IB111 Úvod do programování skrze Python
Přednáška 10

Práce s textem a daty, regulární výrazy

Nikola Beneš

20. listopad 2017

Motivace

Chceme zpracovávat data

- jakou mají podobu?
  - text, čísla, strukturovaná data, ...
- jak jsou reprezentována?
  - tabulka (xls, ods, csv, ...)
  - dokument (doc, odt, md, html, ...)
  - čistý text
  - ...

Základní práce s řetězci (připomenutí)

```
text.strip()          # odstraní bílé znaky ze začátku a konce
text.lstrip()         # totéž, jen ze začátku
text.rstrip()         # totéž, jen z konce
text.strip("ABC")     # odstraní zadané znaky

text.split()          # vytvoří seznam jednotlivých slov
text.split(", ")      # dělí podle zadaného řetězce
text.split(":", maxsplit=n)  # jen daný počet rozdělení

text.replace(x, y)    # nahradí všechny výskyty x -> y

text.find(s)          # index výskytu s nebo -1
s in text             # True/False, jestli text obsahuje s
```

https://docs.python.org/3/library/stdtypes.html#textseq

Speciální znaky

- uvádějí se znakem zpětného lomítka \

```
"\n"      # konec řádku
"\""      # jedna uvozovka
'\''      # jeden apostrof
"\t"      # tabulátor
"\\"      # jedno zpětné lomítko
```

https://docs.python.org/3/reference/lexical_analysis.html#strings

- co když chceme tuto vlastnost zpětného lomítka „vypnout“?
  - *raw string*: přidání `r` nebo `R` před řetězec
  - hodí se u regulárních výrazů (uvidíme za chvíli)

```
r"dvě zpětná lomítka: \\"
r'žádné speciální znaky \n \t'
```

Práce se soubory (připomenutí)

```
f = open("myfile.txt", "r")   # otevření pro čtení
f = open("myfile.txt", "w")   # otevření pro zápis
f.close()                     # uzavření souboru
f.readline()                  # další řádek ze souboru
f.readlines()                 # seznam zbývajících řádků

with open(...) as f:
    # soubor se sám zavře na konci bloku with

for line in f:
    # postupné procházení řádků souboru
```

Příklad: Četnost jmen

- zajímají nás nejčastější jména narozených ve vybraném roce
- zdroj dat: `http://www.mvcr.cz`
  - formát XLS, převedeno do CSV (uložit jako…)
  - první řádek je hlavička tabulky
  - na dalších řádcích je vždy jméno a četnosti podle let

- jak řešit?
  - zjistíme si, který sloupec nás zajímá
  - vyčteme z něj četnost
  - seřadíme jména podle četnosti
  - vypíšeme deset nejčastějších

[ukázka]

Regulární výrazy – motivace

- hledání nebo zpracování nějak strukturovaných dat
  - e-mailové adresy
  - telefonní čísla
  - webové odkazy
  - náhrada „Jméno Příjmení“ za „Příjmení, Jméno“
  - změna formátu data (20. 11. 2017 $\rightarrow$ 2017-11-20)
  - odstranění znaků určitého druhu
  - data v netradičním formátu, která vyžadují předzpracování
- dá se řešit s dosavadními nástroji
  - funkce s řetězci
  - procházení řetězců po znacích
- ale může to být obtížně napsatelné a nešikovné

Regulární výrazy – motivace

```
Arnold Schwarzenegger se narodil 30. 7. 1947.
Madonna se narodila 16. srpna 1958.
Ludwig van Beethoven se narodil 16. 12. 1770.
```

- chceme z těchto dat vyčíst, kdy má kdo narozeniny
- víme, že data mají následující *strukturu*:
  - „*jméno* se narodil(a) *den měsíc rok*.“
  - *jméno* je libovolná (neprázdná) posloupnost znaků
  - *den* je číslo ukončené tečkou
  - *měsíc* je buď číslo ukončené tečkou nebo jedno slovo
  - *rok* je číslo

```
.+ se narodila? [0-9]+\. ([0-9]+\.|\w+) [0-9]+\.
```

- toto je tzv. *regulární výraz*

Regulární výrazy

Kde se s nimi setkáme?

- programovací jazyky
  - zejména tzv. skriptovací jazyky
  - ale dnes už ve všech moderních jazycích
- (chytřejší) textové editory
- (chytřejší souborové manažery)
- nástroje příkazové řádky (`sed`, `grep`, ...)

- teorie: formální jazyky, konečné automaty

Regulární výrazy

- způsob, jak popisovat *vzory* v textu
- obecně používaný nástroj
- základní syntax stejná nebo velmi podobná ve většině jazyků/prostředí
- my si ukážeme
  - základní syntax
  - použití regulárních výrazů v Pythonu
- nebudeme rozebírat všechny technické detaily
  - (podrobněji viz dokumentace)

Příklad: Emailová adresa

- chceme vypsat všechny řádky ze souboru, které obsahují emailovou adresu

```
import re

with open("test.txt") as f:
    for line in f:
        if re.search(r'[a-z]+@[a-z]+\.cz', line):
            print(line)
```

- tento příklad má zatím značné nedostatky

Regulární výrazy

- základní znak popisuje sám sebe
  - př. „cz“ v předchozím příkladu
- speciální znaky: `. ^ $ * + ? { } [ ] ( ) \ |`
  - umožňují konstrukci složitějších výrazů
  - chceme-li popsat přesně jen speciální znak, předřadíme mu zpětné lomítko
  - př. regulární výraz \$ hledá v textu znak dolaru
- zpětné lomítko dále vytváří speciální znaky ze základních

Regulární výrazy

Hranaté závorky `[]`

- popisují rozsah možných znaků
- `[abc]` – jeden ze znaků `a`, `b`, `c`
- `[a-z]` – výběr z intervalu (malé písmeno anglické abecedy)
- `[0-9]` – jedna číslice
- `^` na začátku výběru – negace
  - `[^abc]` – cokoli kromě `a`, `b`, `c`

Regulární výrazy

Často používané skupiny znaků

- `\d` – číslice
- `\D` – cokoli kromě číslic
- `\w` – „znaky ve slovech“ (alfanumerické znaky)
- `\W` – cokoli kromě alfanumerických znaků
- `\s` – bílé znaky (mezery, tabulátory, ...)
- `\S` – cokoli kromě bílých znaků

Regulární výrazy

Speciální symboly

- `.` – libovolný jeden znak
- `^` – začátek řetězce
- `$` – konec řetězce

Skupiny

- `(`, `)` – závorkování, vytvoření skupiny (více za chvíli)
- `|` – alternativa (*nebo*)

Opakování

- `*` – nula nebo více opakování předešlé části výrazu
- `+` – jedno nebo více opakování předešlé části výrazu
- `?` – nula nebo jeden výskyt předešlé části výrazu
- `{`m`,`n`}` – m až n opakování předešlé části výrazu (lze i `{`n`}`)

Poznámka: `*` a `+` jsou „hladové“, pro co nejmenší počet opakování `*?` a `+?`

Regulární výrazy

Jaký je význam následujících výrazů?

- `kocka|pes`
- `^[Pp]rase$`
- `\d[A-Z]\d \d{4}`
- `^\s*Nadpis`
- `^a.+a$`
- `\d{3}\s?\d{3}\s?\d{3}`
- `[a-z]+@[a-z]+\.cz`
- `To:.*(fi|ktp)(-int)?@fi\.muni\.cz`

Regulární výrazy

Backreference

- odkaz na předchozí skupinu
- (z teoretického hlediska tohle není regulární)

- `^(...).*\1$`
- `(.+)\1`

Regulární výrazy v Pythonu

- knihovna `re` (`import re`)
- `re.match` hledá výskyt na začátku řetězce
- `re.search` hledá výskyt kdekoli v řetězci
- `re.findall` hledá všechny výskyty v řetězci
- `re.sub` nahradí výskyty regulárního výrazu zadaným řetězcem
- (`re.compile` – předkompiluje regulární výraz pro větší efektivitu)

- využijeme „raw string“ – `r'regulární výraz'`, aby nedocházelo k interpretaci speciálních znaků (zejména \)

Regulární výrazy v Pythonu

- `match`/`search` vrací speciální objekt
- přístup k jednotlivým skupinám pomocí metody `group`

```
m = re.match(r'(\w+) (\w+)', "Isaac Newton, fyzik")
print(m.group(0))
print(m.group(1))
print(m.group(2))
```

Regulární výrazy v Pythonu

- v substituci můžeme používat zpětné reference \1 apod.

```
print(re.sub(r'(\w+) (\w+)', r'\2, \1', "Isaac Newton"))
```

Příklad: Lepší hledání emailů v textu

- chceme vypsat jen nalezené emaily
- chceme hledat emaily složitějších tvarů

Příklad: Počítání bodů v bloku

- máme zadaný text, který je obsahem ISovského bloku
- chceme sečíst všechny body v bloku
- body jsou ve tvaru **číslo*

- složitější varianta:
  - máme exportovaný blok ve tvaru `učo:jméno:obsah bloku`
  - chceme sečíst body v bloku pro každého studenta

Příklad: hromadné přejmenování souborů

- máme soubory se jmény `něco1.txt` až `něco117.txt`
- chceme je přejmenovat tak, aby se pěkně řadily podle čísla
  - použijeme tvar `něco0001.txt`
- jak přejmenovat všechny?