edice
začínáme s ...

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

soubory ke stažení na www.grada.cz

RUDOLF PECINOVSKÝ

GRADA®

edice
začínáme s ...

Začínáme programovat v jazyku PYTHON

3., rozšířené a aktualizované vydání

RUDOLF PECINOVSKÝ

GRADA Publishing

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.

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

Podrobný obsah

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, jejž 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 zmýlil, 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 vydáni mi o ní napsali. Především pak děkuji Luďkovi 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 řešený 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 (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 padesátiřádkové 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 zapomněli. 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 určené 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 zkušení programátoři, 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 poklepáním 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 přesunotu do ní požadované soubory.
- Stáhnout soubor ze zadané webové stránku 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 přeskočit, 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 procvičit probranou látku na nějaké netriviální aplikaci a při té příležitosti také naučit čtenáře několik klíčových zásad objektově 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 objektově" a řídit se pravidly objektově orientovaného programování.

Text této části vás nejprve seznámí se základy objektově 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 objektově 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 adrese[1] 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 rozborem 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, vysazuji **tučně**.
Název	Názvy firem a jejích produktů vysazuji *kurzivou*. Kurzivou vysazuji také názvy kapitol, podkapitol a oddílů, na něž v textu odkazuji.
Citace	Texty, které si můžete přečíst na displeji, např. názvy polí v dialogových oknech či názvy příkazů v nabídkách, vysazuji **tučným bezpatkovým písmem**.
Odkaz	Celá kniha je prošpikovaná křížovými odkazy na související pasáže. Není-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 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.

[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 progrmech lépe hledaly, jsou vysazeny maličko 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 šťouraly“, 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 nepříliš srozumitelný nebo budete mít nějaký dotaz (ať už k vykládané látce či použitému vývojovému prostředí), anebo 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.

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 zakopnout, nebo by je mohl obdobný dotaz napadnout za pár dní, anebo 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 poskytne 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
Předehra

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 neodehrává 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 počtářů[2] 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 zkratek,[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] Přiznejme 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 programum, 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 paradigmata.

  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 (přečtěte 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 sítě 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.
Dobrý programátor píše programy, kterým porozumí člověk.“**[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 vymýšleny 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 platforem 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. Pojďme 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 knihovou 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 popsaný postup 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.