Práce se soubory.

Práce se soubory. Textový režim. Binární režim. Serializace.

Tomáš Bayer | bayertom@natur.cuni.cz

Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta UK.

Obsah přednášky

1. Práce se soubory
2. Otevření/uzavření souboru
3. Textové soubory
4. Binární soubory
5. Serializace

1. Soubory a operace

Standardní vstup nepostačuje ⇒ zpracování vstupních dat.

Obsah RAM se po vypnutí smaže, výsledek nutno uložit.

Soubor:

Permanentní uložení dat na paměťovém médiu.

Různé atributy a přístupová práva dle operačního systému.

Podpora široké spektra operací:

- *Základní operace:*
  Python: Otevření, čtení, zápis, uzavření.
- *Pokročilé operace:*
  Externí knihovny: Komprese, dekomprese, serializace, rastrová, vektorová data, grafika, multimédia...

Mnoho podporovaných formátů (Python), velké množství externích knihoven:

TXT, HTML, XML, JSON, SHP, JPG, TIFF, ZIP, ...

Soubor reprezentován **streamem** (proudem).

Operace se souborem se **nemusí** podařit:

Neexistuje, nelze přečíst/zapsat, chybná přístupová práva, nulová velikost...

Manipulace se soubory v `try-except` bloku: `IOError` ⇒ Výjimky.

2. Práce se soubory, módy

Práce se soubory v Pythonu:

1. Ověření existence souboru a přístupu k němu.
2. Otevření souboru.
3. Čtení nebo zápis do souboru.
4. Uzavření souboru.

Bod 1 řešen buď výjimkami / “dotazem”.

V případě neúspěšné operace $\Rightarrow$ vznik výjimky. Dva důležité indikátory:

EOL (End of Line), EOF (End of File).

Dva módy (režimy) při práci se soubory:

- *Binary mode*
  Reprezentace znaků (čísla, písmena) v binární/hexadecimální soustavě.
  Přímý zápis/čtení po bytech (8 bitů).
  Nativní pro PC, přímo nečitelné.
- *Text (ASCII) mode*
  Konverze z binárního do textového formátu.
  Čitelné, avšak bez formátování.

Python podporuje čtení i zápis v obou módech.

Vstupem resp. výstupem textový resp. binární soubor.

3. Textový soubor

Představován posloupností řádek.

Každá řádka tvořena posloupností znaků tvořící slova.

Na konci řádky EOL (End Of Line).

Nezachovává formátování.

Čitelný, lze snadno editovat:

```
105   723304.23   1023067.98   208.37
106   723297.17   1023071.04   206.92
107   723301.55   1023068.64   207.76
```

Pro některé znaky nepostačují písmena + číslice + speciálni znaky: *Escape Sequence*

Uvozující znak \ (Backslash).

Escape Sequence	UNICODE	Název	Význam
\n	\u000A	Line Feed	Nová řádka
\r	\u000D	Carriage Return	Návrat na začátek řádku
\t	\u0009	Horizontal Tab	Tabulátor
\\	\u005C	Backslash	Zpětné lomítko
\'	\u002C	Single Quote	Apostrof
\"	\u0022	Double Quote	Uvozovky

Odchylky EOL v implementacích pod různými OS:

```
\n + \r   Windows
\n        GNU/Linux, \r nema specialni vyznam!
\r        MacOS
```

4. Binární soubor

Tvořen posloupností bytů zapsaných v hexadecimální soustavě, hodnoty 0-255.

Používán pro zápis jiných než textových informací.

Není přímo čitelný bez znalosti příslušného formátu.

Nelze prohlížet v běžném editoru (TEXT Mode):

```
’ AÁ˜č řSăĺ64’Áôý Ô X/0Ájĺôő Ž&ÁŮÎ7Kr/Á @ F¶ó}Ö!’ÁZd;žv
/Á<lç{˝!’Áázr Ąv/Á F¶ó}Ö!’ÁHáz’’¬v/ÁŃ"ŰąÄ!’Áázr Ąv/Á<lç
{˝!’Á%U¸v/Á#Ű ů>Ď!’ÁZd;žv/ÁF¶ó}Ö!’ÁHáz’’¬v/Á @ °rh‘Ę!’
```

Použití prohlížeče s podporou HEX Mode:

```
27 20 41 c3 81 7e c4 8d | 20 c5 99 53 c4 83 c4 ba
36 34 27 c3 81 c3 b4 c3 | bd c3 94 58 2f 30 c3 81
6a c4 bd c3 b4 c3 93 c5 | bd 26 c3 81 c5 ae c3 8e
```

Struktura binárního formátu:

Souvislý blok dat, není dělen na řádky, slova.

Často sdružovány po 8 (1 byte = 8 bitů).

Záhlaví představováno hlavičkou a identifikátorem formátu (nepovinné).

Následuje vlastní datová oblast.

Specifické pro různé formáty.

Používán pro kompilovaný kód (exe, bin).

Ukládání grafiky, zvuků, dokumentů, vektorových data.

5. Otevření souboru

První krok, který nutno učinit před čtením/zápisem.

Soubor lze otevřít v různých módech.

Nutno specifikovat:

- zda pracujeme s binárním/textovým souborem,
- jakou operaci chceme provádět $\Rightarrow$ mód.

Označení	Mód	Popis
r	Read	Otevření souboru pro čtení (read only, default).
w	Write	Otevření souboru pro zápis, původní obsah přepsán.
a	Append	Otevření souboru pro zápis, přidání nových dat na konec
r+	Read/Write	Kombinace čtení i zápisu.
t	Text	Otevření v textovém režimu (default)
b	Binary	Otevření v binárním režimu.
x	Exclusive	Vytvoření souboru + otevření pro zápis.

Módy lze je kombinovat: rt+ nebo wb+.

Příkaz `open()`, vrací file object, resp. handler.

```
file_object = open("file_name", "mode")
```

Otevření souboru v aktuální složce:

```
f=open('test.txt')              #Default, open in rt mode
```

Otevření souboru s uvedením plné cesty:

```
f=open('E:/Data/test.txt')      #Do not use backslash, open in rt mode
```

Otevření souboru se specifikacé módu:

```
f=open('img.jpg', 'rb+')       #Open in read and write mode, binary file
```

6. Uzavření souboru

Po skončení operací se souborem nutno provést jeho uzavření.

OS při práci se souborem:

- používá deskriptor ukazující na otevřený soubor,
- alokuje systémové prostředky.

Neuzavření souboru:

Deskriptor násilně ukončen OS až po skončení procesu.

Pokud proces není ukončen $\Rightarrow$ nestandardní chování + únik systémových prostředků.

Možné problémy:

- Do souboru nejde zapisovat (jeví se jako read only).
- Zapsané změny se nemusejí uložit.

Uzavření souboru, metoda `close()`

```
f.close()    #Close the file
```

Nutno vždy provést, i když dojde k výjimce, jak to zaručit?

7*. Využití výjimek

Bezpečnější způsob práce se soubory představuje využití výjimek.
Operace se souborem ošetřeny metodou chráněných bloků.

Výjimky při práci se soubory: potomci `IOError`.

Korektní uzavření souboru: použití `close()` ve větvi `finally`.

```
import os
f = None
try:
   f = open('test.txt', 'rt+')   #Open file in read/write mode
except IOError as e:
   print (e)                     #Print exception
finally:
   if f:
      f.close()                  #Close file
```

Uzavření souboru provedeno ve `finally` bloku.

Poměrně komplikované, Python používá efektivnější nástroje.

8. Další vlastnosti proudu

Existuje několik nástrojů které umožňují zjistit stav streamu:

```
f = open('E:/Tomas/Text/test.txt', 'r')
```

Název souboru/proudu:

```
>>> f.name
'E:/Tomas/Text/test.txt'
```

Zjištění stavu proudu (otevřený/uzavřený):

```
>>> f.closed
False
```

Dotaz na mód souboru:

```
>>> f.mode
'r'
```

Dotaz, zda lze ze souboru číst:

```
>>> f.readable()
True
```

Dotaz, zda lze do souboru zapisovat:

```
>>> f.writable()
False
```

9. Efektivnější práce se soubory

Přístup `try-except-finally` není příliš user-friendly.

Využití příkazu `with.` + `try-except.`

Univerzální nástroj, lze použít pro čtení resp. zápis ze resp. do souboru.
Bezpečný, provádí automatické uzavření souboru i v případě výjimky.

Doporučován jako výchozí nástroj pro práci se soubory.
Výhodou jednodušší syntaxe:

```
with open(file_name) as file_object:
    #perform some operation
```

Kombinace s výjimkami:

```
try:
   with open('test.txt', 'rt+') as f:
      data = f.read()    #Read data from file
except IOError as e:
   e.print()
```

Lze spojit i s dotazem, zda soubor existuje:

```
import os.path
if os.path.exists('test.txt'):
   with open('test.txt', 'rt+') as f:
      data = f.read()                    #Read data from file
```

10. Čtení z textového souboru I.

Předpoklad: soubor otevřen v módu umožňujícím čtení.

Provádí se, dokud nedosáhneme EOF, poté nutno soubor uzavřít.

Tři varianty čtení dat ze souboru:

- načtení souboru nebo jeho části,
- načtení 1 řádky souboru,
- načtení celého souboru po řádkách.

Načtení souboru nebo jeho části.

Výsledek načten do proměnné typu `String`.

Lze specifikovat, kolik bajtů se má načíst (1 byte = 1 znak).

Použita funkce `read()`, dvě varianty.

```
with open('test.txt', 'rt+') as f:          #Open file
   data1 = f.read()                         #Read entire file
   data2 = f.read(5 )                       #Read 5 bytes of file
   data3 = f.read(10)                       #Read next 10 bytes of file
```

Vytištění obsahu souboru:

```
'We are learning Python,\nthe best language.'  #EOL \n not removed
'We ar'
'e learning'
```

Procházení načteného souboru po znacích

```
for line in data:                           #Browse file by chars
   print(line)                              #Do something
```

Odstranění white spaces:

Odstranění EOL, načtení do seznamu Stringů

```
>>>f.read().splitlines()
['We are learning Python,', 'the best language.']
```

11. Čtení z textového souboru II.

Čtení souboru po řádkách.

Načte s každým zavoláním jednu řádku souboru.

Následně skok na další řádku.

Výsledek uložen do řetězce typu String.

```
>>>line1 = f.readline()
>>>line2 = f.readline()
'Monday\n'                                  #EOL \n not removed
'Tuesday\n'                                 #EOL \n not removed
```

Ukázka načtení celého souboru po řádcích:

```
with open('test.txt', 'rt+') as f:
   l = f.readline()                         #Read first line
   while l:                                 #While not EOL
      l = f.readline()                      #Read next line
```

Načtení celého souboru po řádkách.

Soubor načten jednorázově, po řádkách.

Výsledek uložen jako seznam Stringů.

```
>>>lines = file.readlines()
>>> print(lines)
['Monday\n', 'Tuesday\n', 'Wednesday']  #EOL \n are not removed
```

12. Pohyb v otevřeném streamu

V otevřeném souboru se můžeme pohybovat.

Lze se posunout na začátek, konec či konkrétní pozici.

Zjištění aktuální pozice:

Aktuální pozice ve streamu udávána v bytech, nabývá hodnot <0, n-1>.

Po načtení celého souboru ukazuje počet znaků v souboru.

```
>>> f.tell()   #Dotaz na aktualni pozici v souboru
40
```

Posun na konkrétní pozici:

Specifikace pozice v souboru, na kterou se chceme posunout.

Pohyb v souboru oběma směry o *k* znaků.

```
seek(o kolik, odkud)
```

Tři módy nastavení referenčního bodu:

Mód	Vyjádření	Popis	Mód
0	SEEK_SET	Referenční bod na začátek souboru.	t,b
1	SEEK_CUR	Referenční bod na aktuální pozici.	b
2	SEEK_END	Referenční bod na konec souboru.	b

```
>>> f.seek(0)          #Skok na počátek souboru
0
>>> f.seek(10, 0)      #Skok na 10. znak
>>> f.read()
'rning Python,\nthe best language.
```

13*. Ukázka: skok na k-tou řádku v souboru

Zapamatujeme si pozici začátku každého řádku v souboru.

Hodnoty ukládáme do seznamu.

```
with open('test.txt', 'rt+') as f:
    begins = []                          #Create empty list of begins
    b = 0                                #Initialize to the first positio
    for l in f:                          #Read line by line
        begins.append(b)                 #Add first begin
        b +=  len(l)                     #Increment, add line length
```

Základní myšlenka:

Indexujeme-li od 0, délka řádku *m* znaků.

Znak `f[0]`: počátek 1. řádku.

Znak `f[m-1]`: konec 1. řádku.

Znal `f[m]`: počátek 2. řádku.

K předcházejícímu konci řádku přičítáme délku řádku.

Pak:

```
f.seek(begins[k])                        #Jump to the k-th line
```

14. Zápis do souboru

Předpoklad: soubor otevřen v módu umožňujícím zápis.

Tři základní režimy zápisu:

- *Režim `w` (Write Mode):*
  Původní obsah souboru nahrazen.
- *Režim `a` (Append Mode)*
  Připisujeme na konec souboru.
  Zachován původní obsah.
- *Režim `x` (Exclusive Mode)*
  Vytvoření nového souboru + otevření pro zápis.
  Pokud soubor již existuje, výjimka.

Po skončení zápisu soubor nutno uzavřít, využití funkce `with()`.

Zápis do souboru po řádkách.

Funkce `write()`, zapisuje řetězec do otevřeného proudu.

```
write(string)
```

Zápis do souboru, smazání původního obsahu:

```
with open("test.txt",'w',) as f:
   f.write("Monday\n")''          #Delete all + create first line
   f.write("Tuesday\n")''         #Create second line
```

Přidání na konec souboru, ponechání původního obsahu:

```
with open("test.txt",'a',) as f:
   f.write("Wednesday\n")''       #Append new line to existing content
```

15*. Čtení z binárního souboru

Python podporuje práci také s binárními soubory.

Posloupnost kroků stejná jako u textových souborů.

Při otevření nutné použít binární režim ’b’ + mód ’r’.

```
with open("picture.jpg",'rb',) as f:          #Set binary mod, read
```

Binární soubory načítáme po bytech.

Použita metoda `read()`.

```
with open("picture.jpg",'rb',) as f:
   data = f.read()                            #Read entire file
   print(data)
b'\x00\x00\'\na6\xdb\x86\x89\x9f?\x85\xb9      #Byte array
```

Výsledkem operace typ byte array (pole bajtů).

```
   for b in data:                             #Proces bytes one by one
      print(b)                                #Print in readable form
160 161 162 163                               #Numerical values of bytes
```

Bez znalosti definice načítaného formátu není jasné, co položky znamenají.

Rastr interpretován jako proud bajtů.

Pro další práci s ním nutné knihovny (zobrazení rastru, konverze, atd).

16*. Zápis do binárního souboru

Soubor nutno otevřít v binárním režimu + některém módu zápisu.

```
with open("data.shp",'wb',) as f:    #Set binary mod, write to file
```

Zápis dat do souboru probíhá po bytech.

Hodnoty bitů leží v intervalu <0, 255>.

Vyjádřeny jako hexadecimální řetězce: znaky 0-9, A-F.

Typ bytearray:

V Pythonu specializovaný datový typ pro práci s byty: `byttearray`.

Uvozen znakem `'b'`, prefix každého bajtu `\x`.

```
>>>b = [160, 161, 162, 163]
>>>ba = bytearray(b)                    #Create byte array
>>>print(ba)                            #Print byte array
bytearray(b'\xa0\xa1\xa2\xa3')
```

Zápis do souboru metodou `write()`.

Vytvoření nového souboru, mód `x`, zápis pole bytů.

```
with open("data.shp",'xb',) as f:    #Create new file
   f.write(ba)
```

17. Serializace a deserializace

Serializace:

Převod datové struktury uložené v paměti na posloupnost bytů.

Tuto lze následně uložit na paměťové médium.

Deserializace:

Opačný postup, zpětná rekonstrukce posloupnosti bytů na datovou strukturu.

V praxi mají obě operace velký význam.

Možnost uložení resp. načtení aktuálního obsahu datových struktur do resp. z souboru.

Výsledné soubory textové či binární.

Data mohou být použita při dalším spuštění.

Využití pro ladění programů, ukládání pracovních dat,...

Nevýhoda: obtížná editace obsahu (binární soubory).

Podpora serializace/deserializace pro:

- základní datové typy,
- dynamické datové struktury,
- funkce,
- třídy,

Modul pickle:

4 verze protokolů, vzájemná nekompatibilita.

18. Serializace a deserializace pickle, ukázka

Serializace objektu:

Uložení objektu jako posloupnost bytů do souboru

```
dump(data_object, file)
```

Soubor nutno otevřít ve správném módu.

```
import pickle
d = {'name' : 'John', 'surname': 'Smith',
     'job': 'cartographer', 'age' : '25'}
with open('test.pickle', 'wb') as f:
    pickle.dump(d, f)
```

Ukázka souboru test.pickle:

```
€}q (X   nameqX   JohnqX   ageqX   25qX   jobqX
cartographerqX   surnameqX   Smithqu.
```

Deserializace objektu:

Načtení pole bajtů ze souboru

```
load(data_object, file)
```

Opět nutno otevřít ve správném módu.

```
with open('test.pickle', 'rb') as f:
   d2 = pickle.load(f)
```

S daty můžeme pracovat jako s původními.

19*. Serializace s využitím JSON

Formát JSON (Java Script Object Notation):

Výměnný formát pro přenos dat, podmnožina JavaScript formátu.

Textový formát, snadná čitelnost, přehlednost.

Výhodou textová struktura, snadno lze editovat.

Podpora JSON v Pythonu:

V Pythonu implementace JSON založena na Nested Dictionary.

Načtený JSON parsován na Dictionary.

Univerzální formát pro serializaci a deserializaci objektů v Pythonu.

Na rozdíl od Pickle jsou uložená data snadno čitelná.

Existuje i jeho prostorová varianta: GeoJSON.

Jedna z možností vstupu prostorových dat do Pythonu.

Serializace/deserializace JSON:

Podpora pro stejné datové typy jako u Pickle.

Uložení a načtení, metody `dump()` a `load()`.

20*. Serializace a deserializace JSON, ukázka

Serializace objektu:

Uložení objektu jako do souboru JSON.

```
dump(data_object, file)
```

Soubor nutno otevřít ve správném módu, import modulu json.

```
import json
d = {'name' : 'John', 'surname': 'Smith',
     'job': 'cartographer', 'age' : '25'}
with open('test.json', 'w') as f:
    json.dump(d, f)
```

Ukázka souboru test.json:

```
{"age": "25", "surname": "Smith",
 "job": "cartographer", "name": "John"}
```

Deserializace objektu:

Načtení JSON souboru do Dictionary.

```
load(data_object, file)
```

Ukázka deserializace objektu:

```
with open('test.json', 'rb') as f:
     d2 = json.load(f)
```

21*. GeoJSON

Otevřený formát pro výměnu prostorových dat, vznikl v roce 2008.

Interní struktura vychází z formátu JSON.
Textový, snadná čitelnost, univerzalita.
Používán často zejména pro webové mapování.

Podpora prostorových souřadnic a různých geometrických entit dle OGIS:
Point, Line String, Polygon, a jejich multivarianty (celkem 7).
Založen na sytému WGS-84, jednotky: stupně.

Nevýhodou neefektivita popisu: dlouhé identifikátory, white-spaces, atd.
Prostorově náročný, vhodný pro menší data.
Ukládá pouze geometrii, nikoliv atributy.

Existuje i varianta ukládající topologii TopoJSON.

Detailní specifikace formátu GeoJSON:

http://geojson.org/

22*. Podpora GeoJSON

V Pythonu 3 přímá podpora formátu GeoJSON.

Instalace modulu `geojson`

Ukázka struktury formátu GeoJSON:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [
            14.425022006034853,
            50.06900389174371
          ],
          [
            14.423967897891997,
            50.06875597855528
          ]
        ]
      }
    }
  ]
}
```

23*. Serializace a deserializace GeoJSON, ukázka

Deserializace objektu:

Načtení GeoJSON ze souboru do Dictionary

```
load(data_object, file)
```

Ukázka deserializace objektu:

```
import geojson
with open('test.geojson') as f:
    d = geojson.load(f)
```

Entity uloženy ve slovníku, klíč `'features'`.

Každá `'feature'` tvořena slovníkem, klíč `'geometry'`.

Lze procházet cyklem for

```
for f in d['features']:
    print f['geometry']['type']
    print f['geometry']['coordinates']
```

Serializace objektu:

Uložení objektu do souboru GeoJSON.

Ukázka deserializace objektu:

```
dump(data_object, file)
with open('test2.geojson', 'w') as f:
    geojson.dump(d, f)
```

24*. Znázornění GeoJSON