Soubory

Petr Pošík

Katedra kybernetiky, FEL ČVUT v Praze

OI, B4B99RPH: Řešení problémů a hry, 2016

Prerekvizity:

- Cykly

Úvod

Informace jsou na počítači uloženy v pojmenovaných "balících", kterým se říká **soubory**. Naučíme se, jak se ze souborů informace čtou a jak se do nich zapisují.

Soubory jsou na disku uloženy v souborovém systému, který je hierarchicky členěn pomocí adresářů (directories) neboli složek (folders). Nejprve si ukážeme základy práce s nimi.

Adresáře a cesty

Adresáře

Adresář (složka) je organizační jednotka souborového systému, která nám umožňuje informace na disku hierarchicky členit. Může obsahovat soubory nebo další adresáře. Každý adresář je sám součástí jiného adresáře (s výjimkou adresáře kořenového).

Kořenový adresář je zvláštní. Je vždy v souborovém systému přítomen a představuje počátek hierarchické struktury. Obvykle se označuje dopředným lomítkem (`/`). Windows mají zvláštní kořenový adresář na každém disku a označují se zpětným lomítkem (`\`).

Každý adresář obsahuje 2 zvláštní položky:

- `.` je odkaz na adresář samotný
- `..` je odkaz na nadřazený adresář

Aktuální pracovní adresář

Aktuální pracovní adresář (current working directory, CWD) je z hlediska Pythonu obvykle ten adresář, odkud jste spustili interpret Pythonu, nikoliv adresář, kde je interpret Pythonu uložen (na některých platformách to může být jinak).

Jaký je váš aktuální pracovní adresář můžete zjistit pomocí funkce `getcwd` z modulu `os`. Podívejme se, jaký je aktuální pracovní adresář právě nyní:

In [1]:

```
import os
print(os.getcwd())
```

```
C:\P\0Teaching\rph\repos\rph-lectures\files
```

Cesty

Cesty (https://en.wikipedia.org/wiki/Path_%28computing%29) jsou sekvence názvů adresářů (někdy zakončené názvem souboru), které jednoznačně určují soubor nebo adresář v souborovém systému.

Absolutní cesty vždy začínají kořenovým adresářem (`/`, forward slash), na Windows často i písmenem označujícím disk (`C:\\`). Příklad:

```
/home/posik/teaching/rph/lectures/files.pdf
```

Relativní cesty se vždy konstruují od aktuálního pracovního adresáře. Předpokládejme, že CWD je `/home/posik/teaching`. Pak relativní cesty

```
prg/lectures/files.pdf
../../svoboda/presentations/upload_system.pdf
```

označují soubory

```
/home/posik/teaching/prg/lectures/files.pdf
/home/svoboda/presentations/upload_system.pdf
```

Pohyb v souborovém systému

aneb změna aktuálního pracovního adresáře: na příkazové řádce OS byste použili příkazy `cd` nebo `chdir`. V Pythonu můžete použít funkci `os.chdir()`:

In [2]:

```
import os
orig_wd = os.getcwd()
os.chdir('/P/0Teaching')
print(os.getcwd())
```

```
C:\P\0Teaching
```

Nyní jsme v jiném pracovním adresáři. A můžeme jej změnit zpět:

In [3]:

```
os.chdir(orig_wd)
print(os.getcwd())
```

```
C:\P\0Teaching\rph\repos\rph-lectures\files
```

Práce s cestami

Modul `os.path` obsahuje funkce pro práci se souborovými cestami:

In [4]:

```
fpath = os.path.abspath('files.pdf')
print(fpath)
```

```
C:\P\0Teaching\rph\repos\rph-lectures\files\files.pdf
```

In [5]:

```
print(os.path.dirname(fpath))
```

```
C:\P\0Teaching\rph\repos\rph-lectures\files
```

In [6]:

```
print(os.path.basename(fpath))
```

```
files.pdf
```

In [7]:

```
print(os.path.splitext(os.path.basename(fpath)))
```

```
('files', '.pdf')
```

Jak správně vytvořit cestu z jednotlivých fragmentů?

In [8]:

```
fpath2 = os.path.join('\\', 'P', 'Teaching')
print(fpath2)
```

```
\P\Teaching
```

Jak získat cestu k adresáři nebo souboru relativní k aktuálnímu pracovnímu adresáři?

In [9]:

```
print(os.path.relpath(fpath2))
```

```
..\..\..\..\..\Teaching
```

Soubory

Typy souborů

- **Textové soubory**:
  - obsahují "čitelné" znaky
  - dají se přečíst v téměř libovolném textovém editoru
- **Binární soubory**:
  - hudební soubory, videa, dokumenty slovních procesorů (MS Word), prezentace, PDF, ...
  - obsahují různé řídicí informace specifické pro daný formát souboru
  - k jejich přečtení/interpretaci potřebujeme zvláštní program, který jejich struktuře rozumí

V dalším se budeme zabývat jen textovými soubory.

Textové soubory

- Zabírají obvykle malé místo na disku (prázdný textový soubor je skutečně prázdný, tj. jeho velikost je 0).
- I přesto mohou mít jistou vnitřní strukturu:
  - Zdrojové kódy
  - Hodnoty oddělené čárkou (CSV)
  - HTML soubory
  - ...

Otevřít a zavřít

Pokud se chcete podívat na obsah šuplíku, nebo pokud do něj něco chcete vložit, musíte šuplík nejprve otevřít. Když jste s prací hotovi, zase jej zavřete. Totéž platí pro soubory.

Když šuplík otevřete, držíte jej za **držadlo**, pomocí něhož s šuplíkem manipulujete a pomocí něhož ho na konci zase zavřete. Držadlo souboru, **file handle**, vám umožní dělat s otevřeným souborem nejrůznější věci. Můžete soubor číst, přesunout se na nějakou pozici v souboru, atd.

Příklad: čtení obsahu souboru

Vytvořme jednoduchý textový soubor `text.txt` v aktuálním pracovním adresáři. (Na ukázku, abychom měli co načítat. Následující řádky ukazují způsob, jakým se dá vytvořit textový soubor v prostředí Jupyter. V Pythonu by to nefungovalo.)

In [10]:

```
%%writefile text.txt
Hello, world!
How are you?
```

```
Overwriting text.txt
```

Nyní můžeme onen soubor načíst pomocí Pythonu a zobrazit jeho obsah:

In [11]:

```
file = open('text.txt', 'r')
contents = file.read()
file.close()
print(contents)
```

```
Hello, world!
How are you?
```

1. První řádek instruuje Python (a operační systém), aby otevřel (**open**) soubor jménem `text.txt` (první argument) a vrátil "držadlo" tohoto otevřeného souboru. Druhý argument, `'r'` (někdy nazýván *mód souboru*), indikuje, že soubor má být otevřen ke čtení. Existuje několik **módů**, v nichž může být soubor otevřen: pro čtení (`'r'`), zápis (`'w'`), přidávání (`'a'`). Mód souboru také specifikuje, zda jej chceme otevřít jako textový nebo jako binární (`'wb'`, `'rb'`, ...).
2. Na druhém řádku se na souboru zavolá metoda `read()`, čímž se načte celý obsah souboru do paměti ve formě dlouhého řetězce. Tento řetězec se pak přiřadí k proměnné `contents`.
3. Na třetím řádku soubor (pomocí držadla) zavíráme.
4. Na posledním řádku vytiskneme načtený obsah souboru.

Kódování řetězců a textových souborů

Řetězce jsou vlastně abstrakce. Ve skutečnosti jsou to jen sekvence bytů, ale tyto byty (a jejich skupiny) jsou interpretovány jako indexy do tabulky symbolů, která obsahuje velká a malá písmena, číslice, zvláštní znaky a mnoho dalších symbolů. Tato tabulka symbolů představuje **kódování**. Stejná sekvence bytů, která v jednom kódování představuje čitelný řetězec, může při použití jiného kódování vypadat jako naprosto nesmyslná změť znaků.

- ASCII: obsahuje 127 znaků, anglická velká a malá písmena, číslice a nějaké symboly. Žádné znaky z jiných národních abeced.
- ...
- **UTF-8**: "Unicode" obsahující téměř jakýkoli znak jakékoli abecedy. Obsahuje ASCII jako svou podmnožinu. **POUŽÍVEJTE TOTO KÓDOVÁNÍ**, kdykoli vám to okolnosti umožní!

To platí i pro textové soubory!

Otevření souboru se specifikovaným kódováním

Funkce `open()` má několik dalších parametrů; jedním z nich je `encoding`. Pokud **explicitně použijete UTF-8 pokaždé**, když otevíráte textový soubor, ušetříte si mnoho nepříjemností:

```
f = open('file_to_open.txt', 'r', encoding='utf-8')
# Do something with f
f.close()
```

nebo

```
with open('file_to_open.txt', 'r', encoding='utf-8') as f:
    # Do something with f
```

Příkaz `with`

Protože každé volání příkazu `open()` by mělo mít odpovídající volání metody `close()`, Python je vybaven příkazem `with`, který automaticky uzavře soubor na konci bloku příkazů. Kód

```
f = open('text.txt', 'r', encoding='utf-8')
contents = f.read()
f.close()
print(contents)
```

je ekvivalentní následujícímu kódu s příkazem `with`:

```
with open('text.txt', 'r', encoding='utf-8') as f:
     contents = f.read()
print(contents)
```

Čtení souboru: `file.read()`

Použijte tuto techniku, pokud chcete všechen obsah souboru načíst do jediného (možná obrovského) řetězce, nebo pokud chcete určit, kolik znaků se má přečíst.

In [12]:

```
with open('text.txt', 'r', encoding='utf-8') as f:
    contents = f.read()
print(contents)
```

```
Hello, world!
How are you?
```

Když je metoda `read()` zavolána bez argumentů, načte celý zbytek souboru (od aktuální pozice v souboru). Když jí předáme celočíselný argument, načte specifikovaný počet znaků a posune aktuální pozici za načtený úsek.

In [13]:

```
with open('text.txt', 'r', encoding='utf-8') as f:
    first_10_chars = f.read(10)
    the_rest = f.read()
print("The first 10 chars:", first_10_chars)
print("The rest:", the_rest)
```

```
The first 10 chars: Hello, wor
The rest: ld!
How are you?
```

Čtení souboru: `file.readlines()`

Použijte tuto techniku, pokud chcete načtením získat obsah ve formě seznamu řetězců (řádků).

In [14]:

```
with open('text.txt', 'r', encoding='utf-8') as f:
    lines = f.readlines()
print(lines)
```

```
['Hello, world!\n', 'How are you?']
```

Všimněte si, že jednotlivé řetězce obsahují také znak konce řádku, `\n`. Poslední řádek jej obsahovat může, ale nemusí. Těchto znaků se lze zbavit metodou `str.strip()`.

In [15]:

```
for line in lines:
    print(line.strip())
```

```
Hello, world!
How are you?
```

Čtení souboru: `for <line> in <file>`

Tuto metodu použijte, pokud chcete udělat totéž s každým řádkem souboru od aktuální pozice do konce souboru. Zatímco předchozí techniky načetly najednou celý obsah souboru (který se nemusí vejít do paměti), tento způsob čte soubor řádek po řádku, což umožňuje zpracovávat velké soubory.

In [16]:

```
with open('text.txt', 'r', encoding='utf-8') as f:
    for line in f:
        s = line.strip()
        print("The line '" + s + "' contains " + str(len(s)) + " characters.")
```

```
The line 'Hello, world!' contains 13 characters.
The line 'How are you?' contains 12 characters.
```

Čtení souboru: `file.readline()`

Tato metoda umožňuje načíst ze souboru vždy jediný řádek, což je užitečné, když chcete takto načítat jen část souboru.

Předpokládejme, že chceme načíst následující textový soubor, který obsahuje několik různých částí. První řádek je stručný popis dat. Další řádky začínající # jsou komentáře. Zbytek souboru obsahuje data.

In [17]:

```
%%writefile data_collatz_5.txt
Collatz 3n+1 sequence, starting from 5.
# The next number in a Collatz sequence is either 3n+1 if n is odd,
# or n/2 if n is even.
  5
 16
  8
  4
  2
  1
```

```
Overwriting data_collatz_5.txt
```

Zkusme takový soubor načíst. Použijeme `readline()` k načtení popisu a komentářů, data načteme metodou `for line in file`.

In [18]:

```
with open('data_collatz_5.txt', 'r', encoding='utf-8') as f:
    # Read the description line
    description = f.readline().strip()
    # Read all the comment lines
    comments = []
    line = f.readline().strip()
    while line.startswith('#'):
        comments.append(line)
        line = f.readline().strip()
    data = []
    data.append(int(line))
    for line in f:
        data.append(int(line))

print("Description:", description)
print("Comments:", comments)
print("Data:", data)
```

```
Description: Collatz 3n+1 sequence, starting from 5.
Comments: ['# The next number in a Collatz sequence is either 3n+1 if n is
 odd,', '# or n/2 if n is even.']
Data: [5, 16, 8, 4, 2, 1]
```

Zamyslete se:

- Je tento kód napsaný čistě?
- Je znovupoužitelný?
- Šlo by jej dekomponovat? Na jaké části?

Čtení souboru "z internetu"

Pokud je soubor dostupný na Internetu, lze jej číst velmi podobně, jako soubor umístěný na lokálním disku. Stačí použít funkci `urllib.request.urlopen()` (a samozřejmě být připojený k Internetu).

Existuje zde ale mírný rozdíl: protože funkce `urlopen` neví, jaký typ souboru načítáte, metody `read`, `readline`, atd. vrací hodnoty typu `bytes`. Abychom z vrácené hodnoty dostali řetězec, musíme byty dekódovat, tj. přiřadit kódovací tabulku (nejlépe UTF-8, je-li to možné).

In [19]:

```
url = r'http://www.gutenberg.org/cache/epub/1661/pg1661.txt'
import urllib.request
with urllib.request.urlopen(url) as text:
    intro = text.read()
intro = intro.decode('utf-8')
print(intro[:300])
```

```
Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doy
le

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBoo
```

Zápis do souboru

Zápis textu do souboru je velmi podobný načítání. Podobně, jako Python neodstranil znaky nového řádku při čtení, při zápisu budete muset sami tyto znaky do řetězců vložit.

In [20]:

```
with open('topics.txt', 'w', encoding='utf-8') as f:
    f.write('Computer Science\n')
    f.write('Programming\n')
    f.write('Clean code\n')
```

In [21]:

```
!type topics.txt
```

```
Computer Science
Programming
Clean code
```

Připojení textu na konec existujícího souboru

Pokud soubor otevřete pro zápis (mód `'w'`), vytvoří se nový soubor, pokud ještě neexistuje; pokud už existuje, přepíše se novým. Když soubor otevřeme pro přidávání (mód `'a'`), zapisované řetězce se připojí na konec souboru.

In [22]:

```
with open('topics.txt', 'a', encoding='utf-8') as f:
    f.write('Software Engineering\n')
```

In [23]:

```
!type topics.txt
```

```
Computer Science
Programming
Clean code
Software Engineering
```

Příklad: Čtení a zápis

Mějme soubor se 2 čísly na každém řádku:

In [24]:

```
%%writefile number_pairs.txt
1 1
10 20
1.3 2.7
```

```
Overwriting number_pairs.txt
```

Vytvořme funkci se dvěma parametry - názvem vstupního a výstupního souboru, která načte páry čísel ze vstupního souboru a zapíše je společně s jejich součtem do výstupního souboru.

In [25]:

```
def sum_number_pairs(infname, outfname):
    """Read data from input file, sum each row, write results to output file.

    (str, str) -> None

    infname: the name of the input file containing a pair of numbers
             separated by whitespace on each line
    outfname: the name of the output file
    """
    with open(infname, 'r', encoding='utf-8') as infile, \
         open(outfname, 'w', encoding='utf-8') as outfile:
        for pair in infile:
            pair = pair.strip()
            operands = pair.split()
            total = float(operands[0]) + float(operands[1])
            new_line = '{} + {} = {}\n'.format(operands[0], operands[1], total)
            outfile.write(new_line)
```

Když funkci zavoláme, vytvoří se požadovaný výstupní soubor se správným obsahem.

In [26]:

```
sum_number_pairs('number_pairs.txt', 'number_pairs_with_totals.txt')
!type number_pairs_with_totals.txt
```

```
1 + 1 = 2.0
10 + 20 = 30.0
1.3 + 2.7 = 4.0
```

Shrnutí

- Práce s cestami k souborům a adresářům pomocí modulu `os.path`.
- Před čtením ze souboru nebo zápisem do souboru je třeba jej nejdřív otevřít! Funkce `open()`.
  - Vždy specifikujte kódování: `open(filename, mode, encoding='utf-8')`.
- Když jste hotovi, musíte soubor zase zavřít! Metoda `f.close()`.
- Příkaz `with` zajistí automatické uzavření souboru!

```
with open('text.txt', 'r', encoding='utf-8') as f:
      contents = f.read()
      # ... and do other things to the opened file
# When you get here, the file is not opened anymore.
```

Nastavení notebooku

Ignorujte jej.

In [27]:

```
from notebook.services.config import ConfigManager
cm = ConfigManager()
cm.update('livereveal', {
              'theme': 'Simple',
              'transition': 'slide',
              'start_slideshow_at': 'selected',
              'width': 1268,
              'height': 768,
              'minScale': 1.0
})
```

Out[27]:

```
{'height': 768,
 'minScale': 1.0,
 'start_slideshow_at': 'selected',
 'theme': 'Simple',
 'transition': 'slide',
 'width': 1268}
```