	Doplnění rozšiřujících akcí	389
31.4.5	Obhajoba závěrečné práce	390
	Požadavky na modifikaci	390
31.5	Shrnutí a soubory pro opakování	391
Literatura		392
Rejstřík		394
Část E	Přílohy	399
Část F	Seznamy	417

O autorovi

Ing. Rudolf Pecinovský, CSc. je absolventem *Fakulty Elektrotechnické ČVUT* z roku 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 pro naprosté začátečníky. V současné době učí na *Fakultě jaderné a fyzikálně inženýrské ČVUT* a na *Fakultě informatiky a statistiky Vysoké školy ekonomické v Praze*. Vedle toho vyučuje ještě v řadě kurzů pro začátečníky i profesionální programátory. Doposud mu vyšlo přes 60 knih, které byly přeloženy do pěti jazyků. Většina jeho knih je zaměřena na výuku moderního programování a na umění návrhu objektově orientované architektury.



Poděkování

Jeden by řekl, že příprava nového vydání začátečnické učebnice bude vyžadovat jenom pár změn a bude to vcelku „brnkačka“. Bohužel, opak je pravdou. Příprava knihy v současném tempu aktualizací byla spojena s tolika oběťmi řady lidí z mého blízkého i vzdálenějšího okolí, že bych měl velkou újmu na duši, kdybych jim zde nepoděkoval.

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. Původně jsem se domníval, že s třetím vydáním nebude moc práce. Šeredně jsem se zmylil, protože veškeré úpravy, které jsem se rozhodl do knihy zanést (a že jich bylo požehnaně), musely zapadnout do předchozího textu.

Na vylepšování textu nového vydání se ale podílela řada dalších lidí. Mezi nimi musím poděkovat především těm, kteří si dali tu práci a při objevení chyby v minulém vydání mi o ní napsali. Především pak děkuji Ludkovi Šťastnému, který po celou dobu rukopis pročítal a odhaloval v něm pasáže, jež by si zasloužily vylepšit.

Velkou zásluhu na současné podobě má i Jirka Kofránek, který mne upozornil na některé problémy s výukou podle minulého vydání a průběžně pak se mnou konzultoval nejasnosti, na něž narazili jeho studenti.

Zvláštní poděkování patří Janu Lampovi, který upravil mou původní knihovnu robota Karla tak, aby ji bylo možné používat bez nutnosti jejího rozbalení a začlenění do vyvíjeného programu. Kromě toho v rámci své bakalářské práce převedl z *Javy* do *Pythonu* i moji další knihovnu, kterou však použiji až v některé z dalších učebnic.

Velký dík patří i redaktoru Petrovi Somogyimu, který musel opakovaně procházet některé již zredigované pasáže, protože jsem je znovu a znovu upravoval. A nemalý dík patří i šéfredaktoru 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 odevzdání rukopisu.

Úvod

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

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 (nebo pomáhající řešit) jejich problém. Na druhou stranu ale sílí i jeho využití profesionálními vývojáři pro rozsáhlé podnikové a webové aplikace.

Pro koho kniha není

Kniha není určena pro ty, kteří se chtějí co nejrychleji naučit jen naprosté základy jazyka, aby mohli co nejdříve začít vytvářet své dvaceti- až padesátirádkové skripty. Pro ty je určena učebnice [Python snadno a rychle \[13\]](#).

Není určena ani pro zkušené programátory, kteří rozšiřují své portfolio jazyků o jazyk *Python*. Pro ty jsou určeny příručky [Python 3.13 – Kompletní příručka základů jazyka \[14\]](#) a [Python 3.13 – Pokročilé vlastnosti jazyka \[15\]](#).

Komu kniha určena je

Tato kniha je určena především těm, kteří se chtějí naučit dobře programovat v jazyku *Python*. Nezáleží na tom, jestli ještě nikdy neprogramovali, anebo se je to sice někdo snažil naučit, ale oni už většinu látky zapoměli. Kniha nepředpokládá žádné předběžné znalosti a dovednosti kromě základů práce s počítačem. Jejím cílem je předat čtenáři základní znalosti a naučit ho dovednosti potřebné k vytváření jednoduchých aplikací.

Knih a kurzů pro začátečníky je celá řada. Naprostá většina z nich však vysvětlí pouze naprosté základy s tím, že vše ostatní se čtenář doučí sám metodou pokus-omyl. Tato kniha je určena především těm, které povrchní výklad těchto kurzů neuspokojuje, protože nechtějí být pouze tupými opisovači někde získaného kódu, ale chtějí se naučit, jak dobře navrhovat kód vlastní.

Zkušenost ukázala, že v takto pojaté učebnici najdou řadu cenných informací i programátoři, kteří již mají jisté zkušenosti, ale kurzy, jimiž doposud prošli, se soustředily především na odpověď na otázku *jak program zapsat*, případně *jak použít tu kterou knihovnu*, ale oni by se nyní rádi dozvěděli také odpověď na otázku *jak složitější program navrhnout*.

Kniha je učebnicí programování. Učí své čtenáře, jak programy navrhovat, ladit a dále je vylepšovat. Není učebnicí jazyka *Python*, a proto se nesnaží podrobně probrat všechny jeho konstrukce (k tomu jsou určeny výše zmíněné příručky [14] a [15]), ale omezuje se při výkladu pouze na ty rysy jazyka, jejichž zvládnutí je pro návrh jednoduché aplikace nezbytné.

Vedle konstrukcí jazyka učí čtenáře také řadu zásad moderního programování, jejichž zvládnutí je nutnou podmínkou pro všechny, kdo nehodlají zůstat u malých žákovských programů, ale chtějí se naučit efektivně vyvíjet robustní středně rozsáhlé aplikace, jejichž údržba nebude vést jejich uživatele k chrlení nepublikovatelných výroků na adresu autora programu.

Osvojené základy jim pak umožní, aby v případě hlubšího zájmu o programování v jazyku *Python* pokračovali některou z učebnic určených pro mírně pokročilé programátory – nejlépe samozřejmě některou z mých dalších učebnic – viz přehled v sekci [Literatura](#) na straně 392.



Dopředu se omlouvám, že se kniha částečně překrývá se zmíněnými příručkami [14] a [15]. Některé věci je prostě třeba vysvětlit jak naprostým začátečníkům v programování, pro něž je určena tato učebnice, tak těm, kteří jsou sice zkušenými programátory, ale potřebují na *Python* přejít z jiného jazyka nebo si prostě prohloubit své znalosti (pro ně jsou primárně určeny příručky [14] a [15]).

Požadované znalosti

Před chvílí jsem řekl, že: „*Kniha nepředpokládá žádné předběžné znalosti a dovednosti kromě základů práce s počítačem.*“ Jak jsem ale z různých reakcí zjistil, představa o tom, co to jsou základy práce s počítačem, se mezi studenty dost různí. Řada z nich se domnívá, že když umějí počítač zapnout a poklepaním spustit aplikaci, tak základy práce umějí. Bohužel, pro studium této učebnice je potřeba trochu více. Budete potřebovat následující dovednosti:

- Vědět, jaký je rozdíl mezi souborem a složkou, umět vytvořit novou složku a vložit či přesunout do ní požadované soubory.
- Stáhnout soubor ze zadané webové stránky a uložit jej do zadané složky.
- Rozbalit zadaný archivní soubor ve formátu ZIP do zadané složky.
- Otevřít okno příkazového řádku (okno konzoly).
- Zadávat v okně příkazového řádku požadované příkazy.

Není to mnoho, ale bez těchto znalostí se při čtení dalšího textu neobejdete. Na druhou stranu nápověda od umělé inteligence, která je již všudypřítomná, vám chybějící znalosti jistě rychle doplní.

Jak číst tuto knihu

Kniha je koncipována tak, aby mohla sloužit jako učebnice v kurzech programování i jako učebnice pro samouky, kteří se vážně zajímají o programování, a povrchní výklad většiny kurzů je neuspokojuje.

Kniha probírá vše potřebné od naprostých základů až po některé rysy, které se v běžných příručkách většinou neprobírají, ale jejichž znalost považuji za velmi užitečnou, protože pomáhá efektivněji navrhnout program, anebo rychleji odhalit příčiny mnohých chyb.

Před výkladem řady konstrukcí proto předchází teoretický úvod, který považuji za nutný k jejich hlubšímu pochopení. Pokud vás teorie nezajímá, klidně tento úvod přeskočte a vraťte se k němu v případě, že vám některé věci budou nejasné.

Současně jsem chtěl, aby mohla sloužit jako stručná referenční příručka. Proto v zájmu toho, aby vzájemně související věci byly pohromadě, ve výkladu občas maličko předběhnu a představím vám cosi (většinou funkce), co podrobně vysvětlím až někdy později.

Platí zde totéž, co pro teoretické pasáže: **neostýchejte se přeskakovat**. Snažil jsem se, abych v případě, kdy při výkladu začnu používat něco, co jste mohli přeskóčit, anebo vám jen z hlavy vypadly podrobnosti, pokaždé doplnil odkaz na místo, kde jsme látku probírali a kde byste si mohli své znalosti osvěžit.

Uspořádání výkladu

Snažil jsem se látku rozvrhnout tak, aby všechny kapitoly byly přibližně stejně dlouhé (přibližně deset stran) a mohli jste látku vstřebávat po rozumných soustech. Kniha je rozdělena do šesti částí.

První část: [A Superzáklady](#)

První část probírá naprosté základy, bez jejichž znalosti nelze vytvořit ani velice jednoduchou aplikaci. Naučíte se v ní pracovat s čísly a texty a dozvíte se, jak používat funkce. Poté stručně probere naprosté základy používání objektů, bez nichž se v *Pythonu* nedá programovat. Na závěr vysvětlí, jak vytvářet moduly, abyste mohli své programy také ukládat a opakovaně používat, a jak u složitějších aplikací tyto moduly uspořádat do balíčků.

Doprovodné programy v první části se nesnaží nic řešit. Jsou to vesměs AHA-příklady, tj. příklady, jejichž jediným cílem je, aby si čtenář řekl: „Aha, takto to funguje.“

Druhá část: [B Začínáme programovat](#)

V druhé části začnete vytvářet jednoduché programy. Naučí vás definovat vlastní funkce a postupně probere základní algoritmické konstrukce a datové struktury, mezi něž patří především různé druhy kontejnerů. Na závěr vysvětlí koncepci ošetřování chyb a předvede používání k tomu určených programových konstrukcí.

Na konci druhé části budete umět samostatně navrhnout a odladit jednoduchý program. Kdo bude chtít, může v tuto chvíli příručku na chvíli odložit a pustit se do vývoje svých programů. Ke studiu dalších částí se vrátí, až získá v programování *Pythonu* jisté zkušenosti a bude umět docenit témata probíraná v těchto částech.

Třetí část: [C Základy OOP](#)

Třetí část vás seznámí se základy objektově orientovaného programování, jejichž znalost je pro vývoj složitějších programů nezbytná. Vysvětlí, kdy začne být výhodný přechod na objektově orientované paradigma, probere základní vlastnosti objektů a tříd a naučí vás pravidla pro jejich vytváření a používání. Zvláštní kapitoly věnuje dědění, jeho různým podobám, pravidlům a především jeho nástrahám. Poté pokračuje výkladem některých specifických konstrukcí a datových typů.

Tato část probírá především teorii demonstrovanou prostřednictvím AHA-příkladů. Na procvičení této teorie dojde v několika doprovodných příkladech, ale především pak při tvorbě netriviální aplikace, které je věnována čtvrtá část.

Na konci třetí části budete nadstandardně obeznámeni s objektově orientovaným paradigmatem a jeho implementací v *Pythonu*. Budete připraveni vytvářet rozsáhlejší aplikace, které by se vám bez schopnosti vytvářet vlastní třídy a objekty navrhovaly mnohem obtížněji.

Nezávislost kapitol v předchozích částech

V prvních třech částech této učebnice se snažím, aby na sobě jednotlivé kapitoly nezávisely, přesněji aby příklady v následující kapitole přímo nenavazovaly na to, co se řešilo v kapitole předchozí. Pokud ano, tak se v daném příkladu vše potřebné vybuduje znovu.

Čtvrtá část: **D. Vývoj aplikace**

Hlavním cílem čtvrté části je procvíčit probranou látku na nějaké netriviální aplikaci a při té příležitosti také naučit čtenáře několik klíčových zásad objektivě orientovaného programování.

Neméně důležitým cílem je převést čtenáře z *hladiny objektového kódování*, kdy programátor používá objektové konstrukce, aniž by jakkoliv měnil styl svého uvažování (tj. používané paradigma), *do hladiny objektového programování*, kde si při návrhu programu opravdu snaží „přemýšlet objektivě“ a řídit se pravidly objektivě orientovaného programování.

Text této části vás nejprve seznámí se základy objektivě orientované architektury a s principem návrhových vzorů. Poté navrhne základní architekturu demonstrační aplikace a začne ji realizovat. Představí vám metodiku programování řízeného testy a ukáže, jak navrhnout vlastní test. Pak už bude třeba jenom opakovaně spouštět tento test, který vždy oznámí, kde je v programu další chyba, a postupným opravováním těchto chyb nakonec vyvinout celou aplikaci.

Na konci této části budete mít rozchozenou jednoduchou aplikaci realizující textovou konverzační hru a budete mít základní představu o tom, jako se objektivě orientované aplikace navrhují.

V posledních dvou kapitolách nejprve představí dva náměty, jak je možné aplikaci zdokonalit, a předvede realizaci těchto změn. Poté naznačuje některé další možnosti vylepšení a ukazuje učitelům, kteří by chtěli knihy používat při výuce, jak modifikovat vyvíjenou aplikaci, aby mohla sloužit jako příklad k závěrečným zkouškám.

Pátá část: **E. Přílohy**

V prvním vydání této učebnice bylo několik příloh, které samy o sobě zabírají asi 10 % objemu knihy. S nakladatelstvím jsme se dohodli, že je v zájmu snížení ceny do papírové verze učebnice nezařadíme a čtenáři si je budou moci stáhnout z webové stránky knihy. Čtenáři elektronických verzí najdou tyto přílohy přibalené na konci dokumentu.

Šestá část: **F. Seznamy**

Jako třešničku na závěr dostáváte další „detašovanou“ přílohu se seznamy výskytů všech tabulek, obrázků, výpisů kódu a podšeděných bloků. Ty pomohou ve chvíli, kdy si potřebujete připomenout nějaký konkrétní objekt, ale nemůžete si vzpomenout, ve které kapitole jste se s ním setkali. Tyto seznamy mohou takováto hledání urychlit.

Potřebné vybavení

Pro úspěšnou práci s touto knihou je vhodné mít instalovanou platformu *Python* alespoň ve verzi 3.13, pro kterou byla kniha psána. Poslední verzi *Pythonu* lze stáhnout na adrese <https://www.python.org/downloads/>. Při instalaci určitě zaškrtněte, že chcete instalovat „*tcl/tk and IDLE*“ a že chcete instalovat dokumentaci, i když se dá používat i on-line dokumentace na stránkách *Pythonu*.

U některých operačních systémů je instalace poněkud složitější, tak jsem se rozhodl, že budu šetřit místem a nejčastější odchylky shrnu v samostatném dokumentu na webových stránkách knihy.

Doprovodné programy

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/73_python-nz3. Na stránce najdete i podrobný výklad obsahu doprovodných programů a časem i pár doplňkových textů.

Mezi jiným zde budou i soubory obsahující všechny výpisy programů v knize i s uvedenými čísly řádků. Tyto soubory vám mají usnadnit sledování rozboru některých programů, abyste si je mohli vytisknout, položit vedle knihy či čtečky a nemuseli při čtení rozboru programů neustále listovat mezi rozbohem a rozebíraným výpisem.

Kromě toho tu najdete i PDF soubory s přílohami, o nichž jsem se zmiňoval před chvílí, a některé další doplňkové soubory.

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, vysazují **tučně**.
- Název** Názvy firem a jejich produktů vysazují *kurzivou*. Kurzivou vysazují také názvy kapitol, podkapitol a oddílů, na něž v textu odkazují.
- Citace** Texty, které si můžete přečíst na displeji, např. názvy polí v dialogových oknech či názvy příkazů v nabídkách, vysazují **tučným bezpatkovým písmem**.
- Odkaz** Celá kniha je prošpikovaná křížovými odkazy na související pasáže. Ne-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. Použitý prohlížeč by je měl na odkazovaný objekt ihned přenést.
- Adresa** Názvy souborů a internetové adresy vysazují obyčejným bezpatkovým písmem.
- Program** Identifikátory a další části programů zmíněné v běžném textu vysazují **neproporcionálním písmem**, které je v elektronických verzích pro zvýraznění tmavě červené.

¹ Číslem 73 v adrese se nevzrušujte, jedná se pouze o moje interní označení pořadí vytvářené knihy, protože bych v nich jinak bloudil.

- Keyword** Klíčová slova jazyka jsou navíc pro zvýraznění vysazena tučně a podtržena.
- 'string'** Textové řetězce, neboli stringy se v programech používají velmi často. Aby se vám v doprovodných programech lépe hledaly, jsou vysazeny malíčko odlišně od okolního kódu.
- >>>** Značná část výpisů zaznamenává konverzaci mezi uživatelem a interpretem jazyka. Výzvy interpretu k zadání příkazu jsou vysazeny s jiným podbarvením a barvou písma, aby je bylo možné snadno odlišit od zadání uživatele a odpovědi na zadaný příkaz.
- Zadání** V záznamech komunikace se systémem budou texty, které zadává uživatel, vysazeny tučně (v elektronických verzích pro zvýraznění tmavě modře).

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.



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



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



Usměváček vás bude upozorňovat na různé tipy, jimiž můžete vylepšit svůj program nebo zefektivnit svoji práci.



Mračoun vás naopak bude upozorňovat na různá úskalí programovacího jazyka nebo programů, s nimiž budeme pracovat, a bude vám radit, jak se těmto nástrahám vyhnout či jak zařídit, aby vám alespoň pokud možno nevadily.



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.



Brýle označují tzv. „poznámky pro šfouraly“, ve kterých se vás snažím seznámit s některými zajímavými vlastnostmi probírané konstrukce nebo upozorňuji na některé souvislosti, avšak které 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 ostatního výkladu. Nadpis podšeděného bloku pak najdete i v podrobném obsahu mezi nečíslovanými nadpisy.

Zpětná vazba

Předem přiznávám, že tato kniha je sice třetím vydáním učebnice programování v jazyku *Python*, ale oproti předchozímu vydání je natolik přepracovaná, že je to vlastně nová kniha. Nelze proto vyloučit, že přestože knihu četlo několik lektorů, mohou se v ní objevit přehlédnutí, která nemá redaktor šanci zachytit a opravit.

Pokud vám proto bude někde připadat text nepřiliš srozumitelný nebo budete mít nějaký dotaz (ať už k vykládané látce či použitému vývojovému prostředí), nebo pokud v knize objevíte nějakou chybu či budete mít návrh na nějaké její vylepšení, neostýchejte se poslat na adresu rudolf@pecinovsky.cz e-mail s předmětem [73_PYTHON_NZ3_DOTAZ](http://knihy.pecinovsky.cz/73_python_nz3_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/73_python_nz3 odpověď i pro ostatní čtenáře, kteří by mohli o danou chybu zapomenout, nebo by je mohl obdobný dotaz napadnout za pár dní, nebo jsou natolik ostýchaví, že si netroufnou se sami zeptat.

Dopředu se ale omlouvám, že na poštu často odpovídám se značným zpožděním, protože jsem se nechal přemluvit k napsání několika dalších knih a rychleji to stihnout neumím.

Část A

Superzáklady

První část knihy vás seznámí se základními konstrukcemi jazyka *Python*, bez jejichž znalosti se při sestavování programu neobejdete. Nejprve poskytnete základní informace o počítačích, programech a programovacích jazycích. Pak představí zadávání čísel a textů, naučí vás volat funkce, vysvětlí, jak pracovat s objekty, a na závěr ukáže, jak ukládat vaše programy do modulů a organizovat je do balíčků.

Kapitola 1

Přehra



Co se v kapitole naučíte

Kapitola poskytuje základní přehled o historii způsobu vývoje počítačových programů a o základních termínech, které budu v textu knihy používat. Seznámí vás se základními programátorskými paradigmaty, probere rozdíl mezi překladači a interprety a objasní termín *platforma*. Na závěr vás seznámí se základními charakteristikami jazyka a platformy *Python* a stručně vysvětlí, jak *Python* instalovat.

1.1 Hardware a software

1.1.1 První počítače

První samočinný počítač řízený programem byl mechanický a navrhl jej v roce 1837 Charles Babbage. Tou dobou byl také publikován první program. Napsala jej Ada Lovelace a vyšel v jejích poznámkách překladatele ke knize o Babbageově stroji.

První fungující elektromechanický samočinný počítač zprovoznil v roce 1938 Konrad Zuse v Německu. V průběhu druhé světové války vznikly další počítače v Anglii a ve Spojených státech.

Po válce vznikl poměrně rychle trh s počítači. Počítače konstruovali převážně muži a programovaly je převážně ženy, protože řada mužů považovala programování za něco přízemního, co není hodno jejich intelektu. Situace se začala měnit až v šedesátých letech, protože se postupně ukazovalo, že podniky utratí mnohem více za programové vybavení než za vlastní počítač.

Navíc většina důvtipných řešení a patentů v oblasti hardwaru rychle zastarala a byla překonána mnohem důvtipnějšími řešeními někoho jiného, kdežto objevy na poli softwaru přežívaly celá desetiletí. V současné době se už proto hlavní vývoj neodhrává v oblasti hardwaru, ale softwaru.

1.1.2 Co je to program

Program bychom mohli charakterizovat jako v nějakém programovacím jazyku zapsaný předpis popisující, jak má procesor, pro nějž je program určen (v našem případě počítač), splnit zadanou úlohu.

Cílovým procesorem nemusí být vždy počítač. Oblíbeným příkladem programů jsou např. kuchařské předpisy. V předpočítačových dobách zaměstnávaly některé instituce (např. armáda) velké skupiny počtářů² řešících na mechanických kalkulačkách výpočty podle algoritmů, které dnes zakódujeme v nějakém programovacím jazyku a předhodíme počítači. Takto se vytvářely například nejrůznější navigační a dělostřelecké tabulky.

Na programu je důležité, že musí být napsán v nějakém programovacím jazyku, kterému rozumí programátor. V počátcích používání počítačů bylo nutné programovat ve *strojovém kódu*. Programátor si musel pamatovat kombinace čísel zadávající jednotlivé instrukce počítači a ve správném pořadí je zadávat do počítače. To samozřejmě vedlo k velkému množství chyb.

Záhy proto vznikly programy, které umožňovaly zadat jednotlivé instrukce pomocí zkratk,³ a takto zadaný program převedly do podoby, které rozuměl počítač. Takový program byl označován jako *překladač* (anglicky *compiler*), protože překládal zadání z tzv. *jazyka symbolických instrukcí* (JSI) do strojového kódu.

Toto zefektivnění práce se programátorům zalíbilo, začali proto vymýšlet, jak ještě více přiblížit způsob zápisu programu způsobu, jímž člověk přemýšlí. Postupně tak vznikaly stále dokonalejší *vyšší programovací jazyky*, které programátorům umožňovaly se maximálně soustředit na řešený problém, aniž by museli nějak výrazně uvažovat nad tím, jak bude počítač jejich program zpracovávat. Tento proces postupného zvyšování abstrakce zadávání programů počítači trvá prakticky dodnes.

Vybrat jazyk pro kuchařský předpis je poměrně jednoduché, vybrat jazyk pro počítač je mnohem složitější. Vlastnosti použitého jazyka totiž naprosto zásadně ovlivňují jak rychlost vývoje programu, tak i rychlost a kvalitu výsledných programů. Proto také prošly programovací jazyky i metodiky jejich používání celou řadou revolučních změn.

1.1.3 Syntaxe – sémantika – paradigma

Při popisu programovacích jazyků se setkáte s řadou termínů. Tři klíčové termíny jsou:

² Příznějme si, že to tehdy byly většinou počtářky, protože muži dělají při práci podobného druhu příliš mnoho chyb. Takovéto „lidské počítače“ pomáhaly např. v Sovětském svazu s vývojem atomové pumy či s prvními lety do vesmíru.

³ Např. instrukce **ADD** přičetla zadané číslo, instrukce **JMP** (zkratka z anglického *jump* – skoč) „skočila“ na zadané místo v programu, odkud se pokračovalo apod.

- **Syntaxe** definuje, jak se zapisují jednotlivé konstrukce jazyka. Je to soubor pravidel definujících, jaké kombinace symbolů vedou ke správně zapsanému programu. Nijak se však netýkají toho, co bude onen správně zapsaný program dělat.
- **Sémantika** popisuje význam syntakticky správných textů v daném programovacím jazyce a způsob jejich interpretace – např.: „*takto zadaný příkaz definuje, podle čeho se má program rozhodnout, jak bude pokračovat*“.
- **Paradigma** označuje celkový přístup programátora k návrhu programu, jeho programovací styl a s tím spojenou sadu preferovaných programových konstrukcí. Různé programovací jazyky podporují různá paradigma.
Používané paradigma není definováno používaným programovacím jazykem, ale způsobem přemýšlení při návrhu programu. Jazyk může pouze nabídnout nástroje, které používání daného paradigmatu usnadní.

1.1.4 Změny přístupu k tvorbě programů

Jak byly počítače postupně zdokonalovány, byly postupně nasazovány v dalších a dalších oblastech a programátoři pro ně vytvářeli stále dokonalejší programy. Programy byly čím dál rafinovanější a složitější, a to začalo vyvolávat velké problémy. Programátoři totiž přestávali být schopni své programy rozchodit, a když je vítězně rozchodili, nedokázali z nich v rozumném čase odstranit chyby, které uživatelé v programu objevili.

Tato krize vedla k zavádění nejrůznějších metodik a *paradigmat*, které měly jediný cíl: pomoci programátorům co nejefektivněji spolehlivé a snadno upravovatelné programy.

Vyšší programovací jazyky

V padesátých letech minulého století se začala prosazovat myšlenka, že by bylo vhodné, aby si vědci a inženýři mohli programy, které by jim pomáhaly řešit jejich problémy, navrhnout sami a nemuseli se dohadovat s programátory, co že to vlastně potřebují. Vznikla idea vyšších programovacích jazyků, které by umožňovaly nesoustředit se při vývoji programu na jazyk počítače, ale vyjadřovat se stylem, který je bližší běžnému vyjadřování.

V polovině padesátých let se narodil jazyk Fortran a za ním poměrně rychle následovaly další. Používání vyšších jazyků zvýšilo přehlednost vytvářených programů a s ní i produktivitu programátorů. Tato změna spolu se zlepšujícím se hardwarem umožnila psát stále rozsáhlejší programy. Avšak tato rozsáhlost byla příčinou řady chyb způsobených především nedostatečnou komunikací programátorů pracujících na různých částech programu.

Modulární programování

V šedesátých letech se proto začalo prosazovat *modulární programování*, které přinášelo nejrůznější pravidla doporučení, jak vytvářet rozsáhlé programy a minimalizovat přitom pravděpodobnost chyb vznikajících právě v důsledku rozsáhlosti těchto programů.

V sedmdesátých letech se začalo prosazovat *strukturované programování*, jehož pravidla bychom mohli stručně shrnout do doporučení *psát program tak, aby byl co nejpřehlednější*. Vychází se přitom z faktu, že programátor stráví několikanásobně více času čtením částí programu, které jsou nově vytvářenými částmi používány nebo na něž tyto nové části navazují, než psaním nového kódu.

Profesionální programátoři dlouho tento styl odmítali (přečtete si např. článek *Opravdoví programátoři nepoužívají Pascal*)⁴ a prohlašovali jej za výmysl univerzitních teoretiků. Praxe však ukázala, že dodržením jeho zásad se výrazně zvýší efektivita vývoje a softwarové firmy jej začaly u svých programátorů vyžadovat.

Objektově orientované programování

V průběhu osmdesátých a zejména pak devadesátých let ovládlo programátorský svět *objektově orientované programování*, které umožnilo vytvářené programy dále zpřehlednit. Opět se opakovala situace, kdy je praktici nejprve odmítali, ale studie na konci osmdesátých let minulého staletí ukázaly, že od určité velikosti programu (hovoří se přibližně o 100 000 příkazů) přestává být v lidských silách udržet trojici (*doba vývoje, spolehlivost, cena*) v rozumné hladině bez aplikace objektově orientovaného paradigmatu.

Funkcionální paradigma

Objektové paradigma se široce používá dodnes, na přelomu století se v některých oblastech začalo prosazovat *funkcionální paradigma*, které doporučuje definovat program jako funkci, při jejíž definici bude dodržována jistá sada pravidel.

Vývoj podle funkcionálního paradigmatu je sice mentálně trochu náročnější, ale na druhou stranu dodržování oněch pravidel výrazně usnadňuje tvorbu programů, které budou rozdělovat svoji práci mezi několik procesorů, abychom měli výsledek k dispozici co nejrychleji.

Objektově funkcionální paradigma

Objektové a funkcionální paradigma se vzájemně doplňují: používání objektového paradigmatu pomáhá zvládat rozsáhlé programy, které potřebují být průběžně upravovány, používání funkcionálního paradigmatu usnadňuje návrh programů, které rozdělují výpočet do několika paralelně běžících vláken.

Obecně bychom tedy mohli říci, že současné době kraluje *objektově-funkcionální paradigma*.

⁴ Viz např. <https://www.logix.cz/michal/humornik/Pojidaci.Kolacu.xp>, heslo *Real Programmers Don't Use Pascal* najdete i na anglické wikipedii.



Teď jsem vás asi tou záplavou termínů vylekal, ale nebojte se. Během dalšího výkladu vás s jednotlivými paradigmaty seznámím podrobněji a ukážu vám, ve kterých situacích je vhodné určité paradigma zvolit. *Python* je totiž multiparadigmatický jazyk, takže vás ve výběru vhodného paradigmatu nijak neomezuje.

V počátcích programování bylo hlavním cílem programátorů, aby jejich programy spotřebovaly co nejméně paměti a byly co nejrychlejší. Tehdejší počítače totiž měly paměti málo, byly z dnešního hlediska velice pomalé a jejich strojový čas byl drahý. Se stoupající složitostí programů však byly takto psané programy stále méně stabilní a stále hůře udržovatelné. Současně s tím, jak klesala cena počítačů, jejich strojového času i paměti, začínal být nejdražším článkem v celém vývoji člověk.

Cena strojového času a dalších prostředků spotřebovaných za dobu života programu začínala být pouze zlomkem ceny, kterou bylo nutné zaplatit za jeho návrh, zakódování, odladění a následnou údržbu. Začal se proto klást stále větší důraz na produktivitu programátorů, a to i za cenu snížení efektivity výsledného programu.

Na druhou stranu je třeba přiznat, že nejrůznější softwarové nástroje se dále zdokonalují a umožňují programovat efektivně, přičemž další nástroje se postarají o to, aby co nejefektivněji běžel i výsledný program.

1.1.5 Důležitost čitelnosti programu

Prakticky každý program zaznamená během svého života řadu změn. Požadavky zákazníka na to, co má program umět, se většinou průběžně mění, a program je proto třeba průběžně upravovat, rozšiřovat a vylepšovat. Celé současné programování je proto vedeno snahou psát programy nejenom tak, aby pracovaly efektivně, tj. rychle a s minimální spotřebou různých zdrojů (operační paměť, prostor na disku, kapacita síť atd.), ale aby je také bylo možné kdykoliv jednoduše upravit a vylepšit. A k tomu je třeba, aby byl program čitelný a srozumitelný, protože programátoři věnují mnohem více času čtení programu než jeho psaní.

Je přitom úplně jedno, jestli autor píše program pro někoho cizího, anebo jej píše pro sebe, protože jej chce použít při řešení nějakého problému. Každý program, který se osvědčí, nás po chvíli používání přivede na myšlenku jej vylepšit.

Problém je, že když se po čtvrt roce jiných aktivit ke svému programu vrátíte, zjistíte, že je to pro vás program úplně cizího člověka. Pokud jste jej nenapsali dostatečně přehledný, začnete pracně přemýšlet nad tím, jak jste to tenkrát mysleli a proč a jak jste realizovali tu kterou fintu.

Předchozí zásady krásně shrnul Martin Fowler ve své knize Refactoring ([\[6\]](#), český překlad [\[7\]](#))

**„Napsat program, kterému porozumí počítač, umí každý trouba.
Dobry programator pise programy, kterym porozumí clovek.“⁵**

Přesně k tomu se vás budu snažit vést v průběhu celého výkladu. Neustále vám proto budu vštěpovat zásadu CRIDP, což je zkratka z anglického *Code Readability Increases Development Productivity* – čitelnost kódu zvyšuje efektivitu vývoje.

1.2 Překladače, interprety, platformy

Než se pustím do dalšího výkladu, měl bych ujasnit pár termínů, které budu v průběhu dalšího textu často používat.

1.2.1 Operační systém

Operační systém je sada programů, jejímž úkolem je zařídit, aby počítač co nejlépe sloužil zadanému účelu. Operační systémy osobních počítačů se snaží poskytnout co největší komfort a funkčnost jak lidským uživatelům, tak programům, které operační systém nebo tito uživatelé spouští. (Teď nehodnotím, jak se jim to daří.)

Operační systém se snaží uživatele (člověka nebo program) odstínit od hardwaru použitého počítače. Uživatel může střídat počítače, avšak dokud bude na všech stejný operační systém, bude si se všemi rozumět.

Při obsluze lidského uživatele to má operační systém jednoduché: člověk komunikuje s počítačem pomocí klávesnice, obrazovky, myši a případně několika dalších zařízení. Ty všechny může operační systém převzít do své správy a zabezpečit, aby se nejrůznější počítače chovaly vůči uživateli stejně.

U programů to má ale složitější. Programy totiž potřebují komunikovat nejenom s operačním systémem (např. když chtějí něco přečíst z disku nebo na něj něco zapsat), ale také přímo s procesorem, kterému potřebují předat své instrukce k vykonání. Problémem ale je, že různé procesory rozumí různým sadám instrukcí.

1.2.2 Platforma

Abychom věděli, že náš program na počítači správně poběží, musíme vědět, že počítač bude rozumět té správné sadě instrukcí a že na něm poběží ten správný operační systém. Kombinaci *použitý hardware + operační systém* budu v dalším textu označovat termínem *platforma HWOS*.

Platforma HWOS je speciálním případem obecnější počítačové platformy, kterou bychom definovali jako *pracovní prostředí umožňující bezproblémovou činnost programů*.

⁵ “Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

Pokud takováto obecnější platforma poběží nad HWOS, tak odstíní program, který na ní běží, od všeho (nebo alespoň téměř od všeho), co se děje „pod ní“, mimo jiné právě od HWOS.

V současné době se dává často přednost oněm obecnějším platformám běžícím nad HWOS. Tyto platformy definují tzv. **virtuální stroj**, což je speciální program provádějící program zapsaný v kódu dané platformy, označovaném často jako **bajtkód** (bytecode).⁶ Vedle něj pak bývá její součástí i poměrně rozsáhlá knihovna často používaných podprogramů.

Delší dobu byla nejrozšířenější takovou platformou *Java*. Běžní uživatelé se asi nejčastěji setkají s platformou *.NET*, nad níž běží většina programů pro *Windows*, a platformou jazyka *JavaScript* – ta ale často běží sama nad platformou definovanou příslušným prohlížečem.

Takovouto obecnější platformou je i *Python*. Jakmile na nějakém počítači instalujete *Python*, můžete na něm používat prakticky všechny programy vytvořené pro tuto platformu. Výjimkou jsou pouze programy, které vedle platformy *Python* používají i přímý přístup k HWOS.

1.2.3 Programovací jazyky

Jak už jsme si řekli, pro zápis programů používáme nejrůznější programovací jazyky. Ty jsou vymyšleny tak, aby v nich mohl člověk co nejlépe popsat svoji představu o tom, jak má počítač splnit požadovanou úlohu.

Program zapsaný v programovacím jazyku pak musíme nějakým způsobem převést do podoby, které porozumí počítač. Podle způsobu, jakým postupujeme, dělíme programy na překládané, interpretované a hybridní.

Překládaný program

U překládaných programů se musí napsaný program nejprve předat překladači (někdo dává přednost termínu kompilátor), který jej přeloží (zkompiluje), tj. převede do podoby, s níž si již daná platforma ví rady. Jinými slovy: musí jej přeložit do kódu příslušného procesoru a používat instrukce, kterým rozumí použitý operační systém. Přeložený program pak můžeme kdykoliv na požádání spustit.

Interpretovaný program

Naproti tomu interpretovaný program předáváme v podobě, v jaké jej programátor vytvořil, programu označovanému jako interpret. Ten obdržený program prochází a ihned jej také provádí – interpretuje jeho příkazy.

⁶ Virtuální stroj se mu říká proto, že se vůči programu v daném bajtkódu chová obdobně, jako se chová procesor vůči programu v čistém strojovém kódu.

Porovnání

Výhodou překládaných programů je, že většinou běží výrazně rychleji, protože u interpretovaných programů musí interpret vždy nejprve přečíst kus programu, zjistit, co má udělat, a teprve pak může tento požadavek vykonat.

Výhodou interpretovaných programů bývá na druhou stranu to, že jim většinou nezáleží na tom, na jaké platformě běží. Stačí, když na daném počítači běží potřebný interpret. Mohli bychom říci, že platformou těchto programů je právě onen interpret.

Vytvoříte-li pro nový počítač interpret, můžete na něj vzápětí přenést i všechny programy schopné běhu nad tímto interpretem. Kdykoliv tyto programy po systému něco chtějí, požádají o to svoji platformu (interpret), a ta jim příslušné služby zprostředkuje. Takovým programům pak může být jedno, na jakém procesoru a pod jakým operačním systémem běží, protože se beztak „baví“ pouze se svou platformou.

Naproti tomu překládané programy se většinou musí pro každou platformu trochu (nebo také hodně) upravit a znovu přeložit. Při implementaci programu pro více platformem bývá někdy pracnost přizpůsobení programu jednotlivým platformám srovnatelná s pracností vývoje jeho první verze.

Hybridně zpracovávaný program

Vedle těchto základních druhů zpracování programů existuje ještě hybridní způsob zpracování, který se snaží sloučit výhody obou skupin. Při něm se program nejprve přeloží do bajtkódu nějaké platformy, který je vymyšlen tak, aby jej bylo možné co nejrychleji interpretovat. Takto přeložený program je potom interpretován virtuálním strojem dané platformy. Ten je zodpovědný za maximálně efektivní interpretaci kódu připraveného překladačem.

Hybridní zpracování spojuje výhody obou kategorií. K tomu, aby v nich napsané programy mohly běžet na různých platformách, stačí pro každou platformu vyvinout potřebný virtuální stroj. Ten pak spolu s knihovnou vytváří vyšší, mnohem univerzálnější platformu. Je-li tento virtuální stroj dostatečně „chytrý“ (a to jsou v současné době prakticky všechny), dokáže odhalit často se opakující části kódu a někde stranou je přeložit, aby je nemusel pořád kolem dokola interpretovat.

Jazyk versus způsob zpracování

Na překládané, interpretované a hybridní bychom měli dělit způsoby zpracování vytvořených programů, avšak často se takto dělí i programovací jazyky. Je sice pravda, že to, zda bude program překládaný, interpretovaný nebo hybridní, není závislé na použitém jazyce, ale je to především záležitostí implementace daného jazyka, nicméně každý z jazyků má svoji typickou implementaci, podle které je pak zařazován.

Všechny jazyky mohou být implementovány všemi třemi způsoby a pro mnohé opravdu existují všechny tři druhy implementace. U většiny však typická implementace natolik výrazně převažuje, že se o těch ostatních prakticky nemluví. Klasický *Basic* je považován za interpretovaný jazyk, *Java* a *Python* za hybridní a jazyky *C* a *Pascal* za překládané.

Hybridní implementace jazyků se v posledních letech výrazně prosadila a hybridně implementované jazyky jsou dnes králi programátorského světa. Vyvíjí v nich převážná většina programátorů a procento implementací v těchto jazycích neustále vzrůstá.

1.3 Platforma Python

Kniha má sice v názvu, že je učebnicí programování v jazyku *Python*, ale samotný jazyk vám není k ničemu do té doby, než získáte nástroje k tomu, abyste jej mohli používat. Pojdme je probrat a následně instalovat.

1.3.1 Součásti standardní instalace

Pro další práci potřebujete mít instalovanou platformu *Python*. Její součástí je:

- **Knihovna**, v níž jsou připravené podprogramy realizující nejčastější operace, abyste je nemuseli všechny vytvářet znovu sami.
- **Překladač**, který váš program přeloží do bajtkódu dané platformy a propojí jej s použitými knihovnami.
- **Virtuální stroj**, jenž přeložený program, tj. bajtkód, příkaz za příkazem interpretuje a tím jej provede.

To vše je součástí minimální instalace, kterou nabízejí některé distribuce *Linuxu*. Součástí standardní instalace však bývá více. Patří do ní:

- Poměrně podrobná **dokumentace** s definicí jazyka, popisem instalovaných knihoven, jednoduchým tutoriálem a řadou dalších užitečných informací.
- Součástí dokumentace je i **tutoriál**, jenž vás seznámí s naprostými základy.
- Knihovna *Tkinter* pro tvorbu grafického uživatelského rozhraní (GUI).
- Nad touto knihovnou vybudované jednoduché vývojové prostředí **IDLE**.
- Program *pip* sloužící k instalaci externích knihoven.

Protože standardní instalace obsahuje prakticky všechny klíčové součásti, které pro vývoj programů v *Pythonu* potřebujete, bývá označována anglickým marketingovým heslem *battery included*.

1.3.2 Instalace Pythonu

Pro instalační soubory si dojděte na stránku <https://www.python.org/downloads/>, kde si zadáte, pro který operační systém chcete *Python* instalovat. Vedle obligátních systémů *Windows* a *macOS* zde najdete informace při instalaci na řadu systémů IBM od AIX po OS/390, pro *Solaris* a *HP-UX* a řadu dalších systémů.

Instalace pro systémy *Linux* jsou nabízeny ve zdrojové formě, aby se při překladu a sestavní přizpůsobily příslušné distribuci. Řada distribucí vám ale nabídne *Python* předinstalovaný, anebo je instalační program součástí dané distribuce.

Zájemci o podrobnější informace i instalaci v nejrozšířenějších prostředích *Windows* a *macOS* najdou popsání postupu v příloze [A Instalace Pythonu](#) pod *Windows* na straně [400](#). Čtenáři papírových knih, které tuto přílohu neobsahují, si mohou texty příloh stáhnout na stránce knihy, odkud si stahovali doprovodné programy.