EDICE
myslíme v ...

knihovna programátora

- Může sloužit jako učebnice i referenční příručka
- Probírá všechny konstrukce jazyka včetně těch běžně přeskakovaných a některých teprve chystaných
- Všechny probírané konstrukce demonstruje na příkladech
- Vysvětluje principy, na nichž je jazyk postaven, a jejichž znalost umožňuje lépe využívat jeho možnosti
- Pro výklad složitějších konstrukcí používá syntaktické diagramy

RUDOLF PECINOVSKÝ

Python

KOMPLETNÍ PŘÍRUČKA JAZYKA PRO VERZI 3.10

programy ke stažení na WWW.GRADA.CZ

GRADA

GRADA

EDICE
myslíme v ...

knihovna programátora

RUDOLF PECINOVSKÝ

Python

Kompletní příručka jazyka pro verzi 3.10

GRADA
Publishing

Rudolf Pecinovský

Python
Kompletní příručka jazyka pro verzi 3.10

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
obchod@grada.cz, www.grada.cz
tel.: +420 234 264 401
jako svou 8162. publikaci

Odpovědný redaktor Petr Somogyi
Fotografie na obálce Depositphotos/mario7
Grafická úprava a sazba Rudolf Pecinovský
Počet stran 592
První vydání, Praha 2021
Vytiskla tiskárna PBtisk a.s., Příbram

ISBN 978–80–271–4424–2 (ePub)
ISBN 978–80–271–4423–5 (pdf)
ISBN 978–80–271–3442–7 (print)

Všem, kteří se chtějí něco naučit

Stručný obsah

Podrobný obsah

Poděkování

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

Chtěl bych především nesmírně poděkovat své ženě Jarušce, která byla po celou dobu mojí největší oporou a jejíž nekonečná trpělivost a vstřícnost mi pomohla dokončit knihu v termínu, který se příliš nelišil od toho, jejž jsme původně s nakladatelem dohodli, a ne až někdy za rok po něm. Původně jsem se domníval, že s dalším vydáním nebude moc práce. Šeredně jsem se zmýlil, protože veškeré úpravy, jež jsem do knihy zanesl, jsou vykoupeny hodinami studia a experimentování, které rodina s neuvěřitelnou trpělivostí snášela.

Obrovský dík patří Jirkovi Kofránkovi, jenž řadu aktivit spojených s přípravou knihy sponzoroval, a dotlačil mne k přípravě kurzu, který na moje učebnice *Pythonu* navazuje.

Na vylepšování textu nového vydání se podílela řada dalších lidí. Mezi nimi musím poděkovat především Luďkovi Šťastnému a Josefu Froňkovi, kteří vznikající rukopis četli a průběžně mne upozorňovali na drobné nesrovnalosti. Vedle nich pak také těm, kteří si dali tu práci a při objevení chyb v minulých vydáních mi o nich napsali. Mezi nimi pak především (podle abecedy) Jiřímu Andršovi, Pavolu Gajdošovi, Janu Jezlovi, Ondřeji Pantíkovi, Peteru Pesselovi, Honzovi Prokešovi, Radku Stasiakovi a Igoru Tomešovi.

V neposlední řadě patří můj dík redakci, především redaktoru Petru Somogyimu, který trpělivě snášel mé neustálé modifikace již zkorigovaného textu, a Radku Matulíkovi, který musel několikrát přepočítávat výrobní parametry průběžně narůstajícího rozsahu knihy.

Úvod

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

Popularita jazyka *Python* nepřetržitě roste. Postupně se stává klíčovým jazykem v řadě oblastí, především v těch, které souvisejí s výukou a výzkumem. Je to nejčastěji vyučovaný první jazyk na univerzitách, je nejpoužívanějším jazykem ve statistice, programování umělé inteligence, skriptování a psaní systémových testů. Má důležitou roli i v oblasti webového programování a různých vědeckých výpočtů. Často se k němu obracejí odborníci, kteří potřebují na počítači vyřešit nějaký problém a jiné jazyky jim připadají buď příliš těžkopádné, anebo pro ně neexistují potřebné knihovny.

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.

Python je vynikajícím nástrojem i pro ty, kteří vyvíjejí rozsáhlejší programy a ocení jeho obrovské možnosti, které běžné jazyky neposkytují. Musí ale současně být dost ukáznění, aby nepotřebovali pracovat v jazyku plném pravidel zabraňujících častým chybám začínajících programátorů a poskytujícím nepřetržitý dozor přísného překladače kontrolujícího jejich dodržování.

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

Komu je kniha určena

Dopředu musím upozornit, že tato kniha není koncipována pro naprosté začátečníky, kteří se s programováním teprve seznamují. Ti potřebují poněkud jiný postup výkladu a jiný výběr příkladů. Pro ně je určena kniha *Začínáme programovat v jazyku Python* ([7]), která vedle používání základních konstrukcí učí své čtenáře také řadu zásad

moderního programování, jejichž zvládnutí je nutnou podmínkou pro všechny, kteří nehodlají zůstat u malých žákovských programů, ale chtějí se naučit efektivně vyvíjet robustní středně rozsáhlé aplikace, jejichž údržba nepovede uživatele k chrlení nepublikovatelných výroků na adresu autora.

Kniha, kterou právě čtete, předpokládá alespoň minimální programátorské znalosti a zkušenosti, což jí umožní soustředit se na výklad konstrukcí a rysů jazyka včetně těch, na které v učebnicích pro začátečníky již nezbývá místo. Je určena pro tři typy uživatelů:

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

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

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

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

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

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

K dokonalému využití platformy *Python* je však potřeba i znalost základních knihoven. V této knize vám představím pouze ty nejzákladnější funkce a třídy. Těm, kteří touží po podrobnějším seznámení s knihovnou a jejími možnostmi, poslouží publikace [10].

Dopředu se omlouvám, že se tato kniha částečně překrývá se začátečnickou učebnicí [7], ale cítil jsem potřebu některé věci vysvětlit jak začátečníkům, pro něž je určena kniha [7], tak pokročilejším čtenářům, pro něž jsem psal tuto publikaci). Nepočítal jsem to, ale odhaduji, že obě knihy mají asi 30 stránek společných.

Struktura příručky

S nakladatelem jsme se dohodli, že není rozumné následovat hlavní proud a vydat jednu velkou učebnici pokoušející se probrat všechny vlastnosti jazyka včetně jeho knihoven, z nichž mnohé řada čtenářů nevyužije. Rozhodli jsme se, že bude výhodnější vydat paralelně příručku snažící se probrat co nejlépe jazyk *Python* a jeho konstrukce, čímž pokryje základy, které využijí prakticky všichni. Na ni by navázaly další, které by se věnovaly nejpoužívanějším knihovnám, přičemž spektrum těchto navazujících učebnic by mohli ovlivňovat sami čtenáři.

Tato učebnice se proto soustřeďuje na výklad jazyka. Doprovodné knihovny až na pár výjimek neprobírá. Umožňuje to probrat jazyk do větší hloubky, takže se dostane i na oblasti, které běžné učebnice opomíjejí a očekávají, že si některá témata čtenáři osvojí metodou pokus-omyl.

Kniha je rozdělená do pěti částí doprovázených samostatně distribuovanými přílohami:

- První část seznamuje s platformou *Python* a s vývojovým prostředím, které budu používat pro definici ukázkových příkladů a demonstraci jejich funkce. Poté vám ukáže, jak zadávat hodnoty jednoduchých datových typů, seznámí vás s jejich použitím ve výrazech a s jednoduchými příkazy, které s takovýmito daty pracují. Její zvládnutí je podmínkou pro studium dalších částí.
- Druhá část probírá složené příkazy. Postupně probereme definice modulů a řady složených příkazů. Po jejím zvládnutí budete schopni vytvářet jednoduché programky, i když se bude jednat spíše o programky na hraní.
- Třetí část rozšiřuje množinu používaných dat o kontejnery, což jsou objekty určené k uchování jiných objektů. Snažil jsem se v ní uvést i pár příkladů, které by již nebyly pouze demonstrační a měly by i praktické využití. Po jejím zvládnutí byste měli být schopni vytvářet v *Pythonu* užitečné programy, které budou moci používat jiní.
- Čtvrtá část se zaměřuje na výklad konstrukcí umožňujících a podporujících objektově orientované programování. Po jejím zvládnutí budete schopni vytvářet v *Pythonu* středně složité programy.
- Pátá část probírá pokročilejší objektové konstrukce, jejichž znalost umožňuje navrhovat efektivnější architekturu vytvářených programů.
- A jak už bylo řečeno, samostatnou část knihy tvoří přílohy. Ty ale nejsou pro ušetření prostoru její součástí, ale najdete je na webových stránkách příručky.

Koncepce výkladu

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

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

  V opačném případě čtenáři vnímají danou konstrukci jako obrázek, který okopírují a používají, aniž by znali význam jednotlivých jeho součástí. (To je ostatně nectnost řady učebnic začínajících oblíbeným příkladem *„Hello World“*.) Při takovémto přístupu totiž čtenáři občas nechápou důsledky některých obratů a akcí a příčiny ohlášených chyb, jejichž odstranění je pak stojí nemalé úsilí.
- Naproti tomu referenční příručky by měly umožňovat co nejsnazší a nejrychlejší dohledání potřebných informací. Mohou být proto koncipovány tak, že každé téma je v nich cele probráno na jednom místě se všemi detaily, a to i za cenu dopředných odkazů, protože při výkladu často potřebují použít konstrukci, která teprve bude vysvětlena. Mohou si to dovolit, protože jsou určeny především zkušenějším čtenářům, u nichž se předpokládají základní znalosti a danou referenční příručku používají většinou k tomu, aby si ozřejmili některé detaily.

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

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

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

Jazyk identifikátorů

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

Potřebné vybavení

Pro úspěšné studium této knihy je vhodné mít instalovanou platformu *Python*. Tu lze stáhnout na adrese https://www.python.org/downloads/.

Kniha je psána pro verzi 3.10, ale drtivá většina příkladů poběží i na verzi 3.9 a vynecháte-li anotace kontejnerů, tak i na verzi 3.8.

Pokud jste si knihu pořídili před 4. říjnem 2021, na kdy je naplánováno vydání ostré verze, tak navštivte https://www.python.org/ftp/python/3.10.0/, kde najdete poslední betu či kandidáta vydání (release candidate). Na její rodičovské stránce najdete odkazy na stránky pro všechny doposud vydané verze od 2.0 i rozpracované verze, které se teprve připravují.

Protože ale vím, že řada čtenářů pracuje se staršími verzemi (např. proto, že to jejich zaměstnavatel požaduje z důvodu kompatibility), budu se snažit upozorňovat na všechna vylepšení, která se objevila v novějších verzích počínaje verzí 3.5, abyste byli včas upozorněni na to, co by vám nemuselo na počítači chodit bez vašeho zavinění.

Operační systém

Při práci v *Pythonu* vám může být většinou zcela jedno, nad jakým operačním systémem je instalovaný. V některých výjimečných situacích na tom ale záleží. Protože 80 % mých studentů i účastníků mých kurů používá operační systém *Windows*, jsou mé knihy primárně zaměřeny na ně. Pokusím se ale nezapomínat ani na ten zbytek.

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/68_python310.

Názvy zdrojových souborů s doprovodnými programy (v *Pythonu* jsou označovány jako moduly) začínají vždy písmenem `m` následovaným dvojmístným číslem kapitoly. Následují-li dvě podtržítka a text (např. `m01__prolog`), jedná se o soubor příkazů zadaných v průběhu kapitoly. Je-li za číslem malé písmeno následované jedním podtržítkem (např. `m01a_script`), jedná se o samostatný modul.

Soubory s příkazy zadávanými v průběhu kapitoly se vyskytují ve třech podobách se stejným názvem, ale odlišnou příponou. Soubory s příponou `py` jsou zdrojové soubory *Pythonu*, soubory s příponou `pyrec` obsahují záznam konverzace se systémem a soubory s příponou `pydoc` obsahují výpisy programů v knize i s uvedenými čísly řádků. Tyto soubory vám mají usnadnit sledování rozboru některých programů, abyste při něm nemuseli neustále listovat mezi rozborem a rozebíraným výpisem.

Vy si je umístěte, kam uznáte za vhodné. Jak napovídají adresy souborů v jejich úvodních komentářích, já je mám rozbalené do složek nazvaných `68_PYT`, `68_REC` a `68_WRD`, které jsou na substituovaném disku `P:`.

[1] Číslem 68 na počátku poslední složky se nevzrušujte, to je pouze moje interní označení pořadí vytvářené knihy, protože bych v nich jinak bloudil.

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 které v textu odkazuji.

Citace Texty, které si můžete přečíst na displeji, například 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, a použitý čtecí program by je měl na odkazovaný objekt ihned přenést.

Adresa Internetové adresy vysazuji obyčejným bezpatkovým písmem.

`Program` Identifikátory a další části programů zmíněné v běžném textu vysazuji `neproporcionálním písmem`, které je v elektronických verzích pro zvýraznění tmavě červené.

`Zadání` Ve výpisech konverzace s počítačem budou zadání uživatele vysazena **`tučně`** (v elektronických verzích pro zvýraznění modře).

`Keyword` Klíčová slova jazyka (anglicky keywords) budou vysezena také tučně i v běžných výpisech programu (v elektronických verzích pro zvýraznění tmavě červeně).

`Chyba` Obdobně budou zvýrazněna chybová hlášení. Nebudou podbarvena a v elektronických verzích budou vysazena červeně.

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

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

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

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

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

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

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

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

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

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

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

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

Odbočka – podšeděný blok

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

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

Zpětná vazba

Předem přiznávám, že tato kniha je sice mou šedesátou publikovanou učebnicí, ale současně je to moje první učebnice *Pythonu*. 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 nebo budete mít návrh na nějaké její vylepšení, neostýchejte se poslat svůj dotaz či připomínku na adresu rudolf@pecinovsky.cz jako e-mail s předmětem 68_PYTHON_310_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/68_python310 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 se netroufnou sami zeptat.

Jenom se musím dopředu omluvit, že na dotazy neodpovídám okamžitě, protože při svém pracovním vytížení zapínám poštu jen jednou za čas.

Část A
Superzáklady

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

Kapitola 1
Startujeme

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

1.1 Hlavní součást instalace

V dalším textu budu předpokládat, že máte staženou a instalovanou platformu *Python* podle návodu v pasáži *Potřebné vybavení* na straně 29. Projděme si nyní, co jste instalovali, a ukažme si, co z toho budeme využívat.

Platforma

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

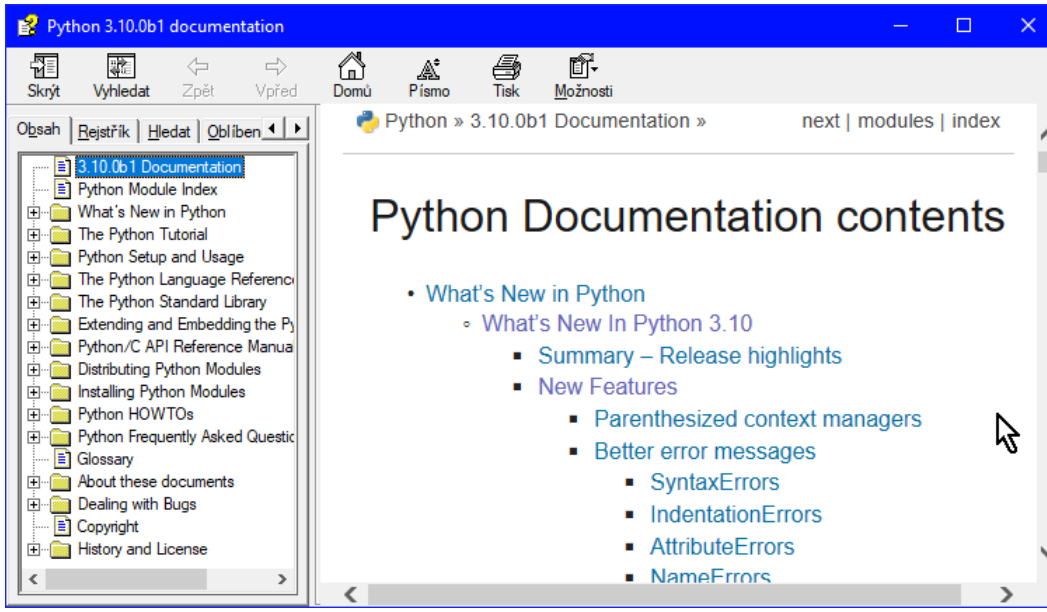
Především potřebujete překladač nebo interpret, abyste svůj program převedli do podoby, s níž je počítač schopen pracovat. Poté potřebujete nějaké vývojové prostředí, ve kterém budete své programy s rozumným úsilím navrhovat. V neposlední řadě pak potřebujete knihovny, v nichž jsou připravené podprogramy realizující nejčastější operace, abyste je nemuseli všechny vytvářet sami. Sadu programů potřebných k tomu, aby se váš program plnohodnotně rozběhl, označujeme termínem *platforma*.

Výchozí standardní implementace, kterou si můžete jako platformu stáhnout na stránkách www.python.org, bývá označována jako *CPython*, protože je implementována v jazyku *C*. Implementace jazyka *Python* vznikla i na jiných platformách. *Jython* přináší implementaci nad platformou *Java*, *IronPython* přináší implementaci nad platformou .NET. Na obou zmíněných platformách však v době psaní této učebnice běžela verze 2, která od počátku roku 2020 již není podporována.

Jednou z poměrně rozšířených platforem je v současné době interaktivní platforma *Jupyter Notebook*,[2] na níž běží odnož jazyka označovaná *IPython*, která byla v době psaní knihy kompatibilní s verzí 376. Tato platforma je používaná zejména při analýze dat a v nejrůznějších vědeckých projektech.

V této učebnici se soustředím na nativní platformu jazyka *Python*, tedy na platformu, kterou jste podle výše uvedeného návodu instalovali, a to konkrétně na **verzi 3.10**, jež vyšla v říjnu roku 2021 (dobrá, příručka vychází o trochu dříve, takže je vše zkoušeno na beta verzi, ale to by na výsledek nemělo mít vliv).

Python se od ostatních platforem liší mimo jiné tím, že kromě základního interpretu, překladače, knihoven a frameworků obsahuje i jednoduché, ale pro začátečníka vcelku použitelné vývojové prostředí, v němž můžete zadávat své příkazy a vyvíjet programy, a mimo jiné také relativně rozsáhlou dokumentaci.



Obrázek 1.1:
Okno s dokumentací platformy

Dokumentace

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

[2] Název je zkratkou z názvů tří primárně zabudovaných jazyků: **Ju**lia, **Py**thon a **R**.

Soubor s doprovodnou dokumentací najdete v podsložce `Doc` složky, do níž jste instalovali *Python*. Můžete ji otevřít v samostatném okně, jehož podobu pod operačním systémem *Windows* si můžete prohlédnout na obrázku 1.1.

Dokumentace je sice pouze anglicky, ale přiznejme si, že všichni, kteří se v současné době zajímají o informatiku, musejí umět anglicky alespoň pasivně, anebo si musejí hledat jiné zaměstnání.

PEP

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

Pracovní režimy

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

- V interaktivním režimu zadáváte příkazy, na něž systém okamžitě zareaguje, zadaný příkaz provede a na následujících řádcích zobrazí svoji odpověď.
- V editačním režimu vytváříte v nějakém textovém editoru zdrojový kód programu, který následně buď přímo, anebo v interaktivním režimu spustíte.

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

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

Překladač a interpret spolu těsně spolupracují. Kdykoliv budu v dalším textu hovořit o zpracování zadaného textu a jeho převodu do spustitelné podoby, použiju termín **překladač**, a když budu hovořit o komunikaci s uživatelem a o tom, co se kdy zobrazí a jak se to zobrazí, použiju termín **interpret**.

1.2 Vývojová prostředí

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

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

PyCharm a IntelliJ IDEA

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

Toto prostředí je postaveno nad platformou *IntelliJ IDEA*, která je základem stejnojmenného nejrozšířenějšího prostředí pro tvorbu programů v *Javě*, ale hojně se používá pro vývoj programů v řadě dalších programovacích jazyků. Rozšíření o podporu *Pythonu* lze do prostředí *IntelliJ IDEA* nahrát i jako plugin. Proto mu zřejmě dají přednost programátoři, kteří již toto prostředí využívají pro tvorbu programů v jiném programovacím jazyce.

Prostředí *PyCharm* můžete stáhnout na https://www.jetbrains.com/pycharm/download, plugin do *IntelliJ IDEA* na adrese https://www.jetbrains.com/idea/download.

Visual Studio Code

Prostředí *Visual Studio Code* vyvíjí *Microsoft* jako open-source. Toto prostředí zřejmě upřednostní programátoři programující v tomto IDE v některém programovacím jazyku pro platformu .NET. Prostředí má nepřeberné množství pluginů pro nejrůznější jazyky včetně *Pythonu*.

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

Jupyter Notebook a JupyterLab

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

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

Jak už jsem řekl, toto prostředí je oblíbené při analýze dat a práci na různých vědeckých projektech, kdy uživatel ocení jeho interaktivitu a možnost prokládat programy běžným textem.

Základní interpret a IDLE

Při používání nejrůznějších propracovaných vývojových prostředí bychom se však zejména na počátku zbytečně rozptylovali vstřebáváním základních pravidel jejich ovládání. Má proto smysl si mezi nimi začít vybírat až v okamžiku, kdy zvládnete základy jazyka a nejdůležitějších součástí standardní knihovny.

V učebnici se oněmi propracovanými vývojovými prostředími zabývat nebudeme, protože využijeme toho, že dvě základní vývojová prostředí nabízí i standardní instalace:

- Základní interpret můžete spustit buď přímo, anebo z příkazového řádku v konzolovém okně zadáním příkazu `python`. Jako vývojové prostředí je však konzolové okno příliš těžkopádné – je vhodné tak maximálně pro spouštění jednotlivých skriptů.
- Prostředí IDLE je interaktivní vývojové prostředí s jednoduchým editorem, které je celé napsané v *Pythonu* a najdete jej v podsložce `Lib\idlelib\` složky, do které jste *Python* instalovali.

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

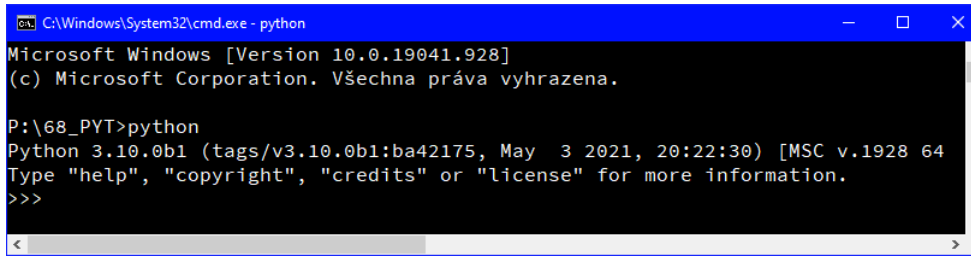
Výběr vývojového prostředí, které budete používat, je však zcela na vás, protože zdrojový kód bude nezávisle na použitém vývojovém prostředí shodný a shodné by měly být i odpovědi v interaktivním režimu. Jak si ukážeme ve výpisech 1.1 na straně 39 a 1.2 na straně 43, bude se lišit nejvýše jejich umístění.

1.3 Komunikace s interpretem

Všechna vývojová prostředí včetně tak jednoduchého, jakým je IDLE, poskytují okna či panely pro komunikaci s interpretem v interaktivním režimu. Představím vám proto základy komunikace s interpretem v okně příkazového řádku, kterou všechna zmíněná okna či panely přebírají.

Interpret spustíte zadáním příkazu `python` (viz obrázek 1.2). Po spuštění *Pythonu* se zobrazí úvodní řádky oznamující spuštění interpretu následované řádkem s výzvou

(anglicky prompt) k zadání příkazů, jejichž prostřednictvím získáte další informace nebo spustíte zadanou akci.



Obrázek 1.2:
Okno příkazového řádku Windows se spuštěným interpretem Pythonu

Výzvu tvoří čtveřice znaků „>>> “ (tři většítka následovaná mezerou). Když za ní něco napíšete a stisknete ENTER, interpret zadaný řádek zpracuje a zdá-li se mu, že jste zadali celý příkaz, vypíše na další řádek výsledek.

Odsazování

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

Příkaz zadávaný za výzvou musí vždy začínat hned za výzvou bez jakýchkoliv úvodních mezer či tabulátorů, jak je předvedeno na řádku 1 ve výpisu 1.1, kde jsem zadal součet dvou čísel. Pokud bychom však před naše zadání vložili byť jedinou mezeru, jak je naznačeno na řádku 3, interpret ohlásí syntaktickou chybu.

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

```
1  >>> 12+34
2  46
3  >>>  0
4    File "<stdin>", line 1
5      0
6  IndentationError: unexpected indent
7  >>> (56
8  ...  + 78
9  ...  ) - 100
10 34
11 >>>
```

Nevejde-li se nám celé zadání na řádek, musíme interpret včas varovat – např. tím, že otevřeme závorku. Po stisku klávesy ENTER pak interpret přečte řádek, zjistí, že ještě není ukončen (otevřená závorka není uzavřena), a připraví další řádek, v němž vypíše pokračovací výzvu tvořenou třemi tečkami a mezerou (viz řádky 8–9). Když při čtení řádku pozná, že zadání skončilo (v našem případě, že před ukončením řádku byla

závorka uzavřena), tak celý blok textu přeloží a zpracuje a na následujícím řádku (u delšího výsledku na následujících řádcích) vypíše případný výsledek.

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

V následující podkapitole vás budu seznamovat s IDLE. Rozhodnete-li se používat okno či panel editoru nabízený vaším vývojovým prostředím, můžete tuto podkapitolu přeskočit, ale určitě si přečtěte navazující podkapitolu *1.5 Používání vývojových prostředí* na straně 45.

1.4 IDLE – seznamte se

Jak už jsem naznačil, prostředí IDLE je pro úvodní kroky asi nejvhodnější, protože je za prvé součástí základní instalace, takže už je nemusíte instalovat, a za druhé je ze všech nejjednodušší. Pojďme si je představit.

Název prostředí je nyní oficiálně zkratkou z anglického *Integrated Development and Learning Environment* (Integrované vývojové a učební prostředí). Původně se sice jmenovalo stejně, ale bylo pojmenováno podle Erika Idlea, jednoho ze zakládajících členů skupiny *Monty Python*, podle níž byl pojmenován celý jazyk.

Spuštění

Spouštěcí skript prostředí IDLE je v souboru `#/Lib/idlelib/IDLE.py`, kde symbol `#` zastupuje složku, kam jste instalovali *Python*.

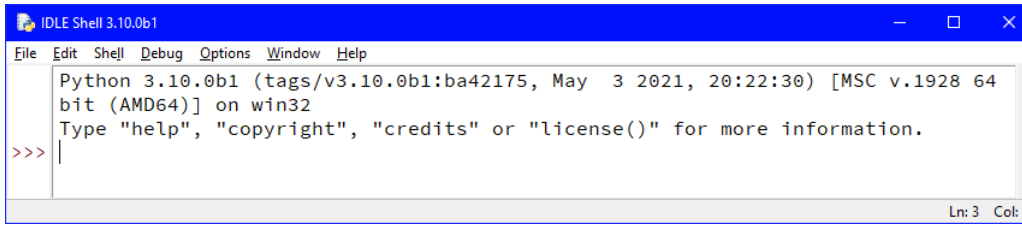
Pracujete-li ve *Windows*, doporučuji spustit skript v souboru s příponou `pyw`, protože tento skript nepotřebuje spouštět okno konzole, jelikož komunikuje s uživatelem prostřednictvím grafického uživatelského rozhraní (GUI) využívajícího (mimo jiné) oken a myši. Standardní systémové vstupy a výstupy budou proto od programu odpojeny a vše, co by na ně program poslal, se ztratí.

Při práci se soubory se otevírací, resp. ukládací okno otevřené z příkazového okna otevře ve složce, z níž byl skript spuštěn, kdežto u příkazů zadávaných z okna otevřeného souboru se primárně otevře ve složce daného souboru. Při práci se soubory umístěnými v podsložkách si vhodnou volbou okna, v němž požádáte o otevření dalšího souboru, můžete ušetřit trochu toho následného klikání.

Důležité je, abyste IDLE otevírali ve složce, v níž jsou umístěné vaše skripty. Optimální je definovat zástupce, kterého umístíte do složky se svými skripty a který IDLE spustí. Používáte-li *Linux*, budete si nejspíš umět nastavit potřebného zástupce otevírajícího IDLE a nastavujícího složku `68_PYT` se zdrojovými soubory jako aktuální. Používáte-li *Windows*, můžete využít mého zástupce v souboru `!IDLE_Python_310.lnk`. Tento zástupce je ale nastaven pro můj počítač, takže si jej budete muset otevřít a upravit podle uspořádání na vašem počítači. Případný návod najdete v podkapitole *A.2 Nastavování zástupce spouštějícího IDLE* na straně 595.

Základní popis

Prostředí IDLE (viz obrázek 1.3) je multiokenní prostředí, jehož okna mohou pracovat v jednom ze dvou režimů:



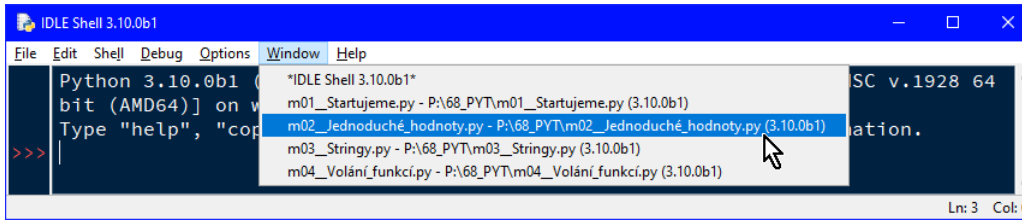
Obrázek 1.3:
Okno vývojového prostředí IDLE

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

Režim aktuálního okna poznáte podle titulkové lišty. Příkazové okno v ní má uvedenu verzi programu následovanou slovem **Shell**. Editační okno v ní uvádí název otevřeného souboru následovaný pomlčkou, úplnou cestou k danému souboru a v kulatých závorkách uvedenou verzi *Pythonu*.

Režim otevřeného okna ovlivňuje i hlavní nabídku (nabídkovou lištu). Příkazové okno má jako třetí položku nabídku **Shell** a jako čtvrtou položku nabídku **Debug** (viz obrázek 1.4), zatímco editační ono má místo nich nabídky **Format** a **Run**.

Mezi okny se přepínáte buď aktivací na příkazovém panelu operačního systému (jednotlivá okna se zde vydávají za samostatné aplikace), anebo výběrem požadovaného cílového okna v nabídce **Window** (viz obrázek 1.4).



Obrázek 1.4:
Výběr požadovaného cílového okna v nabídce Window

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

Na obrázku 1.3 to sice vypadá, že se příkazové okno otevře se čtyřmi řádky textu, ale je to jenom proto, že se první řádek do okna nevešel, a tak jej program zalomil a pokračoval na druhém řádku. Pokud byste uchopili svislý okraj okna a okno roztáhli, přetékající konec prvního řádku by se vrátil na své místo.

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

Protože vím, že většina programátorů dává přednost tmavé barvě pozadí a světlému písmu, protože mnohem méně oslňuje, tak bych zájemce navedl v nastavovacím okně na kartu **Highlights**, kde si v pravé části mohou nastavit tmavé téma (na obrázku 1.4 je nastavené) a v případě potřeby je i upravit na téma vlastní.

Příkazové okno

Nenastavíte-li ve výše zmíněném okně **Settings** něco jiného, tak se po spuštění aplikace otevře příkazové okno (viz obrázek 1.3), v němž se zobrazí úvodní řádek definující verzi *Pythonu* a použitý procesor následovaný řádkem s nápovědou a řádkem s výzvou k zadání příkazů, jejichž prostřednictvím získáte další informace nebo spustíte zadanou akci.

Až do verze 3.9 byla výzva standardní součástí zobrazovaného textu, stejně jako je tomu v příkazovém panelu (viz obrázek 1.2). Na rozdíl od příkazového panelu se v příkazovém okně nevypisovala pokračovací výzva, což občas způsobovalo problémy, o nichž jsem hovořil v předchozích verzích této příručky.

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

Dokumentace přitom slibuje, že takto upravené IDLE bude součástí následující aktualizace předchozí verze, tj. verze 3.9.

Ve výpisu 1.2 si můžete prohlédnout zopakovanou sadu příkazů z výpisu 1.1 a můžete tak porovnat, jak se tyto dva způsoby záznamu komunikace v interaktivním režimu liší.

Výpis 1.2: *Reakce na odsazení příkazu*

```
>>> 0
    0
>>>  0
...
    SyntaxError: unexpected indent
>>> (56
...  + 78
...  ) - 100
    34
>>>
```

Restart interaktivního systému

V interaktivním režimu je občas výhodné smazat všechny výsledky pokusů a začít znovu. K tomu slouží příkaz **Restart Shell** v nabídce **Shell**, anebo stisk klávesové zkratky CTRL+F6.

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

Ukázky ve výpisech jsou většinou koncipovány tak, že žádný restart nevyžadují a mnohé na sebe dokonce navazují. Pokud by někdy byl potřeba restart, vždy na to výslovně upozorním.

Návrat k dříve zadaným příkazům

IDLE si zadané příkazy pamatuje. Naposledy zadaný příkaz zkopírujete za aktuální výzvu zadáním ALT+P (P = Previous). Postupným zadáváním této zkratky místo něj kopírujete dříve zadávané příkazy. Aby to však fungovalo, nesmíte mít za výzvou nic zadáno.

Pokud náhodou přeběhnete, tak zadáním zkratky ALT+N (N = Next) procházíte seznam zadaných příkazů zase vpřed.

Někdy je nejjednodušší se za pomoci posuvníku (nebo kurzorových šipek) prostě na příslušný příkaz přesunout, umístit na něj kurzor a stisknout ENTER. Příkaz se pak zkopíruje za aktuální výzvu a kurzor se umístí na jeho konec.

Použití klávesy ENTER funguje, i když už máte něco zadáno – kopírovaný příkaz se prostě umístí za tento text.

Uložení záznamu seance

IDLE umožňuje uložit záznam aktuální seance do souboru. To se může hodit v případech, kdy se dostanete do potíží a potřebujete se s někým poradit, kde děláte chybu, nebo když naopak chcete někomu ukázat, jak má postupovat: co má zadávat a co může po svých zadáních od systému očekávat.

Záznam seance uložíte příkazem **Save** nebo **Save As...** v nabídce **File**. IDLE otevře dialogové okno a nabídne vám uložit soubor s příponou souboru `py`. To vám vřele nedoporučuji, protože je to přípona vyhrazená pro zdrojové soubory *Pythonu*, a tím záznam seance není, protože vedle vašich příkazů obsahuje i odpovědi systému a jeho následné výzvy (`>>>` – tři většítka následovaná mezerou) k zadání dalšího příkazu. Zvolte proto raději příponu `txt` nebo nějakou jinou vhodnou příponu, která nebude uživatele ani systém mást (já používám příponu `pyrec` jako zkratku z *python record*).

Po uložení svého obsahu se příkazové okno s daným souborem sdruží a název souboru se zobrazí v titulkové liště okna. Při příštím zadání příkazu **Save** se soubor aktualizuje. Příkazem **Save As...** uložíte soubor pod novým jménem a ihned s ním okno sdružíte. Chcete-li průběžně ukládat mezistavy do jiných souborů, aniž byste měnili aktuální sdružení okna se souborem, použijte příkaz **Save Copy As...**.

Editační okno

Editační okno je sice primárně určeno k tvorbě zdrojových souborů modulů (o modulech začneme hovořit v kapitole *9 Moduly* na straně 130), ale můžete v něm zobrazovat a/nebo upravovat libovolný textový soubor.

V nabídce **Options** můžete příkazem **Hide Line Numbers** vypnout číslování řádků. Příkaz se tím automaticky změní na **Show Line Numbers**, jehož zadáním číslování řádků opět zapnete. Číslování se však vztahuje pouze na editační okna. V interaktivním oknu řádky číslovat nelze.

V téže nabídce je také příkaz **Show Code Context**, po jehož zadání se nahoře objeví podšeděný řádek, v němž se budou zobrazovat hlavičky složených příkazů (např. definic funkcí), které mají příkaz svého těla zobrazen v horním řádku editačního okna. Bude-li daný příkaz uvnitř vnořeného složeného příkazu, tak se počet řádků tohoto pole zvětší, aby se v něm mohly zobrazit hlavičky všech otevřených složených příkazů.

S prvním složeným příkazem se seznámíte až v kapitole *11 Definice funkcí* na straně 162. Příkaz **Show Code Context** vám připomenu, až se v kapitole *13 Pokročilé rysy funkcí* na straně 187 začneme bavit o vnitřních funkcích a definice se tak začnou zesložiťovat.

Umístění editovaných souborů

Chcete-li otevřít zdrojový soubor, IDLE jej implicitně hledá v aktuálním adresáři, v němž jste je spustili. Doporučuji vám proto zařídit (například vhodným nastavením zástupce), aby se prostředí spouštělo ve složce, v níž máte umístěné zdrojové soubory.

Já se nakonec rozhodl umístit text knihy i všechny její doprovodné programy na substituovaný disk `P:`.[3] Jak si můžete přečíst v úvodních komentářích stažených souborů, u mě je na něm pro doprovodné programy vyhrazena složka `68_PYT`. V té je umístěn i zástupce spouštějící IDLE, které pak tuto složku nastaví jako svoji pracovní složku.

Barevné zvýraznění textu

Jednou z výhod IDLE oproti používání příkazového řádku je i to, že barevně zvýrazňuje zadávaný i vypisovaný text, aby byl pro uživatele přehlednější a srozumitelnější. Způsob tohoto zvýraznění, tj. přiřazení barev popředí a pozadí jednotlivým druhům textu, si můžete nastavit. O toto zvýraznění vás v textu částečně ochudím – zvýrazněny budou pouze klíčová slova a komentáře. Zadáte-li ale texty doprovodných programů učebnice do okna IDLE, budou barevně zvýrazněny.

1.5 Používání vývojových prostředí

V této podkapitole proberu několik informací, které byste měli znát nezávisle na tom, budete-li při studiu této příručky využívat vývojové prostředí IDLE, nebo dáte přednost nějakému propracovanějšímu prostředí.

Odchylky zobrazení konverzace v IDLE

U minulých knih mi několik čtenářů psalo, že se v mých výpisech programů chová *Python* trochu jinak, než když si dané programy zkoušejí sami. Posléze se ukázalo, že příčinou je nepatrně odlišné chování okna `IDLE`, z nějž záznamy komunikace přebírám, a panelu interpretu, který používají oni.

Hlavní rozdíl spočíval v odsazování počátků řádků. Když se nyní chování IDLE změnilo, tak původní rozdíl sice zmizí, ale objeví se nový.

Hlavní příčinou rozdílu je to, že vývojová prostředí zobrazují ve svých panelech či oknech interpretu celou komunikaci stejně, jak by se zobrazovala při přímém spuštění *Pythonu* z příkazového řádku.

Toto zobrazení se liší především tím, že na řádcích se zadávanými příkazy se před tyto příkazy zapisuje výzva, kdežto odpovědi systému se zapisují přímo od kraje. Jak jsme si ale řekli, IDLE má pro výzvy vyhrazen speciální boční panel, který je však aktivní i u odpovědí, takže při používání IDLE jsou odpovědi zdánlivě posunuté o 4 znaky od levého kraje. Nenechte se tím zmást.

---

[3] Pracujete-li ve *Windows*, můžete popřemýšlet, jestli by se vám také nehodilo využít substituovaný disk. Podrobněji si o této možnosti můžete přečíst v příloze *A Konfigurace ve Windows* na webové stránce knihy.

Tištěná kniha i její PDF verze umějí tento levý panel jasně označit. Nevím, nakolik to dokáží všechny čtečky knih ve formátu EPUB. Mnohé čtečky formátu EPUB totiž nebývají schopné zachytit všechny nuance složitějšího zlomu. Používáte-li knihu ve formátu EPUB, tak se vám za tyto nedokonalosti dopředu omlouvám.

Přepnutí způsobu zobrazení

Z rozhovorů se čtenáři předchozích verzí jsem se dozvěděl, že většina z nich začala v zájmu co nejrychlejšího startu s vývojovým prostředím IDLE, ale po přechodu k výkladu složitějších konstrukcí dala přednost nějakému komfortnějšímu vývojovému prostředí, které více napovídalo a případně i vhodně doplňovalo rozepsaný kód a zefektivňovalo tak práci.

Rozhodl jsem se proto, že v první části budou výpisy programů respektovat způsob zobrazení komunikace používaný vývojovým prostředím IDLE, ale od druhé části dám přednost způsobu používanému při spuštění *Pythonu* v konzolovém okně, který je blíže způsobu používanému v těchto vývojových prostředích.

Použité písmo

Při psaní kódu je nesmírně důležité zapisovat přesně to, co chcete počítači sdělit. Mnohá písma ale neumožňují zkontrolovat, zda jsem napsal přesně to, co jsem chtěl, protože zobrazují některé znaky stejně. Při programování je důležité používat písmo, které jednoznačně odliší nulu a velké O. Stejně tak je důležité odlišit znak velké I od malého L a jedničky.

Velmi často používané písmo *Courier* snadné rozlišení těchto znaků nenabízí – jednoduše se v něm zamění nejenom nula s O, ale také jednička s malým L (`O0 - Il1 # MOJE×M0JE; 321×32l`).

O něco lépe je na tom písmo Consolas definované v *Microsoftu*. To již nulu od O výrazně odlišuje, ale s odlišením jedničky od malého L máme občas u menšího písma stále problém (`O0 - Il1 # MOJE×M0JE; 321×32l`).

Z písem, která jsem měl možnost poznat, dopadlo zatím nejlépe písmo *Source Code Pro* (`O0 - Il1 # MOJE×M0JE; 321×32l`). Je navíc volně k dispozici, takže nemáte-li na svém počítači písmo s vhodnými vlastnostmi, můžete si je stáhnout a instalovat. Toto písmo je použito i v textu této knihy (tedy nepoužíváte-li e-knihu ve formátu EPUB, který s tím má občas problémy).

Doporučuji vám proto, abyste si ve svých editorech, které používáte k psaní kódu, nastavili toto písmo nebo nějaké písmo s podobnými vlastnostmi. V programu IDLE toho dosáhnete zadáním příkazu **Options → Configure IDLE**. V následně otevřeném dialogovém okně pak na kartě **Fonts/Tabs** vyberete v seznamu **Font Face** požadované písmo a stiskem **OK** své zadání potvrdíte.

1.6 Objekty a objektové programování

Python je postaven nad základní myšlenkou objektově orientovaného programování (v dalším textu budu často používat zkratku OOP), že *všechno je objekt*. A když říkám všechno, tak myslím opravdu všechno. Tím se liší od většiny rozšířených jazyků, které o sobě také tvrdí, že jsou objektově orientované, ale onu základní myšlenku akceptují jen částečně.

Pravdou je, že značná část uživatelů *Pythonu* píše jednoduché programy obsahující desítky až stovky řádků kódu. Při psaní takovýchto programů se většinou objektová povaha jazyka a zpracovávaných dat příliš nevyužije. Na druhou stranu bylo ale na přelomu 80. a 90. let ukázáno, že rozrostou-li se programy na desítky až stovky tisíc řádků kódu, přestává být v lidských silách vytvořit rozumně spolehlivý program v rozumném čase a za rozumnou cenu bez použití OOP.

Když učím jazyky používané převážně pro vývoj oněch obludných programů (např. *Javu*), začínám výklad právě vysvětlením základů objektově orientovaného paradigmatu. V této učebnici jsem se ale rozhodl respektovat skutečnost, že řada programátorů v *Pythonu* ve svých programech OOP explicitně nepoužívá, a výklad objektově orientovaného programování a jeho podpory v jazyku *Python* jsem proto přesunul až do čtvrté části, za výklad základních konstrukcí jazyka.

Explicitně

V předchozím odstavci je důležité slovíčko *explicitně*. V *Pythonu*, v němž je vše definováno jako objekt, totiž nelze nepoužívat objekty. Můžete se sice tvářit, že je nepoužíváte, ale to na skutečnosti nic nemění. Proto je mnohem výhodnější objektovou podstatu *Pythonu* akceptovat a naučit se ji využívat. V řadě situací vám pak výrazně stoupne produktivita práce.

Před tím, než spolu dojdeme do části, kde se budeme objektové podstatě jazyka věnovat podrobně, se nebudu moci vyhnout použití některých objektových rysů jazyka. Vždy však před to zařadím výklad následně použitých konstrukcí a základních principů, na nichž jsou postaveny.

A začneme s tím hned teď, kdy vám v kostce představím alespoň základy objektově orientovaného programování.

Objekt, třída, instance, kontejner

Vzhledem k objektové podstatě jazyka *Python* se ale neubráním tomu, abych o objektech hovořil už nyní. Předběhnu proto trochu výklad OOP ve čtvrté části učebnice a seznámím vás s nejdůležitějšími termíny, které budeme v následujícím výkladu používat.

Objekt

OOP staví na základní myšlence, že **všechno je objekt**. Kdykoliv se proto tento termín v textu objeví, dosaďte si za něj „*cokoliv, s čím můžeme v programu pracovat*“, anebo „*cokoliv, co můžeme nazvat podstatným jménem*“.

Třída

Mezi objekty, s nimiž váš program pracuje, se vždy najdou skupiny objektů, které mají nějaké společné vlastnosti a schopnosti. *Python* umožňuje definovat speciální objekty, které označujeme jako třídy. **Třída** (anglicky *class*) sdružuje na jednom místě souhrn informací o oněch společných vlastnostech a schopnostech dané skupiny objektů. Sama pak pracuje jako továrna na objekty s danými charakteristikami.

Instance

Třída je jediný objekt, který je schopen vyrábět jiné objekty. Objekty vytvořené třídou označujeme jako **instance** dané třídy. Třída jim při jejich „porodu“ dá do vínku ony uchovávané vlastnosti a schopnosti objektů dané skupiny.

Každý objekt ví, která třída jej porodila (čí je instancí) a kde má v případě potřeby hledat ony informace společné pro instance dané třídy.

Kontejnery

Speciálním druhem objektů jsou tzv. **kontejnery**, což jsou objekty určené primárně k uchovávání jiných objektů. Protože celé programování je o tom, že si někde musíme něco zapamatovat, abychom to mohli někde jinde použít, jsou kontejnery nejčastějším druhem používaných objektů.

Zvláštní postavení mezi kontejnery mají tzv. **posloupnosti**, což jsou kontejnery, u nichž je definováno pořadí uložených prvků.

Druhou důležitou skupinou kontejnerů jsou tzv. **asociativní paměti** označované často jako **slovníky**, **mapy** nebo **mapovací objekty**. Ty slouží k tomu, aby si program po zadání objektu „vybavil“ jiný, který s tím zadaným definovaným způsobem souvisí.

O kontejnerech se několikrát zmíním již v prvních dvou částech, ale plně se jejich výkladu začnu věnovat až v části *C Kontejnery* na straně 243.

V dalším textu budu často upozorňovat na odchylky *Pythonu* od nejpoužívanějších jazyků, mezi něž řadíme jazyky *Java*, *C/C++*, *C#*, *Visual Basic*, *Delphi*, *Pascal* a jazyky jim podobné. Tyto jazyky budu hromadně označovat termínem *mainstreamové jazyky*.

Mezi mainstreamové jazyky nezařazuji jazyk *JavaScript*, přestože je velmi rozšířený. Jeho koncepce je však zcela osobitá a v řadě ohledů je blíže *Pythonu* než výše zmíněným jazykům označeným jako mainstreamové a jinde zase naopak. Musel bych se proto o něm vyjadřovat samostatně a na to tu není prostor.

1.7 Nejdůležitější zvláštnosti Pythonu

Než se pustíme do výkladu vlastního jazyka, chtěl bych upozornit čtenáře, kteří již pracovali v jiných jazycích, že při práci v *Pythonu* bude možná třeba některé návyky změnit a naučit se přemýšlet v trochu jiném paradigmatu. Pojďme si ve stručnosti projít nejdůležitější odchylky *Pythonu*, abyste se na ně mohli připravit.

Přísné a benevolentní programovací jazyky

Než se rozhovořím o přísnosti jazyků, představím vám základní typy chyb (podrobněji je probereme v podkapitole *16.1 Tři druhy chyb* na straně 227):

- Mezi *syntaktické chyby* řadíme prohřešky proti pravidlům jazyka, na které vás mnohá IDE upozorní hned v okamžiku, kdy daný program píšete, ale nejpozději je označí interpret, kterému je sdělí překladač při prvním pokusu o překlad daného kódu. Programátoři je proto většinou ani nepovažují za chyby. Sem patří například chyba na řádku 13 ve výpisu 2.5 na straně 54.
- *Běhové chyby* jsou chyby, které počítač ohlásí za běhu programu. Program se při nich často zhroutí, ale alespoň už víme, že v něm je chyba, a můžeme ji začít hledat.
- *Logické chyby* jsou takové, že program běží a tváří se, že je vše v pořádku. Když se chyba projeví, bývá už často řada věcí nevratně poškozena.

Podle přístupu k odhalování potenciálních chyb bychom mohli rozdělit programovací jazyky do dvou skupin:

- Jako *přísné jazyky* budu označovat takové, jejichž autoři se snažili navrhnout svůj jazyk tak, aby se při vývoji programu co největší procento případných běhových a logických chyb stalo chybami syntaktickými. Aby toho dosáhli, zavedou do jazyka řadu pravidel, která programátor musí dodržovat a která pak ono zrychlené odhalení chyb umožní. Jedním z nich je i *přísné typování*, při němž je pro každou proměnnou přesně určeno, jaký druh hodnot se do ní smí uložit.
- Jako *benevolentní jazyky* budu označovat takové, jejichž autoři se naopak domnívají, že programátoři se ohlídají sami a ocení spíše to, že jim jazyk poskytne co největší svobodu a nabídne co nejlepší vyjadřovací možnosti, takže jsou s vývojem programu dříve hotovi a program je navíc kratší.

Python patří spíše do skupiny benevolentních jazyků. Nabízí širokou škálu efektivních vyjadřovacích možností (postupně se s nimi seznámíte), i když na druhou stranu nabízí některé konstrukce, které jeho uživatelům umožní používat „přísná“ pravidla, na něž jsou z přísných jazyků zvyklí.

- První změnou, které si všimnete, je to, že *Python* zrušil závorky označující hranice bloků kódu – bloky kódu označuje pomocí odsazení jejich příkazů (viz podkapitolu *8.7 Složené příkazy a odsazování* na straně 126).
- Druhou změnou, která některým programátorům činí potíže, je to, že *Python* důsledně dodržuje zásadu, že vše je objekt, který můžeme uložit do proměnné. V *Pythonu* proto můžete uložit do proměnné nejen číslo nebo text, ale i funkci, třídu, modul, balíček a další objekty, které programátoři v jiných jazycích vnímají spíše jako abstrakce.
- Třetí změnou, na níž si budou muset zvyknout především programátoři pracující v mainstreamových jazycích, je skutečnost, že *Python* patří mezi benevolentní jazyky (viz podšeděný blok *Přísné a benevolentní programovací jazyky*), které předpokládají, že programátora nemusí hlídat překladač, ale že se ohlídá sám. Při psaní je proto potřeba větší sebekázeň a důkladnější testování.

  S tím souvisí i to, že *Python* patří mezi dynamicky typované jazyky, v nichž se datový typ vyhodnocuje až za chodu. Typ zde není vlastnost proměnné, do které se odkaz na objekt ukládá, ale je to pouze vlastnost objektu. K této otázce se ještě vrátíme v kapitole *6 Proměnné* na straně 91.

Kapitola 2
Zadávání jednoduchých hodnot

Co se v kapitole naučíte

Kapitola vám postupně vysvětlí, jak správně zapisovat číselné hodnoty včetně komplexních čísel. V průběhu výkladu ukáže nejčastější místa různých chyb. Poté vám představí některé speciální hodnoty, vysvětlí, co to jsou literály. Vynechá jen zadávání textů, kterým bude věnována celá příští kapitola. Na závěr zdůrazní význam přehlednosti programů.

Než se pustíme do něčeho složitějšího, naučíme se nejprve zadávat jednoduché hodnoty. Pojďme si to zkusit. Otevřete si IDLE (případně vámi preferované vývojové prostředí) nebo spusťte *Python* z příkazového řádku a experimentujte se mnou.

2.1 Zápis celých a desetinných (reálných) čísel

Zadáte-li nějakou hodnotu, na dalším řádku se tato hodnota vypíše. *Python* rozlišuje (jako ostatně většina programovacích jazyků) čísla celá a čísla desetinná, která bývají v učebnicích označována také jako reálná čísla nebo jako čísla s plovoucí desetinnou čárkou (anglicky floating point numbers). Občas budu tyto názvy používat i já.

Celá čísla můžete zadávat libovolně veliká, ale u desetinných čísel musíte mít na paměti, že nepožádáte-li o nestandardní zpracování, bude si je program pamatovat s přesností na přibližně 17 platných cifer nezávisle na tom, bude-li se jejich hodnota pohybovat v miliónech nebo milióntinách. Navíc musíte respektovat, že v programování se nepoužívá desetinná čárka, ale desetinná tečka. Reakci *Pythonu* na zadání velkého celého čísla a na zadání desetinného čísla s mnoha platnými číslicemi ukazuje výpis 2.1.

Výpis 2.1: *Reakce na zadání velkého čísla*

```
1 >>> 123456789012345678901234567890
2     123456789012345678901234567890
3 >>> 0.12345678901234567
4     0.12345678901234568
5 >>>
```

Zpřehlednění dlouhých čísel pomocí znaku podtržení

Jak je z předchozí ukázky patrné, velká čísla jsou nepřehledná a při jejich zápisu se snadno udělá chyba, která se pak špatně hledá. Abychom zápis čísel zpřehlednili, povoluje *Python* vkládat doprostřed čísel znaky podtržení, které ale při výpisu hodnot ignoruje. Nenutí nás přitom k jejich vkládání na speciální pozice (např. mezi trojice číslic), protože v různých zemích platí různé zvyky. Jenom nesmíme vkládat podtržení na první a/nebo poslední pozici.

Zpřehlednění čísel z předchozího výpisu si můžete prohlédnout ve výpisu 2.2. Jak ukazuje řádek 2, u celých čísel se pamatují všechny zadané číslice, kdežto u desetinných čísel (přesněji u čísel typu `float`) se pamatuje jenom 16 až 17 platných cifer.

Výpis 2.2: *Zpřehlednění čísel vložením podtržítek*

```
1 >>> 123_456_789_0_123_456_789_0_123_456_789
2     123456789012345678901234567890
3 >>> 0.123_456_789_012_345_678_9
4     0.12345678901234568
5 >>>
```

Reálné číslo se v programu pozná podle toho, že v jeho zápisu použijeme desetinnou tečku, anebo v něm použijeme písmeno `e` nebo `E`, za něž zapíšeme číslo oznamující, kolikátou mocninou deseti se má vynásobit číslo zadané vlevo od znaku `e`, resp. `E`.[4] O tom, zda při výpisu bude použit tvar s exponentem či bez něj, rozhodne *Python* podle toho, který je podle něj přehlednější.

Například číslo zadané ve výpisu 2.3 na řádku 1 vypsal na řádku 2 v exponentovém tvaru, i když bylo zadané klasicky, ale číslo zadané na řádku 3 vypsal na řádku 4 naopak klasicky, i když bylo zadané v exponentovém tvaru. (O tom, jak lze vypisovanou podobu ovlivnit, si povíme v kapitole *22 Formátování stringů* na straně 312.)

Jakmile je v zápisu čísla použita desetinná tečka nebo exponentový tvar, považuje je *Python* za desetinné, i kdyby se podle jeho hodnoty jednalo o číslo celé, tj. i kdyby mělo prázdnou desetinnou část. *Python* je vypíše s neprázdnou desetinnou částí, jak ve výpisu 2.3 ukazuje zobrazení čísel na řádcích 6 a 8.

[4] Tento zápis bývá v textech označován různě. Nejčastěji se hovoří o zápisu v tzv. exponentovém nebo semilogaritmickém nebo matematickém nebo vědeckém tvaru. Já budu v textu používat termín *exponentový tvar*.

Výpis 2.3: *Zadávání a zobrazování desetinných čísel*

```
>>> 123_456_789_0_123_456_789_0_123_456_789.
1.2345678901234568e+28
>>> 123e-3
0.123
>>> 123.
123.0
>>> 123e3
123000.0
>>>
```

2.2 Komplexní čísla

Python má vestavěnou podporu práce s komplexními čísly. Komplexní čísla se pak zapisují jako součet reálné a imaginární části, přičemž imaginární část se neoznačuje písmenem `i`, jak nás to učili v matematice, ale písmenem `j`, které se používá v elektrotechnice a řídicí technice, protože písmeno `i` je vyhrazené pro označení elektrického proudu. V *Pythonu* je přitom jedno, jestli použijete malé `j` nebo velké `J` – viz výpis 2.4.

Výpis 2.4: *Zadávání a zobrazování komplexních čísel*

```
>>> 3+5j
 (3+5j)
>>> 7j+5
 (5+7j)
>>> 1 + 1J
 (1+1j)
>>> 1+j
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    1+j
NameError: name 'j' is not defined
>>> 2j
2j
>>> 0+3j
3j
>>>
```

Jak ve výpisu 2.4 ukazuje chybová zpráva vyvolaná v reakci na příkaz na řádku 7, na rozdíl od matematiky nemůžeme napsat písmeno `j` samotné, ale vždy u něj musíme uvést číslo. Jak ale naznačuje ukázka na řádku 12, nemá-li dané číslo reálnou část, nemusíme ji uvádět. Na řádcích 13 a 15 se můžete přesvědčit, že při zobrazování čísla s nulovou reálnou částí *Python* nezdůrazňuje jeho „komplexnost“ uzavřením do závorek, jak to dělal u čísel, které měly obě části nenulové.

Chybové zprávy mají přesně definovanou strukturu, kterou se budeme učit analyzovat v podkapitole *16.2 Chybové zprávy* na straně 229.

2.3 Počáteční nula

Vraťme se ale k celým číslům. Pro ně platí jedno důležité upozornění: **v *Pythonu* nesmíte zapsat celé číslo s počáteční nulou**! U desetinných čísel to nevadí, ale u celých čísel je to zakázané a způsobí to syntaktickou chybu.

Důvodem pro tento zákaz je to, že v jazyce *C* a v jazycích přebírajících jeho syntaxi (*Java*, *C++*, *Groovy*, *C#*, …) se počáteční nula používá k označení toho, že číslo je zapsané v osmičkové soustavě. To je ale potenciálním zdrojem chyb, který je o to větší, že jazyk *Python* nebyl primárně vyvíjen pro programátory, ale pro odborníky z jiných profesí, kteří si občas potřebují něco naprogramovat.

Autor jazyka *Python* se proto rozhodl tento způsob označení osmičkových čísel zakázat a zapisovat čísla ve všech alternativních číselných soustavách (za chvíli o nich bude řeč) shodně. Ve výpisu 2.5 je na řádku 1 zadáno číslo začínající nulou a na následujícím řádku je pak reakce systému, který tento zápis označí za syntaktickou chybu.

Výpis 2.5: *Zápis čísel v alternativních číselných soustavách*

```
>>> 0123
  File "<stdin>", line 1
    0123
    ^
SyntaxError: leading zeros in decimal integer literals are not permitted; use an 0o prefix for octal integers
>>> 0b1111_1111
255
>>> 0o377
255
>>> 0xFF
255
>>> 0o0123_4567
342391
>>> 0x0123_4567_89AB_BCDEF
1311768467463720431
>>> -0b0010
-2
>>>
```

2.4 Zadávání čísel v jiných číselných soustavách

V některých programech je výhodné, můžeme-li zadávat čísla i v jiných číselných soustavách, především ve dvojkové, osmičkové a šestnáctkové. Možnost zadávat čísla v těchto soustavách oceníte například tehdy, chcete-li pracovat s jednotlivými bity zadaných hodnot. (Budeme se tomu podrobněji věnovat v podkapitole *7.5 Operace s jednotlivými bity* na straně 115.)

Chcete-li zadat číslo v některé z těchto soustav, napíšete nulu následovanou písmenem označujícím soustavu a číslem zapsaným v zadané soustavě. Pro označení číselné soustavy se používají následující předpony:

`0b` `0B` Dvojková (binární) soustava.

`0o` `0O` Osmičková (oktalová) soustava.

`0x` `0X` Šestnáctková (hexadecimální) soustava.

Na tom, zda k označení číselné soustavy použijete malé či velké písmeno, nezáleží. Obdobně nezáleží na tom, použijete-li jako šestnáctkové číslice malá či velká písmena.

Před chvílí jsem vysvětloval, že v *Pythonu* je zakázáno začínat zápis celých čísel nulou. Při zápisu čísel ve výše uvedených soustavách to již neplatí a úvodní nuly se naopak často používají k zpřehlednění zápisu.

Ve výpisu 2.5 si můžete prohlédnout zápis čísla `255` v jednotlivých soustavách a pak v osmičkové a šestnáctkové soustavě zapsaná čísla, která obsahují všechny číslice své číselné soustavy. Na řádku `13` je pak demonstrována možnost zadání záporného čísla.

2.5 Platí – neplatí

Jednou za čas potřebujeme zadat, zda něco platí, nebo neplatí. *Python* pro označení toho, že něco platí, používá konstantu `True`, pro označení toho, že to neplatí, pak konstantu `False`. Programátory se zkušenostmi z jiných jazyků upozorňuji, že v *Pythonu* se tyto konstanty píší s velkým počátečním písmenem.

2.6 Nic – `None`

Jednou za čas potřebujeme oznámit, že v danou chvíli nemáme žádná data k dispozici. V takovém případě nás nula nebo prázdný string nezachrání, protože jak nula, tak prázdný řetězec jsou platná data. Pro označení toho, že v daném okamžiku nemám k dispozici žádná data, slouží v *Pythonu* konstanta `None`, což je řádný objekt reprezentující ono NIC.

Když v interaktivním režimu zadáte hodnotu `None`, interpret se tváří, jako kdybyste nic nezadali, tj. nic nevytiskne a zobrazí nový řádek s výzvou `>>>` (viz výpis 2.6).

2.7 Výpustka – `Ellipsis`, `...`

Při psaní programů se co chvíli vyskytne situace, kdy víme, že na daném místě musíme něco zadat, ale v danou chvíli ještě nevíme, co tam právě patří. Předpokládáme přitom, že to zanedlouho vědět budeme, ale teď potřebujeme nějakou náhražku, jejímž vložením uspokojíme požadavky syntaxe. V roli této náhražky se občas používá právě objekt

`Ellipsis`, jenž má „přezdívku“ `...` (tři tečky – v běžném textu tento symbol označuje výpustku). Tato přezdívka sice odporuje pravidlům pro tvorbu identifikátorů, ale překladač ji akceptuje – viz výpis 2.6.

Výpis 2.6: *Reakce na použití objektů `None` a `...` (`Ellipsis`)*

```
>>> None
>>> ...
Ellipsis
>>>
```

2.8 Objekt `NotImplemented`

Třetím nestandardním objektem je objekt s názvem `NotImplemented`, který se používá v situacích, kde někdo chce po daném objektu něco, co programátor ještě nedefinoval, anebo definoval, ale nezabezpečil, aby o tom daný objekt věděl. S použitím se setkáte ve výpisu 7.1 na straně 110 nebo ve výpisu 34.1 na straně 497.

2.9 Literály

Přímá zadání hodnoty, která jsme probírali v této kapitole, se označují jako *literály*. Obecně bychom mohli říci, že *literál je konstanta nazvaná svojí hodnotou*. Napíšete-li v programu např. `7` nebo `None`, zapsali jste literál.

Každý jazyk definuje, které druhy dat je možné zapsat prostřednictvím literálů. Všechny jazyky, které znám, umožňují zapsat prostřednictvím literálů celá čísla, reálná čísla (pokud je podporují) a krátké texty, které lze zadat na jeden řádek kódu. Některé jazyky podporují zadávání víceřádkových textů a dalších druhů dat. Z pohledu na to, co vše lze zadat jako literál, patří *Python* mezi ty bohatší.

My jsme zatím probrali zadávání jednoduchých hodnot. V další kapitole vám ukážu, jak se v *Pythonu* zadávají texty a pak si předvedeme, že *Python* umožňuje zadat prostřednictvím literálů i některé kontejnery a časem k nim přidáme i další druhy dat.

2.10 Důležitost přehlednosti

Jednou z nejdůležitějších zásad správného programování je povinnost psát své programy maximálně přehledné. Martin Fowler, známý počítačový guru, napsal ve své knize *Refactoring:*

Napsat program, kterému rozumí počítač, umí každý trouba.
Dobří programátoři píší programy, kterým rozumějí lidé.

Řada programátorů se snaží vytvářet programy, které budou maximálně rychlé a spotřebují minimum paměti. Zkušenost však ukazuje, že ve většině případů dokáže počítač optimalizovat program rychleji a efektivněji. Ve srozumitelně napsaných programech se většinou vyzná lépe i optimalizační program, a proto bývají optimalizovány lépe, než by to dokázal člověk.

Kdykoliv budu v následujícím textu uvádět více možností zápisu programu, tak mějte na paměti důležitou programátorskou zásadu:

Při výběrů z několika možných verzí návrhu programu platí,
že lepší verze je téměř vždy ta,
která je pro vás přehlednější a pochopitelnější,
protože při jejím používán uděláte méně chyb,
a budete proto se svojí prací dříve hotovi a odevzdáte kvalitnější program.

Začínající programátoři se většinou lépe vyznají v podrobněji rozepsaném kódu, ti zkušení již většinou myslí ve zkratkách, a dávají proto přednost hutnějším verzím.

Nesnažte se dokazovat si, že jste zkušení programátoři tím, že budete preferovat hutnější podobu zápisu. Snažte se program psát tak, abyste mu bez problémů porozuměli nejenom zítra, ale po roční odmlce, kdy už si jeho podobu nebudete pamatovat a při pokusech o jeho úpravu či zdokonalení zjistíte, že je pro vás stejně cizí jako program nějakého jiného člověka.

Kapitola 3
Zadávání textů – stringů

Co se v kapitole naučíte
Kapitola vám vysvětlí, jaké možnosti *Python* nabízí pro zadávání textů. Ukáže, jak se v *Pythonu* zapisují komentáře a jak zadávat kód, který se při dodržování konvencí nevejde na jeden řádek. Poté vám předvede, jak vhodnou předponou modifikovat význam zadávaného stringu. Představí vám f-stringy a ukáže, k čemu je můžete využít.

3.1 Zadávání textů

V programu často potřebujeme zadávat nejrůznější texty. Krátké texty, které zadáváme přímo v kódu, se oficiálně nazývají *textové řetězce*, ale programátoři je slangově označují jako *stringy*. Protože tento termín je mezi programátory hluboce zakořeněný, budu jej v dalším textu používat.

Pro text, který můžeme zadat na jednom řádku, poskytuje *Python* dva možné způsoby zápisu: zadávaný text musí být vždy uzavřen mezi apostrofy nebo mezi uvozovky (viz výpis 3.1). Jejich význam se nijak neliší, jenom musíte dbát na to, abyste hraniční znak použitý na počátku použili i na konci.

Volba hraničního znaku může být ovlivněna tím, zda potřebujete některý z nich použít uprostřed textu – viz příkazy na řádcích 5 a 7 ve výpisu 3.1.

Výpis 3.1: *Zápis jednořádkového textu*

```
1 >>> "Text v uvozovkách"
2     'Text v uvozovkách'
3 >>> 'Text v apostrofech'
4     'Text v apostrofech'
5 >>> "Potřebuji 'apostrof'"
6     "Potřebuji 'apostrof'"
7 >>> 'Potřebuji "uvozovky"'
8     'Potřebuji "uvozovky"'
9 >>>
```

Jak jste si jistě všimli, *Python* dává při zobrazení stringů přednost apostrofům. Uvozovky použije k ohraničení textu pouze tehdy, je-li ve vypisovaném textu použit apostrof, jak učinil na řádku 6.

Jak už bylo řečeno, takto označený text se musí vejít na jeden řádek. Chcete-li zadat několikařádkový text a rozhodnete se pokračovat na následujícím řádku před ukončením zadávaného textu uvozujícím znakem, systém ohlásí chybu (viz reakce na řádku 2 ve výpisu 3.2).

Výpis 3.2: *Zápis víceřádkového textu*

```
1  >>> "Neukončený text
2      SyntaxError: unterminated string literal (detected at line 1)
3  >>> """několikařádkový
4  ... text"""
5      'několikařádkový\ntext'
6  >>> '''Jiný způsob
7  ... zápisu'
8  ... '''
9      "Jiný způsob\nzápisu'\n"
10 >>> '''Musí být 'odděleny'''
11     SyntaxError: unterminated string literal (detected at line 1)
12 >>> ''''Apostrof'.'''
13     "'Apostrof'."
14 >>> ""
15     ''
16 >>>
```

Jak ale výpis 3.2 naznačuje, ohraničíte-li zadávaný text trojicemi apostrofů či uvozovek, *Python* bude konec řádku považovat za součást zadávaného textu, dokud nezadáte uzavírající trojici. Jak se můžete přesvědčit na řádcích 5 a 9, při výpisu zadaných textů pak interpret na místech, kde končí jednotlivé řádky zadaného textu, vypíše dvojici znaků `\n` označující právě konec řádku. Tento znak můžete použít k označení konce řádku i vy, ale to si předvedeme v příští kapitole, až budete umět zobrazit, že jste opravdu zadali několikařádkový text.

Jak demonstruje text zadávaný na řádcích 6–8, použijete-li k ohraničení textu trojici znaků, můžete uvnitř textu použít i stejný znak, jaký se vyskytuje v ohraničujících trojicích (například uvnitř textu ohraničeného trojicemi uvozovek se smějí vyskytovat jednoduché či zdvojené uvozovky). Tyto znaky se jenom nesmějí nacházet v bezprostředním sousedství uzavírací trojice.

Apostrof na konci řádku 7 je od uzavírající trojice oddělen koncem řádku. Text zadávaný na řádku 10 však vyvolá chybu, protože je závěrečný apostrof nalepen na uzavírající trojici apostrofů. Jak ale ukazuje text na řádku 12, přilepení na otevírající trojici nevadí. Důležité je, aby mezi případným apostrofem a uzavírající trojicí apostrofů byl nějaký jiný znak. Jak si jistě domyslíte, totéž platí i pro uvozovky.

Reakce na příkaz na řádku 14 ukazuje, že *Python* je schopen pracovat i s *prázdným stringem*, tj. se stringem, který neobsahuje žádný znak.

Programátoři v jiných jazycích budou možná překvapeni, že *Python* nezavádí samostatný datový typ pro znaky. Znaky se v něm zadávají jako jednoznakové stringy. Druhou možností je převést znak na číslo s jeho kódem, a pracovat dále s tímto číslem.

Escape sekvence

Zadávaný text občas obsahuje znak, který neumíte zadat z klávesnice, nebo se to jenom v daném okamžiku nehodí. Ve výpisu 3.2 byl takovým znakem znak konce řádku. Pro takovéto znaky jsou definovány tzv. *únikové posloupnosti*, které však programátoři nenazvou jinak než slangovým *escape sekvence*. Jsou to skupiny znaků začínající znakem \ (zpětné lomítko, anglicky back slash). *Python* definuje následující escape sekvence:

`\\`	Zpětné lomítko (vzhledem k tomu, že je použito jako metaznak uvozující tyto posloupnosti, musíte je zadávat zdvojené).
`\'`	Apostrof (kód `\x27`).
`\"`	Uvozovky (kód `\x22`).
`\a`	Zvukové znamení (BEL – kód `\x07`).
`\b`	Backspace (BS – smazání předchozího znaku – kód `\x08`).
`\f`	Konec stránky (Form Feed – FF – kód `\x0c`).
`\n`	Nový řádek (New Line – NL – kód `\x0a`).
`\r`	Návrat vozíku (Carriage Return – CR – kód `\x0d`).
`\t`	Vodorovný tabulátor (TAB nebo HT – kód `\x09`).
`\v`	Svislý tabulátor (Vertical Tab – VT – kód `\x0b`).
`\OOO`	Znak s 8bitovým kódem zadaným v osmičkové soustavě; `O` představuje osmičkovou číslici – např. `'\101' == 'A'`.
`\xhh`	Znak s 8bitovým kódem zadaným v šestnáctkové soustavě; `h` představuje šestnáctkovou číslici – např. `'\x42' == 'B'`.
`\uhhhh`	Znak s 16bitovým kódem zadaným v šestnáctkové soustavě; `h` představuje šestnáctkovou číslici – např. `'\u263A' == '☺'`).
`\Uhhhhhhhh`	Znak s 32bitovým kódem zadaným v šestnáctkové soustavě; `h` představuje šestnáctkovou číslici – např. `'\U0000266b' == '♫'`.
`\N{název}`	Znak, který má v sadě *Unicode* zadaný `název` – například `'\N{DEGREE CELSIUS}' == '\u2103' == '℃'` nebo `'\N{WHITE SMILING FACE}' == '\u263a' == '☺'`

Při čtení předchozího textu se možná někteří divili, proč ekvivalentnost několika druhů zápisu označuji dvojicí rovnítek. Je to proto, abyste mohli zápis zadat interpretu *Pythonu* (čtenáři elektronické verze knihy jej mohou zkopírovat do schránky a pak zkopírovat do příkazového řádku za výzvu `>>>`). *Python* vám pak odpovědí `True` oznámí, že podle něj jsou porovnávané výrazy ekvivalentní (viz řádky 9 a 10 ve výpisu 3.3), takže je jedno, jakým z uvedených způsobů je zadáte.

Ukázku možností zadání znaků najdete ve výpisu 3.3. Na řádku 1 je zadáno několik znaků ze sady ASCII (znaky s kódem menším než `0x80=128`). První znak se převede z dnes již téměř nepoužívaného osmičkového zápisu do zápisu v šestnáctkové soustavě, ty další pak převede do alternativní, srozumitelnější podoby.

Řádek 3 demonstruje zadání znaků, které sice jsou v používané znakové sadě (fontu), avšak z klávesnice je zadat nelze.

Řádek 5 se pak snaží ukázat možnost zadání znaku názvem. Jak ale vidíte, není-li zadaný znak v použité znakové sadě, vypisuje jej *Python* kódem. Protože kód zadaného znaku je větší než `65 535` (největší číslo zapsatelné pomocí čtyř šestnáctkových číslic), musel pro první z uvedených znaků použít způsob určený pro zápis 32bitového kódu. Další dva zobrazil zadáním 16bitového kódu a závěrečnou mezeru zobrazil jako mezeru.

Zajímavé je, že prostřednictvím kódu zobrazil první tři znaky, přestože druhé dva v tabulce má. Následující mezeru však již zobrazil jako mezeru. Když byl ale na řádku 7 požádán o zobrazení obou zobrazitelných znaků, bez problému je na řádku 8 zobrazil.

Na řádku 9 už pouze demonstruji možnost ověření ekvivalence několika způsobů zápisu, o němž jsem hovořil v poslední poznámce.

Zadáte-li výrazy na řádku 3 nebo 7 na svém počítači, nemusí se vám příslušné znaky zobrazit. Záleží na tom, jaký používáte font.

Výpis 3.3: *Zadání znaků pomocí escape sekvencí*

```
1  >>> '\007\x09\x0a\x0d\x22\x27'
2      '\x07\t\n\r"\''
3  >>> '\u263a'
4      '☺'
5  >>> '\N{ZOMBIE}\N{DEGREE CELSIUS}\N{WHITE SMILING FACE}\N{SPACE}'
6      '\U0001f9df\u2103\u263a '
7  >>> '\N{DEGREE CELSIUS}\N{WHITE SMILING FACE}'
8      '℃☺'
9  >>> '\N{WHITE SMILING FACE}' == '\u263a' == '☺'
10     True
11 >>>
```

Bílé znaky

V souvislosti se zadáváním textů bych měl připomenout rozšířený termín *bílý znak* (*whitespace character*), kterým označujeme znaky reprezentující prázdná místa, například mezeru, tabulátor, svislý tabulátor, konec řádku, konec stránky atd. Zájemci najdou jejich přehled na https://en.wikipedia.org/wiki/Whitespace_character.

3.2 Komentáře

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

V *Pythonu* jsou komentáře uvozeny znakem # (mříž) a ukončeny koncem řádku. Komentáře, které mohou zabírat více řádků, známé z některých jiných jazyků, *Python* nezavádí. Potřebujete-li zapsat několikařádkový komentář, musíte každý řádek komentáře začít znakem #.

Alternativní možností zápisu víceřádkových komentářů (a možností hojně využívanou) je zapsat na daném místě místo komentáře textový řetězec. Pokud se totiž v programu objeví text, který se nikam nepřiřazuje, překladač jej ignoruje. Pouze v interaktivním režimu se hodnoty, jež jsou výsledkem zadaného výrazu (například textového řetězce), následně zobrazují.

Jak si ve vhodnou chvíli vysvětlíme a ukážeme,[5] textový řetězec umístěný na počátku definice je *Pythonem* chápán jako *dokumentační komentář* označovaný často jako *docstring* a *Python* vám jej na požádání zobrazí jako součást nápovědy. To platí nejenom pro kód, jenž je součástí standardní knihovny, kterou jste si spolu s *Pythonem* instalovali, ale i pro kód vámi definovaný.

Klasický komentář začínající znakem # můžete zadat na konec téměř libovolného řádku. Jedinou výjimkou jsou víceřádkové texty, protože zadáte-li do nich znak #, bude se překladač domnívat, že tento znak je součást zadávaného textu. Jinými slovy: komentář musíte zadat až za stringem. Vše jsem se snažil předvést ve výpisu 3.4.

Řada programátorů komentáře ignoruje a nastavuje svá vývojová prostředí tak, aby byly komentáře co nejméně nápadné a při pročítání kódu je neobtěžovaly. Protože většinou vytvářím výukové programy, jejichž čtenáři dodatečné informace v komentářích často ocení, mám ve svých prostředích naopak nastaveno, že komentáře jsou podbarveny. Bude tomu tak i ve výpisech komunikace s prostředím a v definicích částí programů v této knize.

[5] Netrpělivé odkážu na pasáž *Dokumentační komentář a atribut* `__doc__` na straně 148.