Zápisky k předmětu Využití počítačů ve fyzice

Pavel Stránský

16. března 2021

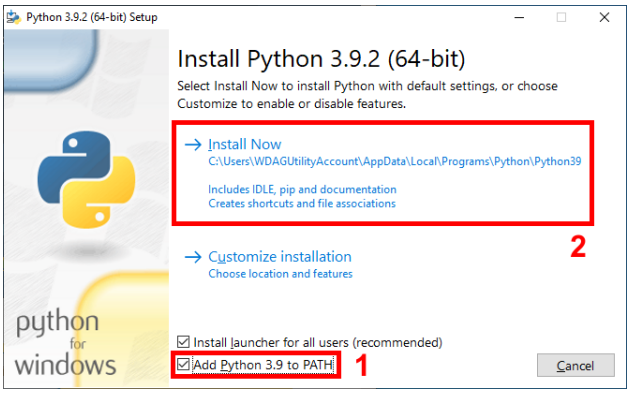
Obsah

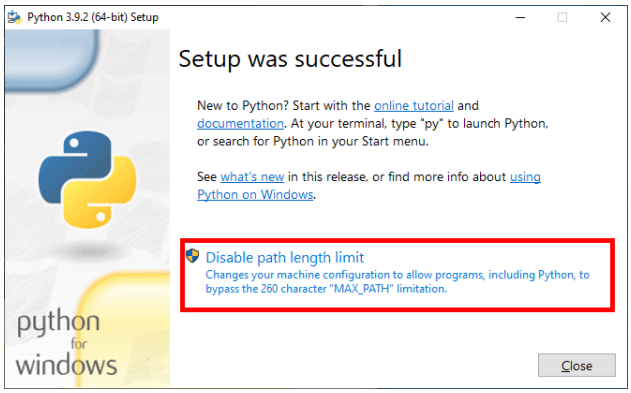
1 Instalace používaných nástrojů

Příklady k cvičení budou demonstrovány v nejnovější verzi programovacího jazyka Python. Jako vývojové prostředí doporučuji Visual Studio Code. Tento volně dostupný program lze nainstalovat na všechny neužívanější operační systémy (Linux, Windows, macOS). Má nepřeberné možnosti při editaci zdrojových souborů, překladu a ladění snad ve všech známých programovacích jazycích. Bohaté možnosti nastavení umožnují přizpůsobit si práci svým potřebám (například zvýrazňování syntaxe, klávesové zkratky či vzhled prostředí). Pomocí doplňků ho můžete integrovat s dalšími službami, například s verzovacím programem Git, či vzdálenými repozitáři, čehož také využijeme. Komunita, která toto vývojové prostředí používá a spravuje, je obrovská, což zaručuje dobrou podporu a rychlé přidávání nových funkcí.

1.1 Instalace Pythonu

Instalační soubor pro svůj operační systém stáhnete ze stránky python.org. Při instalaci na počítač s Windows doporučuji zvolit „Add Python to PATH“, což zjednoduší práci s Pythonem z příkazové řádky, a na poslední obrazovce zvolit „Disable path length limit“:





Pro ověření instalace napíšeme v příkazové řádce příkaz `python`.[1] Tím se spustí REPL[2] Pythonu, ve které můžeme již psát všechny příkazy programovacího jazyka, které se po zadání ihned provedou a vypíšou výsledek.

1.1.1 Instalace doplňujících knihoven

Samotná instalace Pythonu obsahuje jen minimální množství nejnutnějších knihoven. My budeme využívat ještě následující rozšiřující knihovny:

- `NumPy` (**Num**erical **Py**thon: numerická matematika, řady a vícedimenzionální datové typy),
- `SciPy` (**Sci**entific **Py**thon: algoritmy pro optimalizaci, statistiku, řešení diferenciálních rovnic, lineární algebru, atd.),
- `Matplotlib` (vizualizace, grafy).

K jejich doinstalování slouží modul `pip`. V příkazové řádce napíšeme

```
python -m pip install numpy scipy matplotlib
```

čímž se nainstalují naráz všechny tři knihovny.[3]

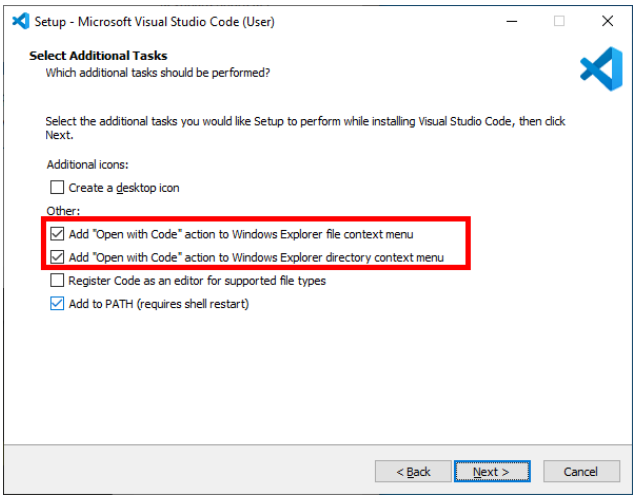
[1] Na počítačích s Linuxem je příkaz v terminálu `python3`.
[2] **R**ead-**E**valuate-**P**rint-**L**oop.
[3] Pokud na počítači s Linuxem uvedený postup nebude fungovat, je potřeba nejprve nainstalovat instalátor `pip` pomocí příkazu `sudo apt install python3-pip`. Pak lze použít buď výše uvedený příkaz, nebo stručnější `pip3 install numpy`.

Existuje samozřejmě celá řada dalších užitečných a používaných knihoven, jako je například `Pandas` pro analýzu dat nebo `SymPy` pro symbolické výpočty, na které v tomto kurzu nedojde.

1.2 Instalace Visual Studio Code

Instalace jazyka Python obsahuje jednoduché vývojové prostředí nazvané IDLE.[4] To však poskytuje jen omezené možnosti co se týče ladění, psaní rozsáhlejších projektů s více zdrojovými soubory nebo integrace s verzovacími programy.

Pro serióznější práci budeme používat Visual Studio Code s doplňkem pro programovací jazyk Python. Instalační soubor stáhnete ze stránky code.visualstudio.com. Během instalace na počítač s Windows doporučuji zvolit obě možnosti „Add Open with Code action to...“,



což zjednoduší otevírání složek s projekty nebo samostatných souborů pomocí pravého tlačítka myši.

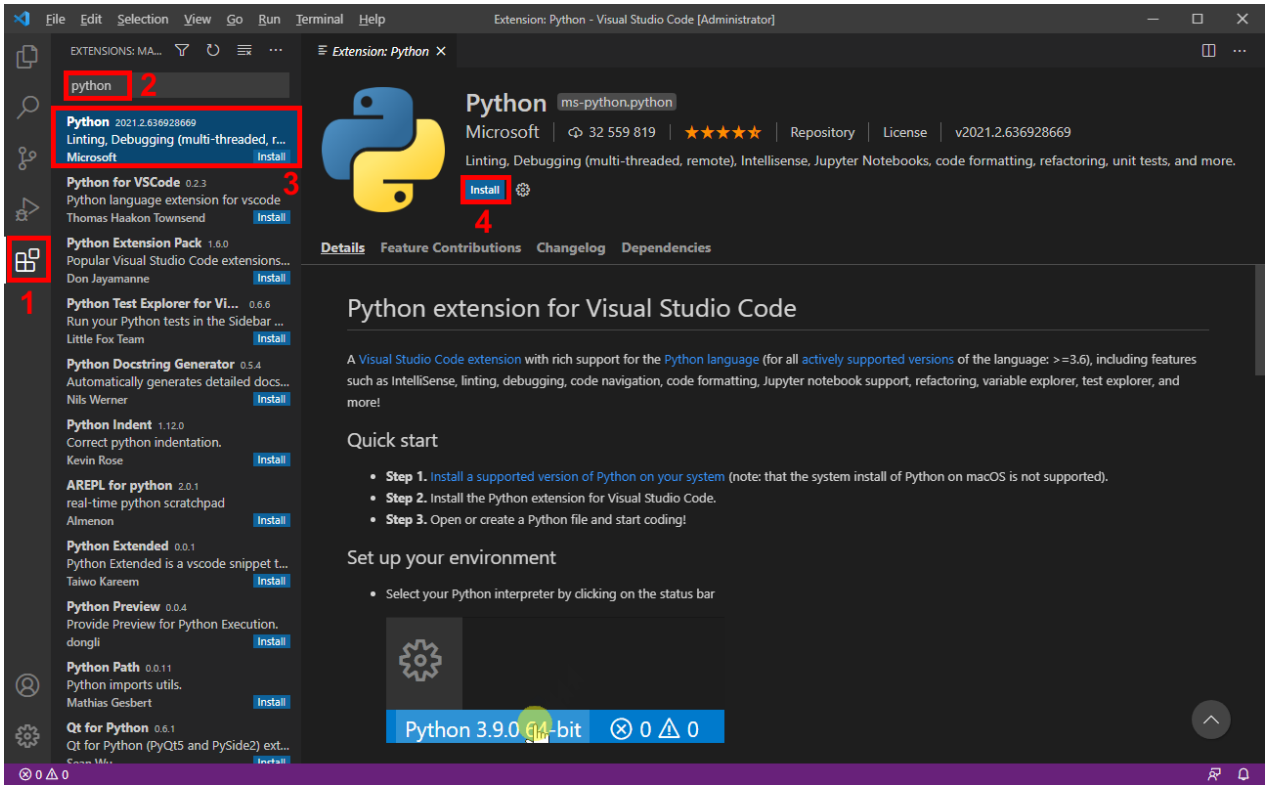
Na operačním systému Linux je nejjednodušší provést instalci pomocí Snap Store příkazem v terminálu

```
sudo snap install --classic code
```

nebo použít tento návod.

[4] **I**ntegrated **D**evelopment and **L**earning **E**nvironment.

1.2.1 Instalace doplňku pro Python

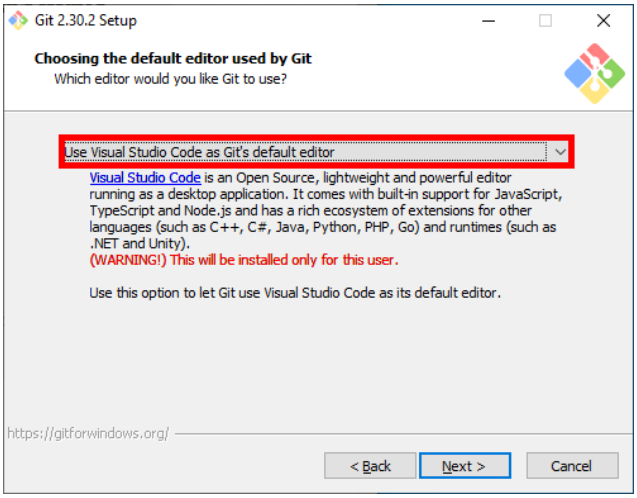


Ke správě doplňků (extensions) se dostanete kliknutím na ikonku 1 nebo stisknutím `Ctrl+Shift+X`. Vyhledáte doplněk Python 2 od Microsoftu, vyberete ho 3 a nainstalujete 4.

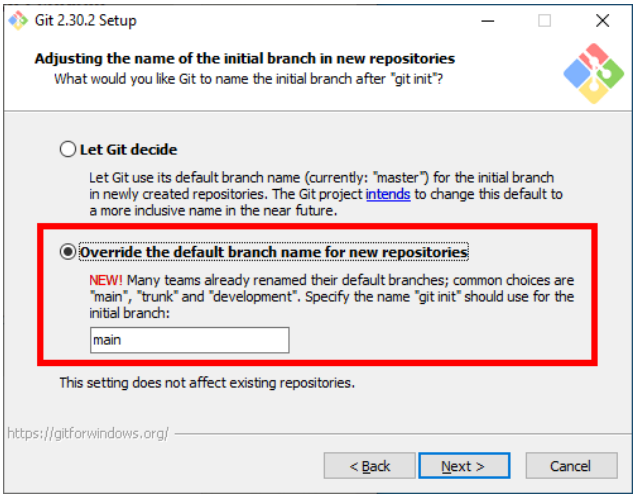
1.3 Instalace Git

Instalační soubor verzovacího systému Git stáhnete z webové stránky git-scm.com. K instalaci Gitu potřebujete administrátorská práva.

V následujícím postupu zobrazuji jen ty snímky obrazovky, na kterých je vhodné vybrat jinou volbu, než jaká je instalátorem standardně nabízena.



Jako editor zvolíme dříve nainstalované Visual Studio Code. Systém Git vyžaduje editor jednak pro povinný komentář každé zapsané změny (commit), jednak pro ošetření kolizí při slučování větví (merge).



Hlavní větev repozitáře se donedávna standardně jmenovala `master`. Vzhledem k negativním konotacím tohoto slova v angličtině se přechází na neutrálnější označení. Nejběžnější je `main`, na které již přešel i populární správce vzdálených repozitářů GitHub. Doporučuji tedy zvolit pojmenování `main`.

Všechna nastavení lze samozřejmě kdykoliv po instalaci změnit.

2 Úvod do používaných nástrojů

V této sekci naleznete základy použití verzovacího systému Git a úvod do syntaxe a idiomatiky jazyka Python. Na rozdíl od zbytku poznámek bude tato sekce v průběhu cvičení postupně doplňována podle toho, s jakými technikami se seznámíme v hlavní části cvičení.

2.1 Verzovací systém Git

Každý si patrně už někdy v životě zasteskl, že nemá uložené dřívější verze svých souborů. Změny, které se ukazují být nevhodné či provedené omylem (kočka nepozorovaně přejde po klávesnici a my pak soubor uložíme), kamarád, který z nějakých důvodů chce tu verzi vašeho programu, kterou jste mu poskytli před rokem, a vy jste mezitím kód zásadně přepsali, to vše jsou důvody k předsevzetí nějakým způsobem důležité milníky v práci na projektu archivovat.

Triviální verzování může spočívat například v ručním ukládání kopií projektu s daty či jinými označeními jednotlivých verzí. Takovýto postup je ale velmi těžkopádný, jelikož v každé kopii musí být uloženy všechny soubory projektu, i pokud se v nich od předchozí verze nic nezměnilo. Obtížně se vyhledává, co přesně se mezi jednotlivými verzemi změnilo a kdo změnu provedl, pokud na projektu pracuje větší tým, přičemž obtížnost roste s velikostí a komplexností projektu. Pro usnadnění uchovávání historie změn proto vznikly verzovací systémy.

Verzovací systém pomáhá udržet historii změn souborů projektu, přičemž ke každému snímku historie se lze vrátit. Vše provádí chytře a v maximální míře automaticky. Systém sám sleduje, v jakých souborech byly provedeny změny. Pokud se jedná o textový soubor, umí porovnat aktuální a libovolnou historickou verzi řádek po řádku a zvýraznit odlišnosti. Umožňuje pracovat s nezávislými vývojovými větvemi (*branches*), ve kterých lze například zkoušet různý přístup k řešení daného problému a mezi kterými lze jednoduše přepínat. Vybrané řešení lze pak snadno začlenit (*merge*) do hlavní vývojové větve.

Soubory projektu a informace o jejich historických změnách se nazývá souhrnně *repozitář*. V něm se uchovávají nejen verze souborů, ale také údaje o tom, kdo a kdy změny provedl. To předurčuje verzovací systémy pro efektivní správu týmových projektů, kdy každý člen týmu pracuje na určité části projektu a své změny následně do projektu včleňuje.

My budeme používat verzovací systém Git. Ten patří mezi *distribuované* verzovací systémy, což znamená, že každý uživatel má na svém počítači celý obsah repozitáře a pouze ve chvílích, kdy uzná za vhodné nebo kdy je k tomu příležitost, může své změny synchronizovat se *vzdáleným repozitářem*, ve kterém se shromažďují změny od všech členů týmu, začleňují do hlavní vývojové větve projektu a hlídají případné kolize. Výhody tohoto přístupu oproti centrálně řízeným verzovacím systémům jsou následující:

1. Práce na projektu nevyžaduje připojení k nějakému centrálnímu serveru s repozitářem, a tedy můžete pracovat offline klidně někde v džungli nebo na Marsu.

2. Není potřeba spravovat zvlášť server s repozitářem a zvlášť klientské počítače. Vše je na jednom místě.

3. Při práci na týmovém projektu není případná porucha počítače spojená se ztrátou dat žádná katastrofa, protože ostatní kolegové mají celou kopii repozitáře na svých počítačích.

Git nejefektivněji funguje na textové soubory, ale zvládne verzovat i soubory binární (například obrázky).

Git je v dnešní době jeden z nejpopulárnějších verzovacích systémů. Pod jedho správou jsou vyvíjeny i velké projekty, například samo Visual Studio Code. Tento projekt je navíc otevřený (open source), což znamená, že kdokoliv, tedy i vy nebo já, má přístup k celému repozitáři, tedy ke všem zdrojovým kódům a k jejich kompletní historii. Může se do práce na projektu zapojit, přispět k jeho vývoji a začít psát historii sám.

Program Git pochází z prostředí Linuxu, a proto je navržený tak, aby se s ním dalo pracovat z příkazové řádky (terminálu) pomocí jednoduchých textových příkazů. Tímto způsobem lze používat všechny dostupné funkce Gitu. My se s nejdůležitějšími funkcemi seznámíme právě pomocí textových příkazů, protože na nich se nejlépe naučí, jak tento Git funguje a jaké jsou jeho možnosti. Jakmile si člověk ujasní principy verzování pomocí Gitu, nabízí se spousta nástrojů a doplňků pro různé programy, které práci s repozitáři usnadní a zrychlí. Obsluha Git repozitářů je ostatně integrována i do vývojového prostředí Visual Studio Code.

Na stránkách projektu Git najdete podrobný interaktivní návod ke všem funkcím verzovacího systému (a to částečně i v češtině).

2.1.1 Prvotní nastavení

Git nelze používat do té doby, než jsou nastaveny základní informace o uživateli. To se provede pomocí příkazů v příkazové řádce (terminálu)

```
git config --global user.name "..."
git config --global user.email "..."
```

přičemž za ... doplníte své jméno (přezdívku) a email. Těmito údaji se podepisují všechny zapsané změny v repozitáři. Pokud tedy spolupracujete na projektu s jinými lidmi, je dobré zvolit takové údaje, pomocí kterých vás kolegové snadno identifikují a úspěšně kontaktují.

Výpis všech nastavení získáte příkazem

```
git config --global
```

Uvidíte, že v mezi nastaveními je i název hlavní větve repozitáře (`init.defaultbranch=main`) a cesta k nastavenému textovému editoru (v našem případě Visual Studio Code).

Všechna nastavení a práci s příkazem `config` najdete v tomto podrobném návodu nebo zadáním příkazu

```
git help config
```

2.1.2 .gitignore

Překladače programovacích jazyků často vytvářejí v adresáři vašeho projektu dočasné pomocné soubory, které nechcete, aby se staly součástí repozitáře (tyto soubory nenesou žádnou relevantní informaci, navíc mohou na různých počítačích vypadat jinak podle toho, jaký překladač či jaké vývojové prostředí zrovna použijete). Abyste mohli používat příkazy pro hromadné sledování či zapisování souborů `git add *` či `git commit -a`, musíte GITu naznačit, jaké soubory má ignorovat. K tomu slouží soubor `.gitignore`.

Každé pravidlo v souboru `.gitignore` zabírá jeden řádek. Řádek, který začíná znakem #, je ignorován a může sloužit například jako komentář. Příklady jednotlivých řádků:

- `tajne.txt`

  Ignoruje soubor s názvem `tajne.txt` (může obsahovat třeba přihlašovací údaje k nějaké službě a ty rozhodně nechceme sdílet ani archivovat; nezapomeňte, že co je jednou zapsané v repozitáři, z něj až na výjimky nelze odstranit).

- `*.log`

  Ignoruje všechny soubory s příponou `log`,

- `!important.log`

  ale neignoruje soubor `important.log`.

- `*.[oa]`

  Ignoruje všechny soubory s příponou `o` nebo `a`.

- `temp/`

  Ignoruje všechny soubory v podadresáři `temp`.

- `doc/**/*.pdf`

  Ignoruje všechny soubory s příponou `pdf` v podadresáři `doc` a ve všech jeho podadresářích. Neignoruje však soubory s příponou `pdf` v hlavním adresáři projektu.

Další příklady jsou například zde.

Pokud na GitHubu zakládáte nový projekt, můžete upřesnit, jaký programovací jazyk budete používat a GitHub automaticky vytvoří optimální soubor `.gitignore`.

Úkol 2.1: *Podívejte se do souboru `.gitignore` v repozitáři k těmto zápiskům. Zatímco Python si téměř žádné pomocné soubory nevytváří, LaTeX jich generuje požehnaně. Proto je tento soubor celkem dlouhý.*

2.1.3 Cheat Sheet

Pro snadnou orientaci v použití Gitu bez nutnosti pamatovat si všechny příkazy je dostupný Git Cheat Sheet nebo tento interaktivní web.

2.2 Programovací jazyk Python

Python je v dnešní době velmi populární programovací jazyk, který pronikl do spousty rozmanitých odvětví, a to zejména díky bohatosti a pokročilosti knihoven, které jsou vyvíjeny komunitou a jsou díky tomu aktuální a odladěné (či průběžně odlaďované). Python existuje na mnoha platformách od osobních počítačů po mikrokontroléry a dá se v něm naprogramovat téměř cokoliv: pokročilé vědecké výpočty používající nejnovější numerické algoritmy, dobře vypadající grafy, statistická analýza, analýza velkých dat a strojové učení, ale také webové služby, okénkové programy či editace obrázků a videí. Je běžné, že i komerční programy obsahují Python coby skriptovací jazyk, čímž umožňují uživateli používat své vlastní kódy „uvnitř“ komerčního produktu a integrovat různé programy mezi sebou.[5]

Python lze charakterizovat jako jazyk:

- *Interpetovaný*, což znamená, že provádění programů probíhá přímo ze zdrojového kódu.[6] Ke spouštění kódů je tedy potřeba mít nainstalovaný interpret, což jsme učinili v sekci 1.1. Pokud kód obsahuje syntaktickou chybu, je odhalena až ve chvíli, kdy se na ni interpret při provádění kódu narazí. U interpretovaných jazyků se neztrácí čas překladem do strojového kódu a je pro ně přirozené dynamické typování proměnných. Tím, že interpet čte text kódu příkaz po příkazu, je provádění programů pomalejší, ale vzhledem k tomu, že velká část časově náročných funkcí a knihoven bývá napsána v rychlejších programovacích jazycích, není to zásadní nevýhoda.

  Opakem intepretovaných jazyků jsou jazyky kompilované, přičemž kompilace se provádí buď přímo do strojového kódu daného procesoru[7] nebo do mezikódu. Ten ke spuštění vyžaduje dodatečnou kompilaci, která však může být vysoce optimalizovaná přímo pro danou hardwarovou konfiguraci použitého počítače.[8]

- *Dynamicky typovaný*, což znamená, že proměnná nemusí mít předem daný typ a typ proměnné se může za běhu programu dokonce měnit.

[5] Jako příklad poslouží nejnovější verze programu Mathematica, Origin či SAS.
[6] Mezi další známé interpetované jazyky patří například JavaScript nebo PHP.
[7] Například C/C++, Pascal nebo Fortran.
[8] Zde se jedná například o jazyky Java, C# a celý .NET framework nebo Julia.

- *S automatickou správou paměti*, takže programátor se nemusí starat o alokování a uvolňování paměti. Python při inicializaci proměnné paměť alokuje automaticky a ve chvíli, kdy proměnnou přestaneme používat, paměť uvolní.[9]

- V Pythonu lze pythonovským způsobem používat tři základní programovací paradigmata: *procedurální, objektové a funkcionální.*

Python klade důraz na jednoduchost, stručnost a čitelnost kódu. Filosofie programovacího jazyka je shrnuta v Zenu Pythonu,[10] se kterým vám doporučuji se seznámit.

2.2.1 Vytvoření a spuštění kódu

Ve Visual Studio Code vytvoříme nový soubor a uložíme si ho s příponou `*.py`. Podle této přípony VS Code pozná, že se jedná o pythonovský kód a použije k práci s ním správný doplněk.

Hotový kód lze spustit ve VS Code jedním z následujících tří způsobů.

1. *Kliknutím na zelenou šipku na nástrojové liště*: Tímto způsobem se spustí celý soubor kódu.

2. *Stiskem* `F5` (a výběrem Python File): Kód se spustí v režimu ladění (debugger). Zastaví se na každém kontrolním bodě (breakpoint, který se vloží na vybranou řádku kódu klávesou `F9`) nebo na každé chybě. Pro pokračování provádění kódu stačí stisknout buď znovu `F5`, nebo `F10` či `F11` pro jeden krok.

3. *Označením části kódu a stiskem Ctrl+Enter*: V okně `TERMINAL` spustí REPL Pythonu a v něm označenou část kódu. REPL se po provedení kódu nezavře, takže v něm lze psát dodatečné příkazy. Pro ukončení REPL do něj stačí napsat `exit()` nebo v něm stisknout `Ctrl+Z` a potvrdit. REPL musí být ukončen, než se použije jakákoliv z dvou dříve popsaných metod spuštění kódu.

2.2.2 Základy syntaxe Pythonu

Soubor se vzorovými příklady najdete v repozitáři ke cvičení: `PythonBasics.py`.[11] Doporučuji vám si ho stáhnout nebo jeho obsah překopírovat a důsledně si ho řádek po řádku projít a spouštět nejlépe pomocí označení a stiskem `Ctrl+Enter`, jak bylo popsáno v předchozí sekci. Některé části kódu odkazují na proměnné zavedené v dřívější části souboru, proto kód procházejte postupně od začátku do konce.

Z vzorového kódu bych vypíchl následující body:

1. *Základní datové typy a operátory*: Zaměřte se na možnosti formátování řetězců. Python obsahuje jen dva základní číselné typy: `int` a `float`. Zbytek je podobný jako v jiných programovacích jazycích.

2. *Proměnné a kolekce*: Důležité standardní kolekce jsou *seznam* (list) [...] a *slovník* (dicctionary) {...}. Dále existuje typ *n-tice* (tuple) (...) a *množina* (set) {...}. O jaký typ kolekce se jedná poznáte podle typu závorek, kterými je uzavřena. Python nabízí bohaté možnosti indexování prvků kolekcí.

   Python nemá typ „řada“ (jedno či vícerozměrný soubor hodnot stejných typů). Ten je implementován až v rozšiřujících knihovnách, například v knihovně **`numpy`**, které se budeme věnovat později.

[9] Algoritmus se nazývá *Garbage collection.*
[10] Zen se také vypíše, pokud do svého kódu zadáte `import this`.
[11] Vzorový soubor je inspirován příkladem Nauč se Python v Y minutách.

3. *Podmínky a cykly*: Příkaz uvozující blok kódu vždy končí znakem „:“. Každý blok je definován svým odsazením, které musí být na každém řádku stejné (tj. musí obsahovat stejný počet odsazujících znaků, mezi které patří buď mezera nebo tabulátor). Konvence je používat k odsazení 4 mezery.

   Cykly jsou možné pouze přes iterovatelné objekty. Pro cyklus přes přirozená čísla (indexy) musíme vytvořit odpovídající iterovatelný objekt příkazem `range`. V drtivé většině se lze při programování v Pythonu indexům zcela vyhnout.

4. *Funkce*: Python umožnuje velkou variabilitu co se týče argumentů funkce a návratových hodnot díky své funkci automatického sbalení a rozbalení kolekcí. Funkce se navíc chová jako objekt, lze ji tedy přiřadit jakékoliv proměnné. To je jeden z prvků funkcionálního programování.

   Funkcionální programování také obsahuje koncept anonymní funkce, což je jednoduchá funkce definovaná na jednom řádku, která nemá vlastní jméno. V Pythonu se vytváří pomocí klíčového slova `lambda`.

5. *Moduly*: Při programování je důležité vhodně strukturovat kód. Python obsahuje jednoduchý koncept modulů, přičemž každý soubor s příponou `*.py` lze použít jako modul. Modul je vlastně speciální typ objektu.

   Dobře se seznamte se způsoby, jak modul načíst a používat, jelikož většina funkcí Pythonu se nachází právě v modulech. Jedná se například o matematické funkce v modulu **`math`** nebo rozšiřující knihovny **`numpy`**, **`scipy`** a **`matplotlib`**.

6. *Třídy*: Python umožňuje objektově orientované programování. Práce s třídami se však v mnohém liší od striktně objektově orientovaných programovacích jazyků, jakými jsou například C++, C# nebo Java. Důležitý rozdíl je například v přístupnosti atributů (všechny atributy v Pythonu jsou veřejné). Rovněž dědičnost a dědění metod se chová odlišně. Dále je nutné pamatovat na to, že každá metoda třídy musí mít v deklaraci jako první argument odkaz na instanci, se kterou je volána (konvenčně se označuje `self`). Ve vzorovém kódu je k objektovému programování opravdu jen to nejnutnější minimum. Pokud chcete v Pythonu využívat objekty seriózně, doporučuji pročíst si odpovídající kapitolu v manuálu.

Uvedený kód s příklady obsahuje jen ty struktury jazyka Python, které budeme používat. V budoucích cvičeních se ještě seznámíte se *správou kontextu* (klíčové slovo `with`). Python umožňuje navíc

- ošetření chyb pomocí *výjimek* (klíčová slova `raise`, `try`, `except`, `finally`),
- tvorbu *generátorů* (klíčové slovo `yield`),
- *asynchronní programování*, korutiny a úkoly (klíčová slova `await`, `async`)
- či tvorbu a použití *dekorátorů*.

Pro plné ovládnutí jazyka vám doporučuji se v budoucnu s těmito koncepty seznámit, a to buď ze specializovaných přednášek, z tutoriálů a návodů na webu, nebo studiem cizích kódů.

Na oficiálních stránkách Pythonu najdete tutoriál k nejnovější verzi programovacího jazyka.

2.2.3 Pojmenování proměnných a formátování

Formátování kódu Pythonu je celkem volné. Povinné je dodržet jen správné odsazení bloků. Pro snazší čitelnost kódu byla nicméně vytvořena řada doporučení, která najdete souhrnně na stránce PEP 8.[12] Python obsahuje dokonce knihovnu **`autopep8`**, která váš zdrojový soubor podle těchto pravidel naformátuje. Automatické formátování podle PEP 8 lze nastavit i ve Visual Studiu Code.

[12] PEP = **P**ython **E**nhancement **P**roposal

Pro pojmenování proměnných existuje jediné závazné pravidlo, které zní, že indentifikátor se nesmí shodovat s žádným z 35 klíčových slov jazyka. Kromě toho jsou ve výše uvedeném dokumentu další nezávazná pravidla k pojmenování proměnných:[13]

- *Proměnné a funkce* se doporučuje pojmenovávat malými písmeny a jednotlivá slova spojovat podtržítky, tzv. underscore style (například `pocet_bodu`).
- *Konstanty* se doporučuje pojmenovávat velkými písmeny a jednotlivá slova spojovat podtržítky (například `PLANCKOVA_KONSTANTA`).
- *Třídy* se doporučuje pojmenovávat tak, že jednotlivá slova názvu začínají velkým písmenem a mezi nimi není žádný znak, tzv. Pascal style (například `PostovniAdresa`).
- *Moduly* se doporučuje pojmenovávat krátkými výstižnými názvy složenými jen z malých písmen.
- Kvůli čitelnosti je dobré se vyhnout jednopísmenným označením `l`, `I` a `O`, jelikož takto označené proměnné mohou být snadno zaměněny s čísly `1` a `0`.

Pokud vám vyhovuje jiný styl pojmenovávání, lze ho použít. Je však dobré být v pojmenovávání konzistentní napříč celým projektem.

Častá otázka je, zda proměnné označovat *česky či anglicky*. Jelikož nikdy nevíte, kdo v dnešním propojeném světě bude chtít váš kód použít a třeba i upravit pro své potřeby, doporučuji používat anglická pojmenování.

Snadná čitelnost kódu je podpořena i vhodným psaním *komentářů*. Na drouhou stranu je vhodné vycházet z předpokladu, že dobře napsaný kód je čitelný i bez komentářů. Čitelnost totiž zaručují zejména vhodně a výstižně označené proměnné a správně navržená struktura kódu. Komentovat je dobré jen ty části kódu, kde se používá nějaký ne všeobecně známý trik nebo postup. Nadbytek komentářů je velmi těžké udržovat při úpravách a může vést i k tomu, že po několika změnách kódu nebudou komentáře v souladu s tím, co kód vykonává.

Naopak doporučené je nešetřit popisem, co dělají jednotlivé funkce, jaké jsou jejich argumenty a co vracejí na výstupu. K tomu slouží komentář typu *docstring*, který je vysvětlený v souboru `PythonBasics.py` v sekci o funkcích.[14] Použití *docstringu* usnadní i tvorbu a správu doprovodné dokumentace k celému projektu. Existují například nástroje, které vám z těchto komentářů vytvoří strukturované webové stránky.

2.2.4 Grafy

K vykreslení grafů slouží knihovna `Matplotlib`. V základní instalaci Pythonu není obsažena, je nutné ji doinstalovat podle návodu v sekci 1.1.1.

Jednoduchý příklad vykreslení grafů je v souboru `plot.py` v repozitáři.

[13] Čitelnosti různých stylů pojmenování proměnných se věnují i seriózní vědecké články, například http://www.cs.kent.edu/~jmaletic/papers/ICPC2010-CamelCaseUnderScoreClouds.pdf.

[14] Stejně jako v mluvených jazycích se zlepšujete čtením literatury, v programovacích jazycích pomáhá čtení cizích kódů. Pro přiklad použití komentářů se koukněte třeba na zdrojový soubor funkce `optimize` z knihovny **`scipy`**.

3 Obyčejné diferenciální rovnice 1. řádu

V této části se budeme věnovat řešení jedné diferenciální rovnice prvního řádu,

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y,t) \tag{1}$$

s počáteční podmínkou

$$y(t_0) = y_0. \tag{2}$$

Zde $y = y(t)$ je hledaná funkce a t je nezávisle proměnná.

Numerické řešení diferenciální rovnice spočívá v nahrazení infinitezimálních přírůstků přírůstky konečnými:

$$\frac{\Delta y}{\Delta t} = \phi(y,t) \tag{3}$$

kde ϕ je funkce, která udává směr, podél kterého se při numerickém řešení vydáme. Volbá této funkce je klíčová a záleží na ní, jak přesné řešení dostaneme a jak rychle ho dostaneme.

3.1 Pár důležitých pojmů

- **Explicitní algoritmy:** K výpočtu hodnoty funkce y_{i+1} se vyžadují pouze hodnoty z aktuálních a minulých kroků, tj. y_i, y_{i-1}, atd.

- **Jednokrokové algoritmy:** K výpočtu hodnoty funkce y_{i+1} v následujícím kroku vyžadují pouze znalost hodnoty funkce v aktuálním kroku y_i. Rozepsáním (3) dostaneme

$$\boxed{y_{i+1} = y_i + \underbrace{\phi(y_i,t)}_{\phi_i}\Delta t}, \tag{4}$$

  přičemž počáteční hodnota y_0 je dána počáteční podmínkou. My se omezíme pouze na tyto algoritmy.

- **Lokální diskretizační chyba:**

$$\mathcal{L} = y(t+\Delta t) - y(t) - \phi(y(t),t)\Delta t, \tag{5}$$

  kde $y(t)$ udává přesné řešení v čase t.

- **Akumulovaná diskretizační chyba:**

$$\epsilon_i = y_i - y(t_i) \tag{6}$$

- **Řád metody:** Metoda je p-tého řádu, pokud

$$L(\Delta t) = \mathcal{O}(\Delta t^{p+1}). \tag{7}$$

- **Kontrola chyby řešení:** Chybu numerického řešení diferenciální rovnice lze zmenšit 1) menším krokem, 2) lepší metodou (metodou vyššího řádu). Menší krok však znamená vyšší výpočetní čas. Sofistikované metody proto průběžně mění velikost kroku: když se funkce mění pomalu, krok prodlouží, když se mění rychle, krok zkrátí (tzv. **metody s adaptivním krokem**). Tím se docílí vysoké přesnosti při co nejmenším výpočetním čase.

3.2 Eulerova metoda 1. řádu

$$\phi_i = f(y_i, t_i), \tag{8}$$

tj. krok do y_{i+1} děláme vždy ve směru tečny v bodě y_i.

- Nejjednodušší metoda integrace diferenciálních rovnic.
- Chyba je obrovská, k dosažení přesných hodnot je potřeba velmi malého kroku, což znamená dlouhý výpočetní čas.

3.3 Eulerova metoda 2. řádu

$$\begin{aligned} k_1 &= f(y_i, t_i) \\ k_2 &= f\left(y_i + k_1 \Delta t, t + \Delta t\right) \\ \phi_i &= \frac{1}{2}\left(k_1 + k_2\right), \end{aligned} \tag{9}$$

tj. uděláme jednoduchý Eulerův krok ve směru k_1, spočítáme derivaci k_2 po tomto kroku a vyrazíme z bodu y_i ve směru, který je průměrem obou směrů (doporučuji si nakreslit obrázek).

Ekvivalentní je udělat „Eulerův půlkrok“ a vyrazit z bodu y_i ve směru derivace spočtené po tomto půlkroku:

$$\begin{aligned} k_1' &= f(y_i, t_i) \\ k_2' &= f\left(y_i + k_1' \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\right) \\ \phi_i &= k_2' \end{aligned} \tag{10}$$

3.4 Runge-Kuttova metoda 4. řádu

$$\begin{aligned} k_1 &= f(y_i, t_i) \\ k_2 &= f\left(y_i + k_1 \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\right) \\ k_3 &= f\left(y_i + k_2 \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\right) \\ k_4 &= f\left(y_i + k_3 \Delta t, t + \Delta t\right) \\ \phi_i &= \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \end{aligned} \tag{11}$$

- Jedna z nejčastěji používaných metod.
- Vysoká rychlost a přesnost při relativní jednoduchosti.
- Existují i Runge-Kuttovy metody vyššího řádu p, avšak vyžadují výpočet více než p dílčích derivací k_j. Obecně platí, že metoda řádu $p \leq 4$ vyžaduje p derivací, metoda řádu $5 \leq p \leq 7$ vyžaduje $p + 1$ derivací a metoda řádu $p = 8, 9$ vyřaduje $p + 2$ derivací.

3.5 Odvození metod

Metody se odvozují na základě rozvoje do Taylorovy řady. Volně řečeno platí, že kolik členů Taylorova rozvoje se vezme, takový bude řád metody p.

- *Eulerova metoda 1. řádu.* Odvození je triviální:

$$y(t+\Delta t) \approx y(t) + \underbrace{y'(t)}_{f(y,t)} \Delta t, \tag{12}$$

takže

$$y_{i+1} = y_i + f(y_i, t_i)\Delta t. \tag{13}$$

- *Eulerova metoda 2. řádu.*

$$\begin{aligned} y(t+\Delta t) &\approx y(t) + y'(t)\Delta t + \frac{1}{2} \underbrace{y''(t)}_{\approx \frac{y'(t+\Delta t)-y'(t)}{\Delta t}} \Delta t^2 \\ &\approx y(t) + f(y,t)\Delta t + \frac{1}{2}[f(\underbrace{y(t+\Delta t)}_{\approx y(t)+y'(t)\Delta t}, t+\Delta t) - f(y,t)]\Delta t \\ &\approx y(t) + \frac{1}{2}\left[f(y,t) + f(y(t) + f(y,t)\Delta t, t+\Delta t)\right]\Delta t, \end{aligned} \tag{14}$$

a odtud

$$y_{i+1} = y_i + \underbrace{\frac{1}{2}\left[f(y_i,t_i) + f(y_i + f(y_i,t_i)\Delta t, t_{i+1})\right]}_{\phi_i} \Delta t, \tag{15}$$

což je jen jinak napsané vztahy (9).

Druhá Eulerova metoda 2. řádu (10) se obdrží stejným způsobem, jen při aproximaci druhé derivace se použije poloviční krok:

$$y''(t) \approx \frac{y'\left(t + \frac{\Delta t}{2}\right) - y'(t)}{\frac{\Delta t}{2}}. \tag{16}$$

Úkol 3.1: *Dokončete odvození vztahů (10).*

Analogicky se postupuje při odvození metod vyššího řádu. Jeho složitost však narůstá exponenciálně s řádem metody. U Eulerovy metody 2. řádu se navíc ukázalo, že lze odvodit několik stejně dobrých vztahů. Tato volnost s řádem metody narůstá.

Úkol 3.2: *Naprogramujte Eulerovu metodu 1. a 2. řádu a Runge-Kuttovu metodu. Vyřešte diferenciální relaxační rovnici*

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -y \tag{17}$$

s počátečními podmínkami $y_0 = 1$ *(analytickým řešením je funkce* e^{-t}*). Integrační krok* Δt *ponechte jako volný parametr. Nakreslete grafy řešení* $y(t)$ *pro rozdílné hodnoty integračních kroků, například* $\Delta t = 0.01$ *a* $\Delta t = 0.1$ *pro čas* $t \in \langle 0; 10 \rangle$*.*

Úkol 3.3: *Rozšiřte kód tak, aby počítal průměrnou kumulovanou chybu*

$$\mathcal{E} = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} \left(y_i - y(t_i) \right)^2}, \tag{18}$$

kde $y(t)$ *je analytické řešení diferenciální rovnice. Nakreslete závislost* $\mathcal{E}(\Delta t)$ *pro* $\Delta t \in \langle 0.002; 0.1 \rangle$ *a pro různé metody. Jelikož očekáváme mocninnou závislost dle* (7)*, kde exponent je tím větší, čím větší je řád metody, je výhodné graf* $\mathcal{E}(\Delta t)$ *kreslit v log-log měřítku. V Pythonu použijete místo* `plot(...)` *funkci* `loglog(...)` *z knihovny* `matplotlib.pyplot`*. Ověřte, že získané křivky jsou v souladu s řády použitých metod.*

Úkol 3.4: *Pomocí naprogramovaných metod vyřešte nelineární diferenciální rovnici*

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \sin(ty) \tag{19}$$

s počáteční podmínkou $y_0 = 1$ *a vykreslete graf jejího řešení.*