Část F
Přílohy

V této části naleznete čtyři přílohy: první vám vysvětlí, jak lze ve *Windows* používat substituované disky, druhá vám představí syntaktické diagramy, třetí vás seznámí se základními konvencemi pro psaní programů v *Pythonu* a odkáže vás na dokumenty, v nichž je dané téma rozebráno podrobněji, a čtvrtá poskytne stručný přehled základních inovací v několika posledních subverzích jazyka.

Příloha A
Konfigurace ve Windows

A.1 Definice substituovaných disků

V operačním systému *Windows* můžete používat 26 logických disků – pro každé písmeno abecedy jeden. Většina uživatelů však používá pouze zlomek tohoto počtu. *Windows* umožňují použít volná písmena pro tzv. **substituované disky**, což jsou složky, které se rozhodnete vydávat za logický disk. Protože o této možnosti většina uživatelů neví, a přitom je to funkce velice užitečná, ukážu vám, jak ji můžete využít. Substituované disky se definují pomocí příkazu:

```
SUBST název_disku substituovaná_složka
```

Nejjednodušší způsob, jak definovat ve *Windows* např. substituovaný disk `P:`, je vložit do složky, kterou budete chtít substituovat jako disk `P:`, dávkový soubor s příkazem k substituci. (Písmeno `P` se pro *Python* hodí nejlépe, ale můžete si vybrat jakékoliv jiné, které je na vašem počítači volné.)

Pokud jste ještě nepracovali s dávkovými soubory, tak vězte, že to jsou obyčejné textové soubory, do nichž zapisujete příkazy pro operační systém. Jejich název může být libovolný, ale musí mít příponu bat (zkratka ze slova batch – dávka). V dávkovém souboru budou následující příkazy (na velikosti písmen nezáleží):

```
SUBST P: /d
SUBST P: .
```

První příkaz má za úkol zrušit případnou doposud nastavenou substituci disku `P:` (není-li v daném okamžiku označený disk substituován, systém vypíše chybovou zprávu, ale jinak se nic nestane), druhý příkaz pak substituuje aktuální složku jako disk `P:`.

Soubor umístíte do složky, z níž budete chtít udělat substituovaný disk. Kdykoliv pak tento dávkový soubor spustíte, dávka substituuje složku, v níž je umístěna, jako příslušný disk. Dávka se přitom spouští obdobně jako aplikace – např. poklepáním na ikonu jejího souboru v *Průzkumníku*.

Kdykoliv od této chvíle budete pracovat s diskem `P:`, budete ve skutečnosti pracovat s obsahem substituované složky. A naopak: cokoliv uděláte s obsahem substituované složky, uděláte zároveň s obsahem disku `P:`.

Budete-li chtít mít danou substituci nastavenu trvale, můžete umístit zástupce dávkového souboru do položky Po spuštění ve startovní nabídce. Protože je v každé verzi operačního systému jinde, bude nejlepší, když ji ve startovní nabídce najdete, klepnete na ni pravým tlačítkem a v následně otevřené místní nabídce zadáte **Otevřít**. Tím otevřete okno průzkumníka s touto složkou. Pak v druhém okně průzkumníka otevřete složku s příslušným dávkovým souborem, uchopíte jeho ikonu PRAVÝM tlačítkem myši, přesunete ji do složky nabídky a pustíte. Otevře se místní nabídka (ta se otevře, pouze pokud přesouváte soubor pravým tlačítkem myši), ve které zadáte, že zde chcete vytvořit zástupce, a tím celý proces končí.

Abyste mohli složku substituovat jako nějaký disk, nesmí váš operační systém používat disk označený tímto písmenem. Písmeno může být použito nejvýše pro jiný substituovaný disk, protože tuto substituci můžete před nastavením nové substituce zrušit (pro tento případ je v dávkovém souboru první příkaz s parametrem `/d` – delete).

Používáte-li operační systém *Windows*, můžete urychlit budoucí vyhledání složky s projekty právě tím, že pro ni zřídíte zvláštní substituovaný disk. Kdykoliv se pak obrátíte na příslušný disk, obrátíte se ve skutečnosti k příslušné složce. Pomocí substituce si tak můžete zkrátit cestu k často používaným složkám.

A.2 Nastavování zástupce spouštějícího IDLE

Používáte-li *Windows* a nemáte-li zkušenosti s nastavováním zástupců, doporučuji využít mého zástupce a pouze mu upravit některá nastavení. Postupujte následovně:

1. Otevřete ve svém oblíbeném správci souborů (budu předpokládat, že jím je *Průzkumník* nebo *Total Commander*) složku s doprovodnými programy `68_PYT`.
2. Klepněte na soubor `!IDLE_Python_39.lnk` (používáte-li *Průzkumník*, tak ten příponu zástupců, tj. `lnk`, nezobrazuje).
3. Stiskněte ALT+ENTER. Otevře se okno z obrázku A.1, v němž můžete nastavit parametry zástupce.
4. V poli **Cíl** je zadáno:

```
C:\_PGM\Python\Python39\pythonw.exe
C:\_PGM\Python\Python39\Lib\idlelib\idle.pyw
```

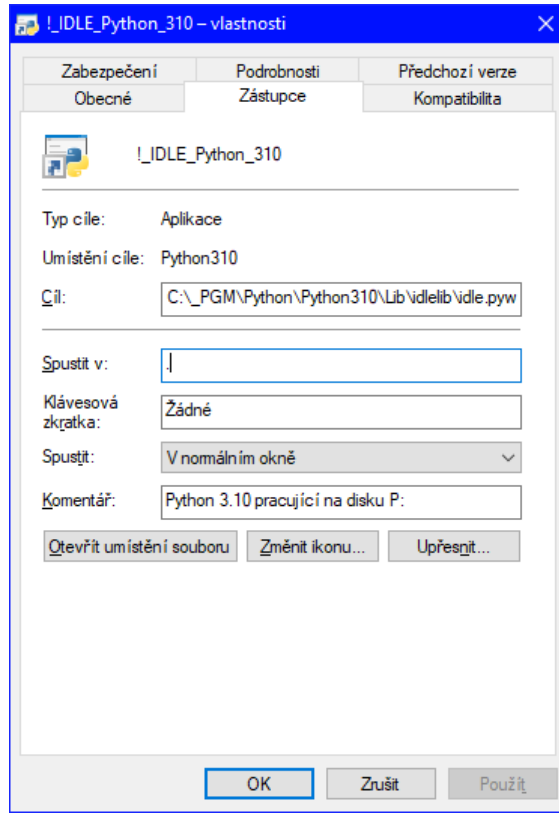
   První část končící `pythonw.exe` je úplná cesta k programu spouštějícímu *Python*, který nebude otevírat okno příkazového řádku, protože očekává, že spouštěný skript používá GUI.

   Tato část je nepovinná. Máte-li korektně instalovanou pouze jednu verzi *Pythonu*, nemusíte ji vůbec zadávat, protože *Windows* poznají, co mají spustit, podle přípony spouštěného skriptu. Já ji zadávat musím, protože mám v době

psaní knihy instalovanou vedle stabilního *Pythonu 3.8.3* také beta verzi *Pythonu 3.9*, kterou v této učebnici používám, a potřebuji si být jistý, která verze daný skript spouští.

Potřebujete-li proto stejně jako já také zadávat spouštěný program, musíte zde zadat úplnou cestu k tomuto programu na vašem počítači.

5. Druhá část příkazu je již povinná a zadává úplnou cestu ke spouštěnému skriptu, tj. k souboru `idle.pyw`. Vy budete mít spouštěný skript nejspíš jinde, tak zde musíte zadanou cestu příslušně upravit.
6. V poli **Spustit v** je zadána cesta ke složce se zdrojovými soubory doprovodných skriptů. Tu si také upravte podle svého počítače.
7. Když máte vše korektně upraveno, uložte upraveného zástupce stiskem **OK**, a od této chvíle můžete IDLE spouštět poklepáním na daného zástupce.



Obrázek A.1:
Okno zástupce spouštějícího IDLE

Příloha B
Syntaktické diagramy

Syntaktická pravidla zápisu složitějších konstrukcí jazyka jsou definována prostřednictvím syntaktických diagramů, které poskytují nejnázornější způsob jejich popisu. Tyto diagramy se v sedmdesátých letech minulého století používaly poměrně běžně, ale vzhledem k pracnosti jejich tvorby se následně z učebnic programování poněkud vytratily a v dnešní době se s nimi setkáte spíše výjimečně.

To ale nic nemění na skutečnosti, že jsou stále nejnázornějším popisem, prostřednictvím nějž si může začátečník ověřit, jak přesně vypadá syntaxe použité konstrukce a kde pravděpodobně udělal ve svém programu chybu. Proto je také ve svých učebnicích používám.

Syntaktické diagramy jsou grafickým ekvivalentem zápisu v EBNF[58] používaném v převážné většině specifikací programovacích jazyků. V této pasáži si na příkladu zápisu čísel ukážeme základní pravidla interpretace syntaktických diagramů.

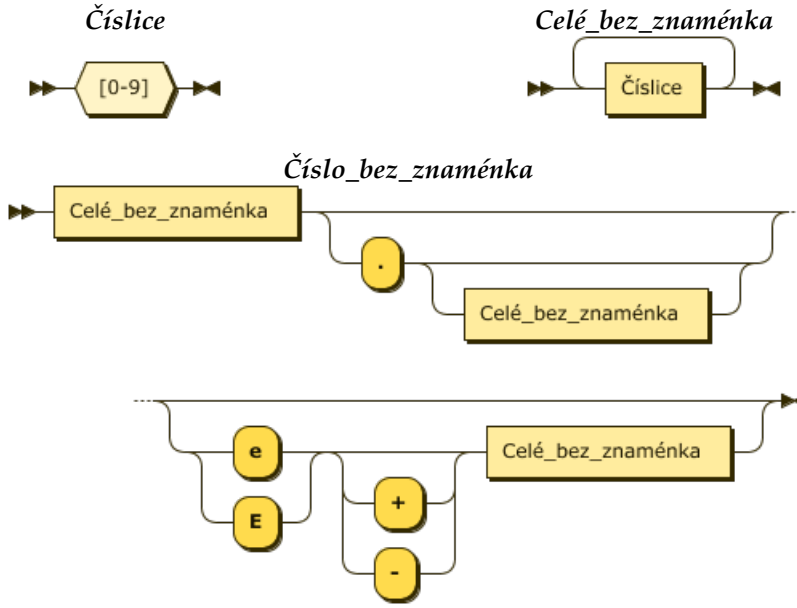
Syntaktické diagramy interpretujeme tak, že jakákoliv cesta od počáteční značky ⏩ (vstupního bodu) po koncovou značku ⋈ (výstupního bodu) generuje syntakticky korektně zapsanou konstrukci jazyka. Pokud existují nějaké výjimky, bývají popsané v doprovodném textu. Cestou procházíte elementy, které budou jednoho ze tří druhů:

- Šestiúhelník obsahuje specifikaci zapsanou přímo v EBNF. Používá se spíše výjimečně, a to ve chvíli, kdy je tento zápis úspornější než odpovídající diagram, aniž by ztratil na své přehlednosti. Zápis v diagramu prvku *Číslice* na obrázku B.1 např. označuje, že za číslici považujeme jakýkoliv znak od nuly do devítky.
- Obdélník označuje prvek, jenž je definován v jiném syntaktickém diagramu.
- Obdélník se zaoblenými rohy obsahuje znak či skupinu znaků, která se má na daném místě vložit (zapsat) do vytvářeného kódu.

Elementy jsou navzájem pospojovány spojnicemi určujícími možné cesty od prvku k prvku při skládání dané syntaktické konstrukce.

[58] EBNF je zkratka termínu *Extended Backus-Naur form* (*Rozšířená Backus-Naurova forma*), což je jazyk používaný k definici syntaxe programovacích jazyků.

Syntaxe zápisu čísel je specifikována prostřednictvím tří syntaktických diagramů na obrázku B.1. Symbol v diagramu ***Číslice*** symbolizuje, že číslicí může být libovolný znak od `0` do `9`. Diagram ***Celé_bez_znaménka*** prozrazuje, že tento druh entity je tvořen nejméně jednou číslicí, ale že číslic může být i více. Diagram ***Číslo_bez_znaménka*** pak specifikuje všechny povolené způsoby, jakými lze zapsat číslo.



Obrázek B.1:
Syntaktický diagram zobrazující pravidla pro zápis čísel

Příloha C
Konvence pro psaní programů v Pythonu

Oficiální konvence jsou včetně demonstračních ukázek podrobně popsány v dokumentu PEP 8,[59] který najdete na adrese https://www.python.org/dev/peps/pep-0008/, a PEP 257, který najdete na adrese https://www.python.org/dev/peps/pep-0257/. První probírá konvence pro psaní kódu, druhý konvence pro psaní dokumentačních komentářů. Zde uvedu jen stručný výčet.

Zájemcům, kteří chtějí rychle ověřit, že nějaký kód vyhovuje těmto konvencím, doporučuji, aby si na adrese https://pep8.readthedocs.io stáhli program nazvaný příhodně `pep8`, jenž dodržování těchto konvencí ověří za vás.

Uspořádání kódu

Hlavní zásadou při tvorbě kódu by měl být fakt, že program se daleko častěji čte, než zapisuje, takže bychom jej měli zapisovat tak, aby se následně dobře četl. Bude-li kód čitelnější, když v daném místě porušíte některou z konvencí, porušte ji. To je obzvláště důležité u programů určených pro výuku.

- Odsazujte o 4 znaky a v textu nepoužívejte tabulátory, ale jen mezery.
- Omezte délku řádků na 79 znaků, u komentářů (i dokumentačních) na 72 znaků.
- U delších řádků vkládejte konec řádku před binární operátor, takže výraz bude na dalším řádku začínat operátorem – např.

```
suma = (první proměnná
        + druhá proměnná)
```

[59] PEP je zkratka z anglického *Python Enhancement Proposal* (doporučení pro vylepšení *Pythonu*). Jsou to dokumenty poskytující informace komunitě *Pythonu* nebo popisující nové funkce, procesy nebo prostředí. Jejich přehled najdete na https://www.python.org/dev/peps/.

- Definice tříd a samostatných funkcí oddělujte dvěma řádky, definice metod uvnitř třídy oddělujte jedním prázdným řádkem.
- Používejte kódování UTF-8, a to bez občas nabízené úvodní deklarace.
- Vkládejte každý import modulu na samostatný řádek, import několika identifikátorů z daného modulu (`from … import …`) je však možné uvést společně.
- Umístěte importy na počátek modulu před definice globálních proměnných.
- Nepoužívejte hvězdičkový import (`from … import *`).
- Identifikátory uvozené a ukončené dvojicí podtržení (tzv. *dunders* jako zkratka z anglického ***d****ouble* ***unders****cores*) by měly být definovány na počátku modulu za dokumentačním komentářem, ale před importy.
- Pro trojité ohraničení stringů používejte trojité uvozovky, abyste byli konzistentní s dokumentačními komentáři podle PEP 257. Jednoduchý ohraničující znak (apostrof či uvozovky) používejte dle svých preferencí, ale buďte konzistentní.
- Mažte závěrečné mezery na koncích řádků. (Některé editory tuto funkci nabízejí.)
- U pojmenovaných argumentů nepoužívejte mezery kolem znaku `=`.
- Nevkládejte více příkazů na jeden řádek.
- U složených příkazů pokračujte na řádku za hlavičkou pouze v případě, že tělo tvoří jeden jednoduchý příkaz – např.

```
for x in lst: total += x
```

- Nebojte se použít závěrečné čárky v seznamech argumentů, mohou usnadnit přidání dalšího členu – např.

```
FILES = [
    'setup.cfg',
    'tox.ini',
    ]
```

- Komentáře, které neodpovídají kódu, jsou horší než žádné. Udržujte komentáře neustále aktuální (*up-to-date*).

Jmenné konvence

Jmenné konvence bohužel nejsou zcela konzistentní, nicméně existuje několik doporučení, které by měly nové programy dodržovat.

- Ve standardní knihovně musí všechny identifikátory používat pouze znaky `ASCII` a měly by pokud možno používat angličtinu. Doporučuje se toto pravidlo dodržovat i v ostatních programech.
- Názvy viditelné uživateli jako součást API by měly reflektovat spíše užití než implementaci.
- Vyvarujte se názvů tvořených pouze znakem `l` (malé L), `O` (velké o) nebo `I` (velké i). Některé fonty neumí odlišit znaky `1-l-I` (jedna – L – I) a `0-O` (nula – o).

- Používejte ve svém vývojovém prostředí font, který tyto znaky odliší.
- Názvy modulů by měly být krátké a používat jen malá písmena. Použití znaku podtržení se nedoporučuje.
- Názvy tříd by měly používat `VelbloudíNotaci`.
- Názvy funkcí by měly používat jen malá písmena a jednotlivá slova názvu oddělovat podtržítky – např. `long_function_name`.
- První parametr instančních metod by se měl vždy jmenovat `self`.
- První parametr třídních metod by se měl vždy jmenovat `cls`.
- Názvy neveřejných metod a instančních proměnných by měly začínat podtržítkem.
- Názvy konstant by měly používat pouze velká písmena a jednotlivá slova názvu oddělovat podtržítky – např. `LONG_CONSTANT_NAME`.
- U každého atributu (proměnné i metody) se vždy rozmyslete, zda má být veřejný. Neveřejný atribut lze snadno zveřejnit, obrácená operace je po rozšíření dané třídy či modulu již zakázaná.

Dokumentační komentáře (PEP 257)

Dokumentační komentář je stringový literál zadaný jako první příkaz v modulu, třídě, metodě či funkci. Ten se automaticky stane hodnotou atributu `__doc__` daného objektu.

- Dokumentační komentář by měly mít všechny moduly a všechny funkce a třídy exportované z těchto modulů.
- Dokumentační komentář balíčku je možné zadat v jeho souboru `__init__.py`.
- Dokumentační komentáře ohraničujte vždy trojitými uvozovkami (`"""`), s případnou předponou `r`, používáte-li v nich zpětná lomítka.
- Trojité uvozovky používejte, i když se komentář vejde na řádek, jak je tomu např. ve výpisu 38.2 na straně 559. Neoddělujte ohraničení mezerami.
- Víceřádkové komentáře zahajte jednořádkovým shrnutím umístěným na stejném řádku s ohraničujícími uvozovkami. Protože může být použito automatickými indexačními programy, je důležité, aby bylo na jednom řádku a aby bylo od dalšího textu odděleno prázdným řádkem. Další řádky komentáře zarovnávejte stejně jako uvozovky na prvním řádku.
- Za dokumentačním komentářem třídy vynechte řádek.
- Dokumentační komentář modulu by měl vypsat všechny třídy, výjimky a funkce daného modulu.
- Dokumentační komentář třídy by měl vypsat její chování, seznam veřejných metod a atributů včetně instančních.
- Dokumentační komentář funkce by měl vypsat její funkci, argumenty a návratovou hodnotu, vedlejší efekty a případná omezení.

Příloha D Stručná historie posledních verzí

Jak jsem již řekl v pasáži *Potřebné vybavení* na straně 29, pokusím se knihu psát tak, aby ji mohli používat i uživatelé starších verzí než verze 3.10, která je uvedena v podtitulu. Nebudu se ale dívat příliš zpět. Kniha předpokládá, že všichni mají instalovánu minimálně verzi 3.6, která vyšla v prosinci 2016 a která je velmi často deklarována jako povinný základ. Následující přehled stručně připomíná verze, jež vyšly v posledních letech. Přidaných vlastností a rysů je samozřejmě mnohem více, uvádím jen ty, které se přímo týkají probíraných konstrukcí. Kompletní přehled najdete v dokumentaci, která je součástí standardní instalace (viz pasáž *Dokumentace* na straně 35).

- Verze 3.6 – prosinec 2016:
  - F-stringy, např. `f'Proměnná x = {x}'`.
  - Anotace proměnných, např. `atr:str`.
  - Podtržení pro zpřehlednění čísel, např. `123_456_789`.
  - Konzole *Windows* akceptuje UTF-8.
  - Slovník iteruje v pořadí přidání.
- Verze 3.7 – červen 2018:
  - Klíčová slova `async` a `await`.
  - Zrušení `ASCII` jako implicitního kódování textu.
  - Příprava odloženého vyhodnocování anotací.
  - Funkce mohou mít více než 255 argumentů.
- Verze 3.8 – listopad 2019:
  - Přiřazovací výraz, např. `pi:=3.14`.
  - Pouze poziční parametry, např. `divmod(x, y, /)`.
  - Přiřazení v nahrazovacích polích f-stringů, např. `f'Proměnná {x=}'`.

- Verze 3.9 – říjen 2020:
  - Slovníky akceptují operátor **|** a **|=**, např. `d |= {k1:v1, k2:v2}`.
  - V anotacích lze používat deklarace kontejnerů standardních typů, např. `iarr:tuple[int]`.
  - Stringům přibyly metody pro odstranění předpon a přípon.
  - Uvolnění gramatických omezení pro dekorátory.
  - Přidána třída `typing.Annotated` pro rozšíření možností anotací.
- Verze 3.10 – říjen 2021:
  - Zavedena měkká klíčová slova.
  - V příkazu `with` je nyní možné použít závorky, a tím umožnit rozložení hlavičky na více řádků.
  - Upřesnění hlášení o syntaktické chybě.
  - Zavedení přepínače `match … case`.
  - Vyhodnocování anotací je implicitně odloženo; anotace se nyní ukládají jako stringy.
  - Datové typy lze nyní sjednocovat, např. anotace `number:int|float` deklaruje, že proměnná `number` bude buď celé, nebo reálné číslo.
  - Zaveden datový typ `typing.Calleable` umožňující deklarovat typy parametrů volatelných objektů.
  - Zavedena anotace `TypeAlias` označující, že anotovaná proměnná bude anotací – hovoří se o anotacích přezdívek typů (TypeAlias Annotation).