České vysoké učení technické v Praze FIT

Programování v Pythonu

Jiří Znamenáček

Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.

Praha & EU: Investujeme do vaší budoucnosti

Python – Práce se soubory I

Úvod

Se soubory v Python'u se na první pohled pracuje velmi podobně, jako kdekoliv jinde – soubor z daného umístění otevřeme, pracujeme s jeho obsahem a na konci ho zase spořádaně uzavřeme:

```
f = open('cesta/k/souboru', mode='r', encoding='utf-8')
...
f.close()
```

> → Výchozí mód je `r` a protože je pozičně na druhém místě, stačilo by psát `open('CESTA', 'r')` nebo rovnou `open('CESTA')`. Problém je v tom, že výchozí kódování je závislé na systému, takže `open()` bez parametru *encoding* je v podstatě nepoužitelný (rozhodně nepřenosný).

Protože cestou se může stát mnoho nepředvídaného, je dobré se pojistit a soubor pro případ takových nenadálých událostí stejně pěkně zavřít. Po staru to člověk dělal soustavou bloků `try-finally`, ale v Python'u 3 je mnohem lepší použít příkaz `with` pro vytvoření příslušného operačního kontextu (*runtime context*):

```
with open('cesta/k/souboru', encoding='utf-8') as f:
    BLOK
```

Uvnitř bloku máme pod identifikátorem `f` k dispozici otevřený *stream* (aneb „proud dat“) a můžeme s ním úplně normálně pracovat. Jakmile kód v bloku proběhne, ať už úspěšně nebo neúspěšně, tak je stream automaticky uzavřen (a nemusíme se o to tudíž starat sami).

> → Technicky vzato objekt typu stream zde funguje jako „správce kontextu“ (*context manager*) – při vstupu do bloku je zavolána jeho „magická“ metoda `__enter__()`, při opuštění bloku pak `__exit__()`. Blíže viz B.11 .

Vlastnosti streamu

Otevřený soubor (tedy *stream*, resp. „proud dat“) má mnoho zajímavých vlastností:

```
>>> f = open('cestina.txt', encoding='utf-8')

>>> f
<_io.TextIOWrapper name='cestina.txt' encoding='utf-8'>

>>> dir(f)
['_CHUNK_SIZE', '__class__', '__delattr__', '__doc__', '__enter__'
```

Na mnohé o souboru se můžeme ptát:

```
>>> f.name
'cestina.txt'
>>> f.encoding
'utf-8'
>>> f.mode
'r'

>>> f.readable()    # Je stream možno číst?
True
>>> f.writable()    # Je do souboru možno zapisovat?
False

>>> f.seekable()    # Podporuje stream náhodný přístup? aka Je možn
True
```

Všimněte si, že (dosud) otevřený soubor se patřičně hlásí jako nezavřený a naopak:

```
>>> f.closed
False
>>> f.close()
>>> f.closed
True
```

Módy otevření souboru

Textový soubor (volitelně identifikovaný příznakem 't') můžeme otevřít v několika módech:

- **r** – soubor je otevřen pouze pro čtení
- **w** – soubor je otevřen pro zápis (již existující neprázdný soubor bude smazán)
- **a** – soubor je otevřen pro přidávání (zapsaná data budou přidána na konec)
- **r+** – soubor je otevřen pro čtení i zápis

- **w+** – soubor je otevřen pro čtení i zápis (již existující neprázdný soubor bude smazán)

  → Nezadáme-li mód otevření souboru, je výchozí hodnotou *textový soubor, pouze čtení* 'tr'.

Pro případ **binárního souboru** kombinujeme výše uvedené módy s příznakem 'b' (tedy např. `br` otevře binární soubor pro čtení). Rozdíly z toho plynoucí jsou dva, a to zcela zásadní:

1. *Binární* soubory jsou čteny (a zapisovány) **po bajtech**, *textové* jsou zpracovávány **po znacích** (přičemž jeden znak zabírá podle použitého kódování místo jednoho či několika bajtů).
2. V textových souborech je automaticky prováděna konverze konců řádků různých platforem (`\n` na Unixu, `\r\n` na Windows, `\r` na Mac OS) na jednotné pracovní `\n` (konverze je samozřejmě obousměrná).

Čtení ze streamu

Z otevřeného streamu (proudu) je možno číst data několika různými způsoby. Základní představuje metoda `read()`:

- načtení celého streamu najednou:
  ```
  content = stream.read()
  ```
  Pro binární stream je volání `read()` bez parametrů (resp. s `-1`) v podstatě ekvivalentní zavolání metody `readall()`.

- načtení daného počtu **znaků** (pro případ *textového souboru*) nebo **bajtů** (pro případ *binárního souboru*):
  ```
  part = stream.read(ZNAKŮ|BAJTŮ)
  ```

Vstup *cestina.txt* **:**

```
Příliš žluťoučký kůň úpěl ďábelské ódy.
```

Program *cestina.py* **:**

```
with open('cestina.txt', encodi
    print( f.read(13) )
    print( f.read(13) )
    print( f.read(13) )
```

Výstup:

```
Příliš žluťou
čký kůň úpěl
ďábelské ódy.
```

→ Shodou okolností má tato oblíbená česká testovací věta

právě 39 znaků :-)

Můžeme se ale „vyřádit“ ještě víc, protože mimo jiné lze načítat soubory klasicky pythonovsky po řádcích:

```
soubor = open('soubor.txt', encoding='utf-8')

for řádka in soubor:
    BLOK

for (číslo, řádka) in enumerate(soubor):
    BLOK
```

...nebo po blocích řádek:

```
# otevřeme soubor v textovém módu a kódování UTF-8 pro čtení
>>> f = open('cestina.2.txt', encoding='utf-8')

>>> f.readlines()    # načteme z něj všechny řádky
['Třistatřiatřicet stříbrných stříkaček\n', 'stříkalo\n', 'přes tř
>>> f.readlines()    # jsme na konci, už není co dál číst
[]

>>> f.seek(0)    # vraťme se na začátek
0

>>> for line in f.readlines():    # proiterujme přes seznam vrácený
...     print(line)
...
Třistatřiatřicet stříbrných stříkaček

stříkalo

přes třistatřiatřicet stříbrných střech.

>>> f.seek(0)    # vraťme se na začátek
0

# načtěmež soubor řádku po řádce
>>> line = f.readline()
>>> while line:    # vrácený prázdný řetězec na konci zajistí ukonč
...     print(line)
...     line = f.readline()
...
Třistatřiatřicet stříbrných stříkaček

stříkalo

přes třistatřiatřicet stříbrných střech.

>>> f.close()    # zavřemež pracovní soubor
>>> f.closed
True
```

→ Uvedené cyklení (stejně jako metody `readline()` a `readlines()`) se dá použít i pro binární proudy. Jestli je ovšem k něčemu dobré záleží na konkrétním případu.

Vstup *cestina.2.txt* :

```
Třistatřiatřicet stříbrných stříkaček
stříkalo
přes třistatřiatřicet stříbrných střech.
```

Program *cestina.2.py* :

```
with open('cestina.2.txt', enco
    for (index, řádka) in enume
        print( index, ': ', řád
```

Výstup:

```
0 :  Třistatřiatřicet stříbrnýc

1 :  stříkalo

2 :  přes třistatřiatřicet stří
```

→ Přebytečné odřádkování na výstup probublá z výchozí hodnoty parametru *end* u funkce `print()`.

→ Na běžném terminálu ve Windows máte smůlu, protože nedokáže zobrazit unicodové znaky. Alespoň trochu (ale jen trochu :) si můžete pomoci nastavením proměnné prostředí `PYTHONIOENCODING` na hodnotu `utf_8`.

Znaky vs bajty

Základní třídou pro vstup a výstup je třída `IOBase`. Z ní dědí jak třída `TextIOBase` pro práci s textovými soubory, tak třída `RawIOBase` pro práci se soubory binárními. Pamatovat na rozdíl mezi *bajty* a *znaky* je skutečně důležité:

```
# otevřme textový soubor v kódování UTF-8
>>> f = open('japonstina.txt', encoding='utf-8')

>>> f.tell()   # s čertvě otevřeným souborem stojíme na začátku pr
0
>>> f.read()   # načtěmež celý proud
'狼.cz'
>>> f.tell()   # jsme na jeho konci - v bajtech je delší, než ve z
6

>>> f.seek(0)   # vraťme se na začátek
0
>>> f.tell()
0

>>> f.read(1)   # přečtěmež jeden znak..
'狼'
>>> f.tell()    # ..ale posunuli jsme se o tři bajty dále!
3

>>> f.read(1)   # '.' je ve spodní polovině ASCII-tabulky, takže n
'.'
>>> f.tell()    # ..nás posune pouze o jeden bajt dále
4

>>> f.close()
```

Asi už tušíte, do jakých nesnází se dostanete, pokud se pokusíte na *textovém* proudu posouvat začátky čtení dalšího znaku do nesmyslných míst:

```
>>> f = open('japonstina.txt', encoding='utf-8')
>>> f.tell()
0

>>> f.seek(1)   # posuňme se o jeden bajt po streamu dopředu..
1
>>> f.tell()
1
>>> f.read(1)   # ..a pokusme se nyní načíst jeden znak
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.1/codecs.py", line 300, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, fi
UnicodeDecodeError: 'utf8' codec can't decode byte 0x8b in positio
```

Kdybyste si neměli pamatovat nic jiného: **Nikdy nemíchejte binární a textové streamy dohromady! ^_^**

Zápis do streamu

Podobně jako u čtení ze streamu máme pro zápis k dispozici základní metodu `write()`:

- zapsání daných **znaků** (pro případ *textového souboru*) nebo **bajtů** (pro případ *binárního souboru*):
  `stream.write(ŘETĚZEC|BAJTY)`

```
# otevřme soubor v textovém módu a kódování UTF-8 pro zápis (vlast
>>> f = open('test', 'w', encoding='utf-8')
>>> f.write('Ahoj, světe!')    # metoda write() vrací počet zapsaný
12
>>> f.write('Jak se máš?')
11
>>> f.close()

# podívejme se, co jsme zapsali
>>> f = open('test', 'r', encoding='utf-8')
>>> f.read()
'Ahoj, světe!Jak se máš?'
>>> f.close()
```

A podobně jako u čtení můžeme použít i několik dalších způsobů:

Vstup *cestina.2.txt* **:**

```
Třistatřiatřicet stříbrných stříkaček
stříkalo
přes třistatřiatřicet stříbrných střech.
```

Program *writelines.py* **:**

```
lines = []

with open('cestina.2.txt', mode
    lines = f.readlines()

with open('cestina.3.out', mode
    f.writelines( lines )

with open('cestina.3.out', mode
    print( f.readlines() )
```

Výstup:

```
Třistatřiatřicet stříbrných stř
stříkalo
přes třistatřiatřicet stříbrnýc
```

→ Jak vidno, `writelines()` vezme na vstupu seznam a prvek po prvku ho překopíruje na výstup (takže žádné přebytečné `\n` na konec nepřidává).

Další možností na zápis dat do souboru je použití parametru *file* u funkce `print()`:

Program *print2file.py* **:**

```
with open('print2file.out', mod
    print( 'Ahoj, světe!', file
    print( 'Jak se máš?', file=
```

Výstup:

```
Ahoj, světe!
Jak se máš?
```

→ `print()` na pozadí stejně zavolá metodu `write(string)` předávaného objektu.