Část E
Přílohy

Tato část není součástí tištěné knihy, aby zbytečně nezvyšovala její cenu. Je však součástí elektronických verzí a čtenáři tištěných knih si ji mohou stáhnout.

Příloha A Instalace Pythonu pod Windows

Jak bylo řečeno v pasáži *1.3.2 Instalace Pythonu* na straně 40, instalační soubory najdete na stránce https://www.python.org/downloads/, kde si zadáte, pro který operační systém chcete *Python* instalovat. Poté si stáhnete instalační soubor a spustíte jej.

Ve zbytku této přílohy najdete stručný popis instalace pro nejrozšířenější operační systémy *Windows*, který používá 92 % čtenářů mých knih.

A.1 Vlastní instalace

Při instalaci na systémech *Windows* se vás instalační program nejprve zeptá, jestli chcete spustit instalaci s implicitním nastavením, anebo budete chtít její nastavení ovlivnit (viz obrázek A.1).

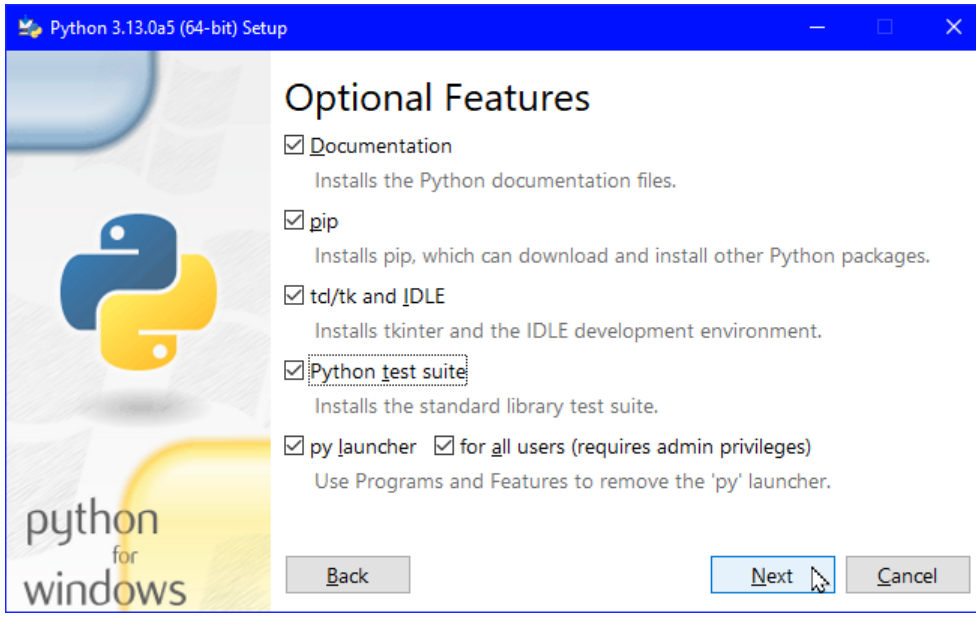
Pod heslem **Install now**, jehož zadáním zvolíte implicitní instalaci, se dozvíte složku, do níž bude *Python* instalován, a to, že součástí instalace bude vývojové prostředí *IDLE* (a s ním samozřejmě grafická knihovna), program `pip` a dokumentace. Současně se vytvoří zástupci a nastaví se spouštění interpretu při poklepání na soubor s příponou z definované sady.

Jste-li opravdoví začátečníci, asi budete volit tuto možnost. Budete-li chtít některé z výše uvedených nastavení změnit, zvolte **Customize installation**, po níž se otevře okno s možnostmi podrobnějšího nastavení.

Před zadáním této volby byste ale měli zaškrtnout políčko **Add python.exe to PATH**, aby se cesta k programu dostala do seznamu prohledávaných cest, a mohli jste pak zadávat samotný název programu bez příslušné cesty k němu.

Obrázek A.1:
Úvodní instalační okno pro Windows

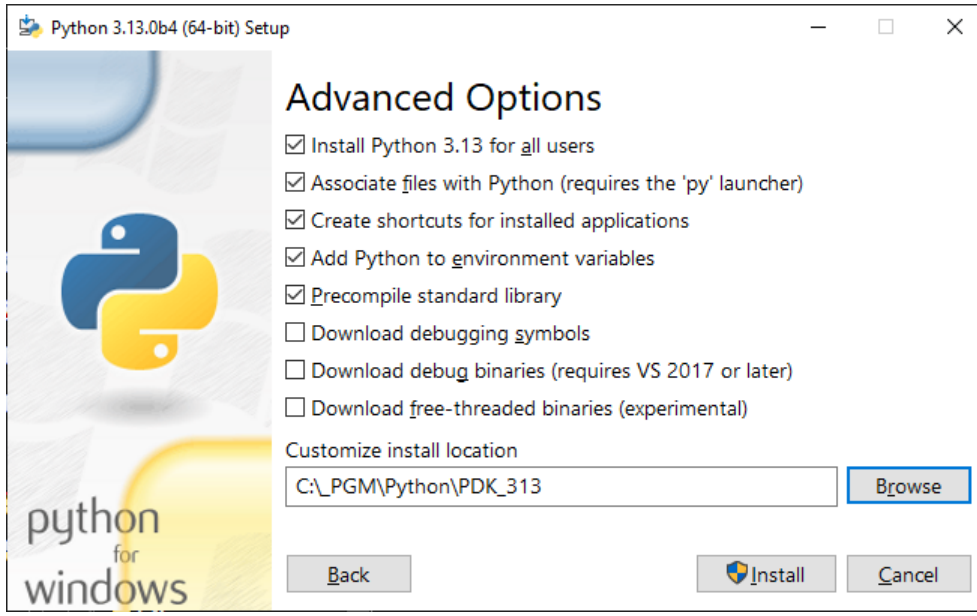
A.1.1 Uživatelská nastavení



Obrázek A.2:
Okno pro nastavení instalovaných součástí

Jak jsem řekl, po aktivaci uživatelského nastavení se otevře okno **Optional Features** (viz obrázek A.2), v němž budete moci zadat, které ze součástí budete, resp. nebudete chtít instalovat.

Po stisku tlačítka **Next** se otevře okno **Advanced options**, v němž můžete nastavit ještě některé pokročilé volby a především pak můžete v poli **Customize install location** zadat, kam chcete danou verzi *Pythonu* instalovat (viz obrázek A.3).



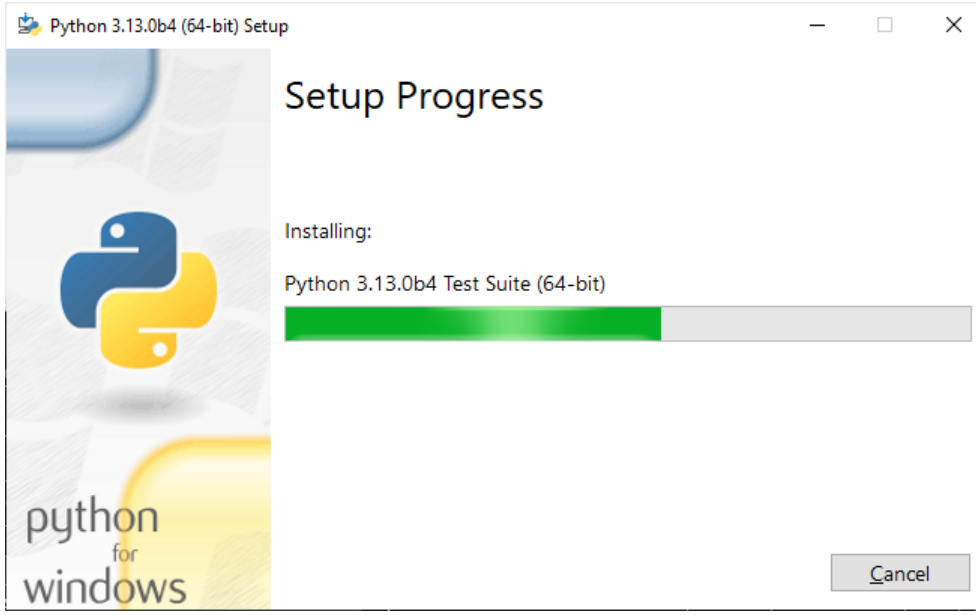
Obrázek A.3:
Okno pro nastavení pokročilejších voleb a umístění programu

Po nastavení všech voleb a případného vlastního umístění stisknete **Install**.

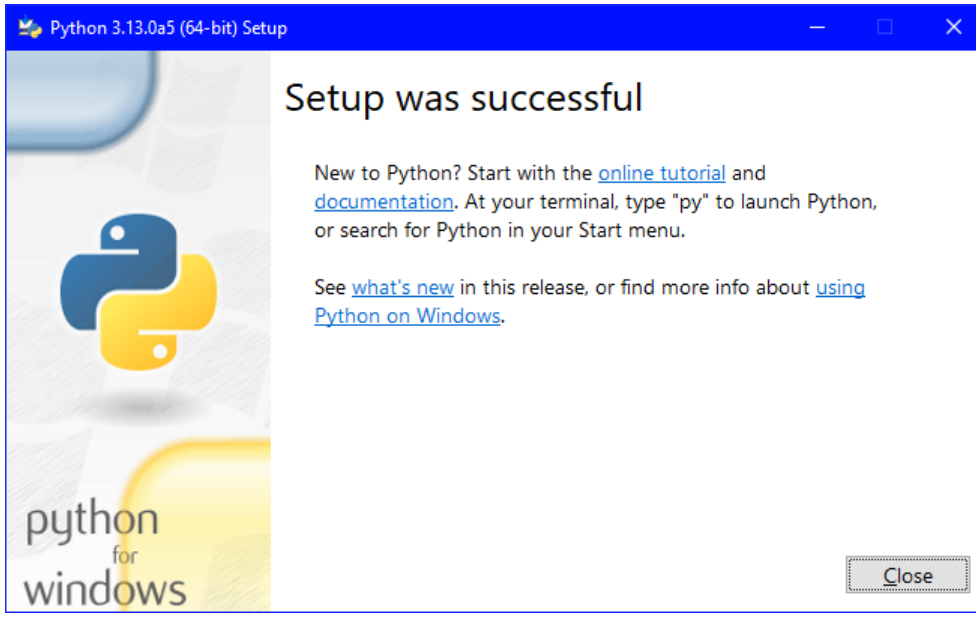
A.1.2 Instalace

Po spuštění instalace na obrazovce vše zhasne (používáte-li více displejů, zhasnou všechny) a systém otevře jediné okno, v němž se vás zeptá, jestli chcete spuštěné aplikaci povolit, aby prováděla změny ve vašem počítači. Samozřejmě jí to povolíte (jinak byste nic nenaistalovali), načež se na všechny displeje vrátí původní obsah a na hlavním se objeví okno **Setup Progress** (via obrázek A.4), které vás informuje o postupu instalace .

Po úspěšné instalaci se otevře okno **Setup was successful** (viz obrázek A.5), v němž vás instalační program informuje, že se vše podařilo, a nabídne odkazy na tutoriál, dokumentaci, informace o novinkách instalované verze a na stránku o rozšířených možnostech, určenou pro zkušenější uživatele s některými nestandardními nároky.



Obrázek A.4:
Indikace postupu instalace



Obrázek A.5:
Indikace postupu instalace

Příloha B
Konfigurace ve Windows

Co se v kapitole naučíte
Tato příloha vás seznámí s některými operacemi, které je vhodné provést při instalaci doprovodných programů v prostředí *Windows*.

B.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 věřte, 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 vytvořit 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.

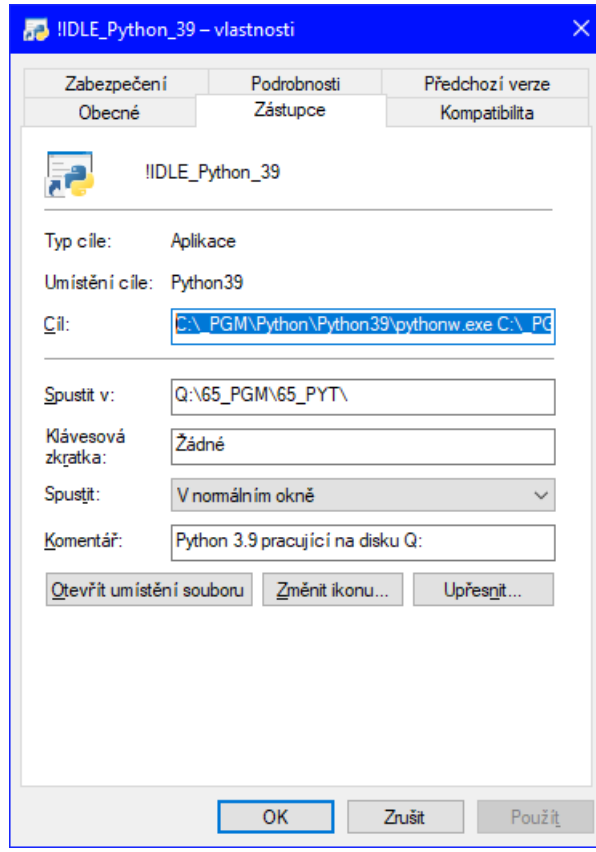
B.2 Nastavování zástupce spouštějícího IDLE

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 `65_PYT`.
2. Klepněte na soubor `!IDLE_Python_39.lnk` (používáte-li *Průzkumník*, ten příponu zástupců, tj. `lnk`, nezobrazuje).
3. Stiskněte ALT+ENTER. Otevře se okno z obrázku B.1.
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.



Obrázek B.1:
Okno zástupce spouštějícího IDLE

Tato část je nepovinná. Máte-li korektně instalovanou pouze jednu verzi *Pythonu*, nemusíte ji vůbec zadávat, protože *Windows* podle přípony spouštěného skriptu poznají, co mají spustit. 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**. Od této chvíle můžete IDLE spouštět poklepáním na daného zástupce.

Příloha C
Použité funkce ze standardní knihovny

C.1 Zabudované funkce

Ležaté hranaté závorky v následujících hlavičkách funkcí označují nepovinné součásti.

abs(x)
Vrátí absolutní hodnotu zadaného čísla (řádky 1 až 4).

```
>>> abs(-3.14)
3.14
>>> abs(1+1j)      #Absolutní hodnotou komplexního čísla je jeho velikost
1.4142135623730951
```

bool(x)
Vrátí logickou hodnotu, na kterou se daná hodnota v případě potřeby převede – viz např. výpis 10.1 na straně 146.

dict(x)
Vytvoří slovník z prvků dodaných generátorem zadaným v argumentu.

divmod(x, y, /)
Očekává jako argumenty dvě (ne komplexní) čísla a vrátí dvojici hodnot obsahující jejich celočíselný podíl a zbytek po dělení. Pro celočíselné argumenty platí, že výsledek je `(x//y, x%y)`. Pro reálné argumenty je to trochu složitější – podrobnosti najdete v dokumentaci.

enumerate(iterable, start=0)
Očekává jako argument zdroj sekvenčních dat. Vrátí instanci třídy `enumerate` generující posloupnost dvojic, v nichž je nultým prvkem klíč a prvním prvkem příslušná hodnota.

eval(výraz[, globals[, locals]])
Vyhodnotí zadaný výraz a vrátí spočtenou hodnotu. Při vyhodnocení může používat zadané globální a lokální proměnné, které však musejí být zadány prostřednictvím slovníku.

float(x=0, /)
Vrátí reálné číslo vytvořené ze zadaného čísla či stringu. Je-li argumentem string, požaduje, aby vyhovoval pravidlům pro zápis reálného čísla, na něž jej převede. Jsou ale povoleny pouze úvodní a závěrečné bílé znaky.

frozenset([iterable])
Vrátí nový objekt typu `frozenset`, volitelně s prvky převzatými z iterable. O typu `frozenset` jsme hovořili v podkapitole *12.3 Druhy kontejnerů* na straně 176.

hash(obj, /)
Vrátí heš-hodnotu objektu (má-li ji), což je celé číslo, které daný objekt do jisté míry charakterizuje. Podrobněji jsme o heš-kódu hovořili v pasáži *12.2.1 Hešovatelné objekty* na straně 175.

help([objekt])
Funkce slouží k získání nápovědy. Protože je ale výklad rozsáhlejší, je mu věnována samostatná podkapitola *5.4 Získání nápovědy – dokumentace* na straně 84.

chr(i, /)
Očekává celé číslo a vrátí string tvořený znakem s daným kódem. Je to vlastně inverzní funkce k funkci `ord()`.

```
>>> chr(0x20ac)    #Aby se znak zobrazil, musí být definován v použitém fontu
'€'
>>> chr(0x1D120)   #Houslový klíč není v použitém fontu => zobrazil se jako kód
'\U0001d120'
```

id(object, /)
Vrátí celé číslo sloužící k identifikaci zadaného objektu. Toto číslo můžeme chápat jako virtuální adresu objektu. Objekty se stejným `id` považuje systém za totožné. Rozdílné objekty musí být na různých místech paměti, a proto musejí mít i rozdílná `id`.

input(*[výzva]*)
Může být volána buď bez argumentů, anebo s argumentem představujícím text výzvy. Vypíše na standardní výstup zadanou výzvu (nebyla-li zadána, nevypíše nic) a čeká, až za ni uživatel zapíše to, k čemu byl vyzván, a stiskne ENTER. Vrátí uživatelem zadaný text.

```
>>> input("Zadej číslo: ")
Zadej číslo: 123
'123'
```

int(x=0, base=10, /)
Prvním argumentem je objekt, který lze převést na celé číslo. Je-li argumentem reálné číslo, vrátí celé číslo představující jeho celou část. Je-li argumentem string, požaduje, aby vyhovoval pravidlům pro zápis celého čísla v desítkové soustavě, na něž jej převede. Přitom jsou povoleny pouze úvodní a závěrečné bílé znaky. Není-li zadán argument, vrátí nulu.

```
>>> int(3.9)    #Při převodu se nezaokrouhluje, ale usekává se desetinná část
3
>>> int (-3.9) #Stejně je tomu i při převodu záporného čísla
-3
```

Je-li zadán druhý argument, pak v prvním argumentu očekává string reprezentující celé číslo zadané v číselné soustavě definované druhým argumentem, který musí být z rozsahu `2<=base<=36`. V zadaném čísle jsou povolena zpřehledňovací podtržítka. Číselné soustavy se základem větším než `10` používají v roli dalších číslic písmena `A-Z`, resp. `a-z`. Velikost použitých písmen přitom nehraje roli.

len(obj, /)
Očekává objekt s více prvky a vrací počet těchto prvků. Je-li argumentem string, vrátí počet jeho znaků.

```
>>> len("")
0
>>> len('12345')
5
```

list(x)
Vytvoří seznam z prvků dodaných generátorem zadaným v argumentu. Podrobněji viz *12.4.2 Vytváření prostřednictvím konstruktorů* na straně 178.

object()
Vytvoří novou instanci třídy `object`.

ord(c)
Při zadání řetězce představujícího jeden znak vrátí celé číslo představující kód zadaného znaku v sadě *Unicode*. Například `ord('a')` vrátí celé číslo 97 a `ord('€')` (znak eura) vrátí 8364. Je to inverzní funkce k funkci `chr()`.

print(argumenty)
Očekává seznam čárkami oddělených výrazů, jejichž hodnoty má vytisknout na standardní výstup. Podrobnosti viz *9.5 Funkce print() a její parametry* na straně 138.

range(stop)
range(start, stop[, step])
Očekává celočíselné argumenty a vrátí objekt, který se používá jako zdroj posloupnosti celých čísel začínající číslem `start`, pokračující čísly zvětšujícími se postupně o `step` a končící posledním číslem menším (při záporné hodnotě `step` větším) než `stop`.

Není-li zadán argument `step`, rostou čísla po jedné. Volání `range(stop)` je ekvivalentní volání `range(0, stop)`, respektive `range(0, stop, 1)`.

reload(modul); přesněji importlib.reload(modul)
Znovu načte zadaný modul. Podrobněji viz *6.7 Oprava načteného modulu* na straně 98.

repr(x)
Vrátí string představující systémový podpis svého argumentu.

set(x)
Vytvoří množinu prvků dodaných generátorem zadaným v argumentu. Podrobněji viz *12.4.2 Vytváření prostřednictvím konstruktorů* na straně 178.

str(object='')
Převede zadaný argument na text představující jeho uživatelský podpis.

super([type[, object-or-type]])
Vytvoří instanci třídy `super` schopnou delegovat požadavek na přístup k zadanému atributu na správného rodiče. Použijete-li bezparametrickou verzi, dodá potřebné argumenty překladač. Volání `super()` je pak ekvivalentní s voláním

```
super(type(self), self)
```

tuple(x)
Vytvoří n-tici prvků dodaných generátorem zadaným v argumentu. Podrobněji viz *12.4.2 Vytváření prostřednictvím konstruktorů* na straně 178.

type(object)
Vrátí typ zadaného argumentu.

C.2 Speciální metody

`__init__()`
Initor instance zodpovídající za její inicializaci.

`__repr__()`
Vrátí systémový podpis své instance. Podrobnosti viz *17.1.10 Metody `__repr__()` a `__str__()`* na straně 230.

`__str__()`
Vrátí uživatelský podpis své instance. Podrobnosti viz *17.1.10 Metody `__repr__()` a `__str__()`* na straně 230.

C.3 Metody třídy `dir`

`fromkeys(iterable[, value]) -> dict`
Vytvoří slovník se zadanými klíči a jednotně inicializovanými hodnotami.

C.4 Metody posloupností – `Sequence`

Z probraných datových typů patří mezi posloupnosti instance tříd `list`, `tupla` a `str`, tj. seznamy, n-tice a stringy.

`index(x [, i [, j ]] ) -> int`
Vrátí index prvního výskytu argumentu `x` v zadané posloupnosti. Je-li zadán argument `i`, začne se hledat od tohoto indexu, je-li zadán argument `j`, přestane se hledat před tímto indexem. Pokud hledanou položku v zadané oblasti nenajde, vyvolá výjimku `ValueError`.

`sorted(x) -> list`
Vrátí nově vytvořený seznam, obsahující seřazená data dodaná zdrojem předaným jako argument. Podrobněji je funkce rozebrána v publikacích [14] a [15].

C.5 Metody třídy `list`

`append(x) -> None`
Přidá na konec seznamu svůj argument.

`sort(*, key=None, reverse=False) -> None`
Seřadí prvky v seznamu podle jejich velikosti. Prvky ale musejí být vzájemně porovnatelné. Příklad použití je na řádku 4 ve výpisu 26.2 na straně 331.

C.6 Metody třídy `str`

`lower() -> str`
Vrátí kopii svého stringu převedenou na malá písmena podle standardu *Unicode*. Příklad použití je na řádcích 18–19 ve výpisu 25.4 na straně 324.

`split(sep=None, maxsplit=-1) -> list[str, ...]`
Vrátí seznam stringů, z nichž je tvořen oslovený string. Tyto stringy jsou v něm odděleny stringem zadaným v argumentu `sep`. Není-li tento argument zadán, považuje se za oddělovač posloupnost bílých znaků.

Je-li zadán argument `maxsplit`, bude vrácen seznam o nejvýše `maxsplit+1` prvcích. Příklad použití je na řádku 6 ve výpisu 26.4 na straně 332.

`upper() -> str`
Vrátí kopii svého stringu převedenou na velká písmena podle standardu *Unicode*.

`strip([ chars ]) -> str`
Vrátí kopii svého stringu s odstraněnými úvodními a závěrečnými znaky vyskytujícími se v zadaném argumentu. Není-li zadán argument, odstraní úvodní a závěrečné bílé znaky. Příklad použití je na řádcích 15–16 ve výpisu 25.4 na straně 324.

Příloha D
Konvence pro psaní programů v Pythonu

Oficiální konvence pro psaní kódu jsou včetně demonstračních ukázek podrobně popsány v dokumentu PEP 8[41] na adrese https://www.python.org/dev/peps/pep-0008/. Konvence pro psaní dokumentačních komentářů jsou v dokumentu PEP 257, který najdete na adrese https://www.python.org/dev/peps/pep-0257/. 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.

D.1 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
```

[41] 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/.

```
        + druhá_proměnná)
```

- 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 *double underscores*) 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 v seznamu prvků závěrečnou čárku – může usnadnit přidání dalšího prvku – např.

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

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

D.2 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.

D.3 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 z něj exportované funkce a třídy.
- 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 9.2 na straně 131. 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.

Část F
Seznamy

Tato část obsahuje seznamy hypertextových odkazů na výpisy programů, obrázky, tabulky a odbočky – podšeděné bloky. Není součástí tištěné knihy, aby zbytečně nezvyšovala její cenu. Je však součástí elektronických verzí, v nichž můžete hypertextové odkazy efektivně využít.

Seznam výpisů programů

Seznam obrázků

Seznam tabulek

Seznam odboček – podšeděných bloků