VŠB TECHNICKÁ UNIVERZITA OSTRAVA | HORNICKO GEOLOGICKÁ FAKULTA | KATEDRA GEOINFORMATIKY

Programování v GIS 1
5 – práce se soubory

Michal Kačmařík

A924, tel.: 5512

e-mail: michal.kacmarik@vsb.cz

https://www.hgf.vsb.cz/548/cs/

Úvod

- Programy a skripty pracují s daty
- Data mohou být uložena v samostatných souborech či v komplikovanějších strukturách (např. databáze)
- Cílem přednášky je představit problematiku načítání a ukládání dat z/do binárních/textových souborů pomocí jazyka Python ve verzi 3

Soubor a jeho přípona

- soubor je fyzicky uložen jako sada čísel (bytů)
- význam jim přisoudíme až interpretací souboru – např. určením formátu dat a otevřením souboru ve vhodném programu
- pro zajištění správné interpretace jsou názvy souborů standardně opatřeny příponou (.html, .docx, .jpeg, .txt, .exe)
- sama přípona však nijak neovlivňuje obsah souboru – změnou přípony nijak nezměníme data uložená v souboru

- *Vytvořte textový soubor text.txt, který bude obsahovat několik řádků libovolného textu. Pak jej 1) přejmenujte na text.text a zkuste jej otevřít v Poznámkovém bloku; 2) přejmenujte na text.jpeg a zkuste jej otevřít v prohlížeči obrázků. Co se stane?*

Cesta k souboru

- Při práci se souborem musíme vždy definovat cestu k němu
- **Absolutní cesta**
  - Začíná označením disku (názvem kořenového adresáře)
  - např. D:\data\programovani\soubor.txt (nebo /home/data/programovani/soubor.txt na OS Linux)

- Poznámka: OS Windows používá zpětná lomítka („\“) při určování absolutní cesty k souboru. Programy však běžně umí použít i absolutní cestu zadanou dopřednými lomítky („/“). Pro určování relativní cesty jsou využívána dopředná lomítka
- OS Linux vždy využívá dopředná lomítka při definici cesty

Cesta k souboru

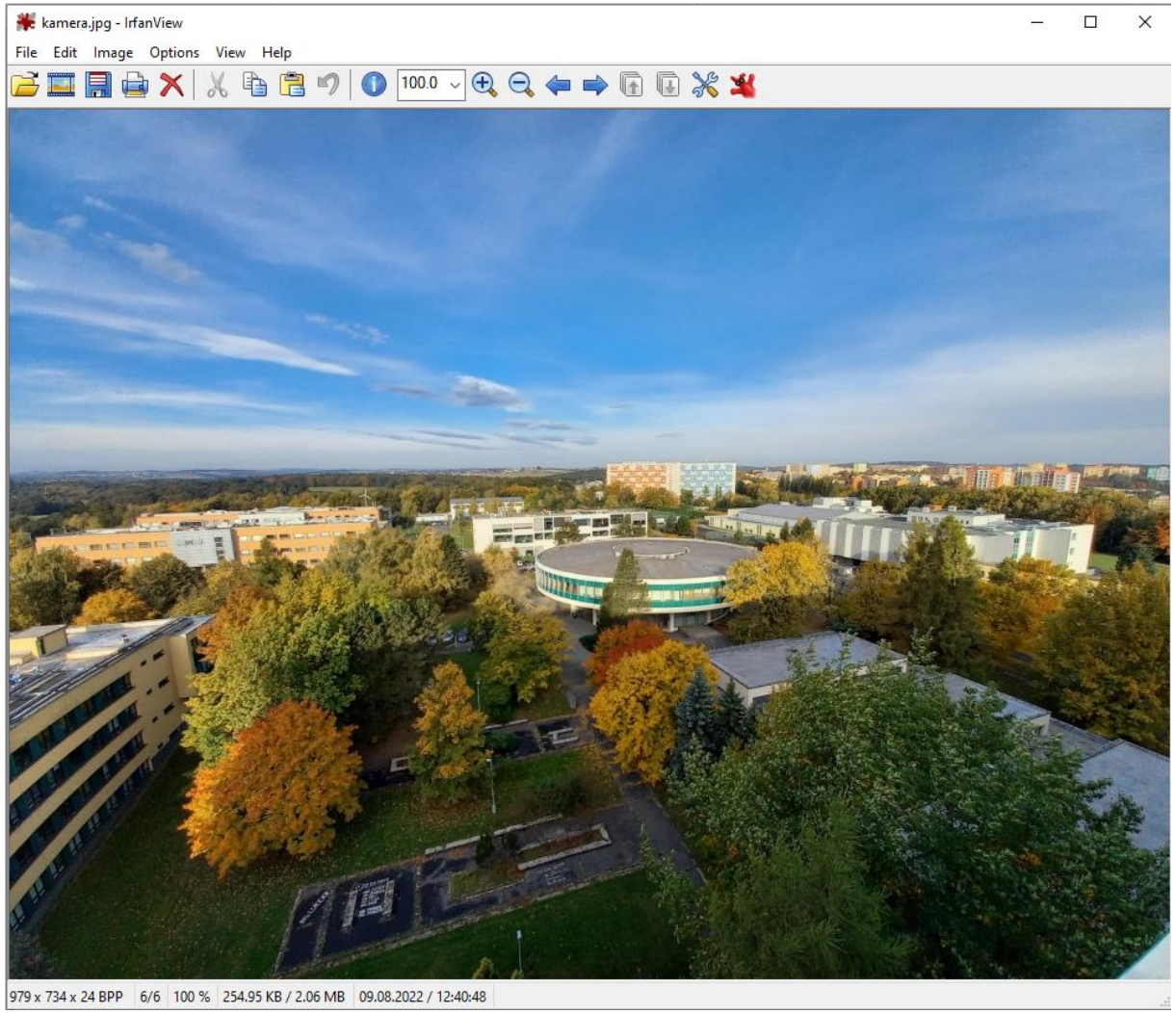
- **Relativní cesta**
  - je vyjádřena **relativně** vůči aktuálnímu pracovnímu adresáři (standardně adresář, ve kterém je uložen náš skript či program, či který si nadefinujeme pro práci)
  - např. máme skript uložen v cestě D:/data/programovani, cestu k souboru zadáme jednoduše pomocí soubor.txt
  - formy zápisu relativní cesty
    - soubor.txt – soubor se nachází přímo v pracovním adresáři
    - ../soubor.txt – soubor se nachází v adresáři o úroveň výše než je pracovní adresář, zápis ../ se odkazuje na rodičovský adresář
- Poznámka: některé jazyky podporují zápis ./, který se odkazuje na současný adresář (relativní cesta ./soubor.txt je tedy identická jako při zápise soubor.txt)
- Kde by byl vůči pracovnímu adresáři uložen soubor s relativní cestou ../../test/soubor.txt?

Textový versus binární soubor

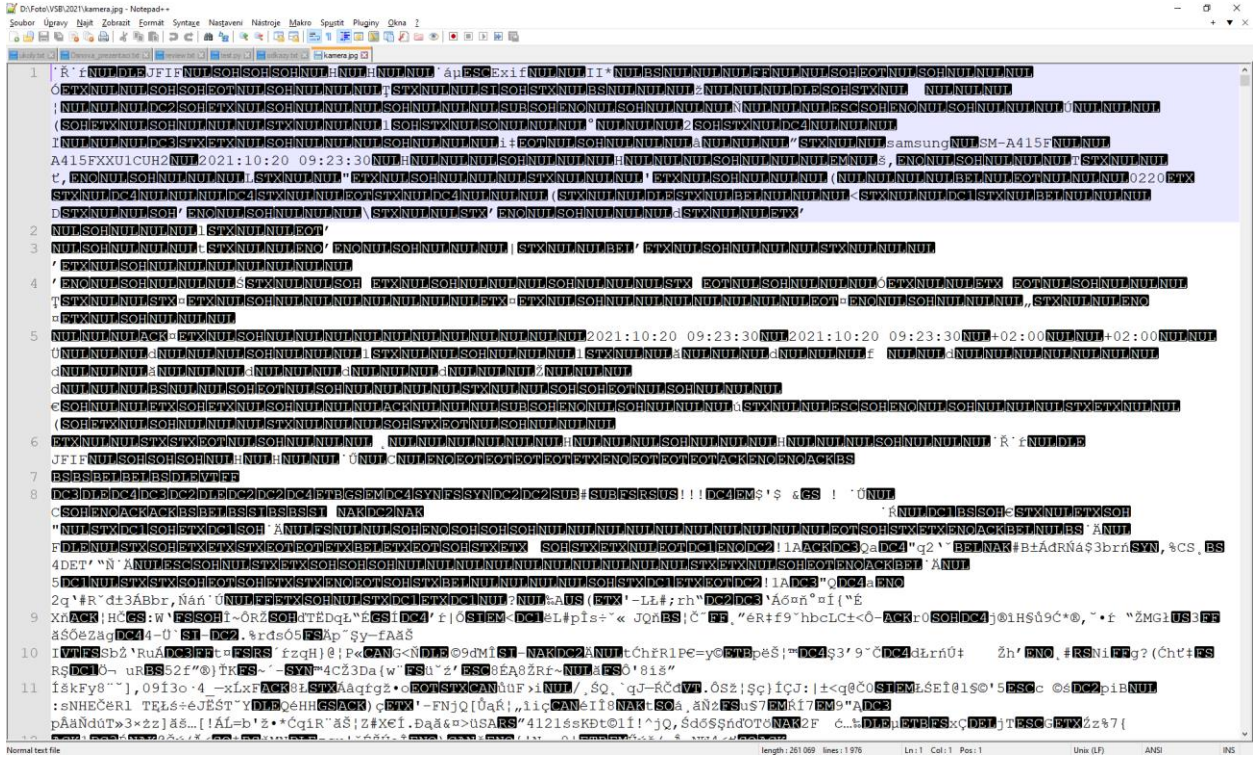
- **Textový soubor**
  - Data jsou ukládána po řádcích s využitím znaků vybrané znakové sady (např. ASCII)
  - Soubor je čitelný pro stroj i člověka a pro člověka je i snadno editovatelný
- **Binární soubor**
  - Data jsou ukládána přímo v binárním zápise (tedy ve stejné podobě, jako jsou uložena v paměti počítače při načtení souboru)
  - Soubor je čitelný jen pro stroj
  - Binární zápis je kompaktnější než textový a rychleji se čte/zapisuje

Textový versus binární soubor

Binární soubor formátu .jpg otevřený v prohlížeči obrázků



Stejný .jpg soubor otevřený v textovém editoru



Python – open()

Základním příkazem pro práci se soubory je

open()

Má tři základní parametry:

- cesta k souboru včetně jeho názvu
- specifikace módu, v jakém má být soubor otevřen
- *encoding* = umožňuje specifikovat, jaká znaková sada se má použít při práci s textem. Pokud není parametr definován, použije se výchozí (tzv. defaultní) znaková sada (typicky UTF-8)

https://docs.python.org/3/library/functions.html#open

open() – módy otevření

Mód otevření	Popis
"r"	read – soubor se otevře pro čtení, příkaz vrátí chybu pokud soubor neexistuje
"w"	write – soubor se otevře pro zápis, pokud soubor již existuje, je přepsán, pokud soubor neexistuje, je vytvořen prázdný
"a"	append – soubor se otevře pro přidání dalšího obsahu (zápis je realizován na konec již existujícího souboru), pokud soubor neexistuje, je vytvořen. Hodí se například pro logy.
"x"	create – soubor je pouze vytvořen, pokud soubor již existuje, příkaz vrátí chybu
"r+"	soubor se otevře pro čtení i zápis, příkaz vrátí chybu pokud soubor neexistuje
"w+"	soubor se otevře pro čtení i zápis, chování v případě (ne)existence souboru je stejné jako u obyčejného "w"
"a+"	append – soubor se otevře pro čtení pro přidání dalšího obsahu (zápis je realizován na konec již existujícího souboru), pokud soubor neexistuje, je vytvořen
"t"	text – textový režim, s obsahem je nakládáno jako s textem. Defaultní režim.
"b"	binary – binární režim, s obsahem je nakládáno jako s binárními daty. Měl by být použit pro všechny binární (= netextové) soubory

Python – open()

Syntaxe příkazu

```
f = open("test.txt", "r") #do promenne f se otevre soubor test.txt v modu pro cteni ze souboru, textovy rezim

f = open("test.txt", "w") #do promenne f se otevre soubor test.txt v modu pro zapis do souboru, textovy rezim

f = open("image.jpg", "rb") #do promenne f se otevre soubor test.txt v modu pro cteni ze souboru, binarni rezim
```

Python – close()

Po ukončení práce se souborem je potřeba jej zavřít příkazem

close()

V některých případech nejsou změny v souboru realizovány dokud nedojde k jeho uzavření tímto příkazem (z důvodu tzv. bufferování = počítač nezapisuje neustále, aby tím nezatěžoval svůj výkon; shromažďuje data pro zápis a pak je zapíše najednou)

```
f = open("test.txt", "r")
print(f.readline())
f.close()
```

Python – with

Automatického uzavření souboru je možné docílit použitím příkazu

with

v rámci příkazu open()

Například:

```
with open("test.txt", "r") as source_file:
    radky = source_file.readlines()
```

Čtení ze souboru

Několik možností:

- read() - načte celý soubor najednou jako jeden textový řetězec.
- read(5) - číslo v závorce definuje počet znaků, které mají být načteny
- readline() - načte jeden řádek souboru jako jeden textový řetězec. Lze volat opakovaně pro postupné čtení řádků
- readlines() - načte celý soubor najednou jako seznam, jednotlivé prvky seznamu jsou tvořeny jednotlivými řádky souboru

Čtení ze souboru

```
f = open("test.txt", "r")

soubor_jako_retezec = f.read()

prvni_tri_znaky = f.read(3)

prvni_radek = f.readline()

druhy_radek = f.readline()

soubor_jako_seznam_radku = f.readlines()

prvnich_pet_radku_se_ulozi_do_seznamu = f.readlines()[0:5] #prace s indexy

f.close()
```

Pozor, výše uvedené příkazy na sobě nejsou vzájemně nezávislé. Pokud použiji například příkaz f.readline() a následně příkaz f.readlines(), tak ve výstupu z druhého uvedeného příkazu již nebude první řádek souboru!

Čtení ze souboru

Postupné načítání jednotlivých řádků souboru s využitím cyklu:

```
f = open("test.txt", "r")
for radek in f:
    print(radek)
f.close()
```

```
f = open("test.txt", "r")
radky = f.readlines()
for radek in radky:
    print(radek)
f.close()
```

Zápis do souboru

Dvě možnosti:

- write() - zapíše zadaný textový řetězec
- writelines() - zapíše najednou celý seznam prvků

Odsazení konce řádku (jen v textovém režimu):

- pomocí znaku **"\n"**
- poznámka: OS Linux a MacOS používají pro zapsání konce řádku ASCII znak LF (line feed). OS Windows používá dvojici ASCII znaků CR (carriage return) a LF, v tomto pořadí. Při zapsání konce řádku pomocí "\n" dojde na Windows ke konverzi na dvojici znaků " "\r\n", v Linuxu a MacOS k zapsání jen "\n".

Zápis do souboru

```
f = open("test.txt", "w")
f.write("Zkusebni radek textu\n")
f.writelines(soubor_jako_seznam_radku)
f.close()

f = open("log.txt", "a")
f.write("Jeden radek textu navic na konci souboru\n")
f.close()
```

Smazání souboru

- pro smazání existujícího souboru je možné využít funkce unlink() z knihovny pathlib (jazyk Python má celou řadu vestavěných knihoven, jejichž funkce je možné přímo využívat; pathlib je k dispozici od verze Python 3.4, obdobnou funkcionalitu nabízí pro všechny verze Pythonu knihovna os)
- například:

```
import pathlib #import knihovny pathlib
pathlib.Path("test.txt").unlink() #parametrem funkce je cesta k souboru
```

- pro smazání existujícího adresáře je možné využít funkce rmdir() z knihovny pathlib (pozor! adresář musí být prázdný)
- například:

```
import pathlib
pathlib.Path("D:/data/folder").rmdir() #parametrem funkce je cesta k adresari
```

https://www.freecodecamp.org/news/how-to-use-pathlib-module-in-python/

Otestování existence cesty k souboru

Otevírat soubor pro čtení (zápis) má smysl jen v tom případě, kdy cesta k němu a soubor samotný existuje. Lze ověřit například použitím:

```
#nacteni obsahu knihoven os a sys pro pouziti jejich funkci
import pathlib
import sys
input_file_path = "../data/program1.txt" #zadani relativni cesty k souboru
#pokud neexistuje zadana cesta, vytiskne se hlaska a ukonci se chod programu pomoci prikazu
sys.exit()
if not pathlib.Path(input_file_path).exists():
    print('soubor v zadane ceste', input_file_path, 'nebyl nalezen!')
    sys.exit()
else:
    print('vstupni soubor nalezen, program pokracuje')
```

Čeština a jiné speciální znaky

- Python verze 3.x používá jako výchozí znakovou sadu UTF-8 (Python starších verzí 2.x používal znakovou sadu ASCII)
- zdrojový kód pro verzi 3.x proto může obsahovat znaky české abecedy, důrazně se však nedoporučuje to používat!
- data uložená v datovém typu string (str) jsou v Python 3 v podobě Unicode (UTF-8) textu
- znakovou sadu pro skript můžeme nastavit pomocí příkazu na prvním (druhém) řádku souboru, například:

```
# -*- coding: utf-8 -*-
# -*- coding: windows-1250 -*-
```

Čeština a jiné speciální znaky

- Pokud vstupní soubor není založen na znakové sadě UTF-8 (ASCII), je potřeba kódování korektně nastavit v příkaze open()
- v prostředí Windows se s češtinou můžeme typicky setkat v souborech využívajících znakovou sadu Windows 1250 (= cp1250)
- i jiné znakové sady však nejsou neobvyklé

Important

- `utf-8` - a.k.a. Unicode - international standard (should be always used!)
- `iso-8859-1` - ISO standard for Western Europe and USA
- `iso-8859-2` - ISO standard for Central Europe (including Poland)
- `cp1250` or `windows-1250` - Central European encoding on Windows
- `cp1251` or `windows-1251` - Eastern European encoding on Windows
- `cp1252` or `windows-1252` - Western European encoding on Windows
- `ASCII` - ASCII characters only
- Since Windows 10 version 1903, UTF-8 is default encoding for Notepad!

Čeština a jiné speciální znaky

Příklad načítání souboru využívajícího znakovou sadu cp1250

Obsah vstupního souboru: *aábcčddďeéěfghiíjklmnňoópqrřsštťuúůvxyzž*

Pokud použijeme příkaz:

```
f = open("abeceda_cp1250.txt", "r", encoding = "utf-8")
```

Objeví se na standardním výstupu:

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe1 in position 1: invalid continuation byte

Pokud použijeme příkaz:

```
f = open("abeceda_cp1250.txt", "r", encoding = "cp1250")
```

Příkaz proběhne bez chyby a obsah souboru se korektně načte

Čeština a jiné speciální znaky

Příklad načtení souboru v UTF-8 a jeho konverze do výstupního souboru v cp1250:

```
#definice cesty ke vstupnimu a vystupnimu souboru
ff_name = 'test.txt'
target_file_name = 'test_cp1250.txt,
with open(ff_name, 'rb') as source_file: #otevirame vstupni soubor v binarnim rezimu pro cteni
  with open(target_file_name, 'wb') as dest_file: #otevirame vystupni soubor v binarnim rezimu pro zapis
    contents = source_file.read()
    dest_file.write(contents.decode('utf-8').encode('cp1250'))
```

funkce encode(): slouží k převodu textového zápisu na binární

funkce decode(): slouží k převodu binárního zápisu na textový

Parsování vstupu

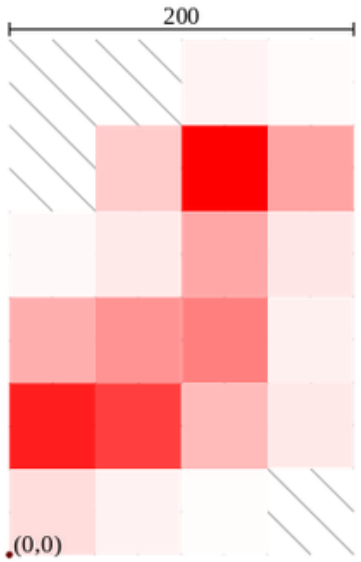
- Data uložená v (textovém) souboru mají standardně definovanou nějakou strukturu
- Jako příklad si můžeme představit jeden řádek záznamu z pomyslné meteo stanice:

  20220811;08:40;18.3;989.5;2.3;315;0.0 s významem:

  datum;čas;teplota;tlak;rychlost větru;směr větru;srážky

- Nebo ukázku rastrového souboru ve formátu ESRI ASCII grid:

Grid image



Values

	25	75	125	175
275	NA	NA	5	2
225	NA	20	100	36
175	3	8	35	10
125	32	42	50	6
75	88	75	27	9
25	13	5	1	NA

ASCII grid format

```
ncols         4
nrows         6
xllcorner     0.0
yllcorner     0.0
cellsize      50.0
NODATA_value  -9999
-9999 -9999 5 2
-9999 20 100 36
3 8 35 10
32 42 50 6
88 75 27 9
13 5 1 -9999
```

zdroj: https://en.wikipedia.org/wiki/Esri_grid

Parsování vstupu

- Velmi často potřebujeme přistupovat jen k určitým částem souboru – abychom je získali, editovali, smazali, apod.
- funkce split() slouží k rozdělení textového řetězce (string) na vstupu na samostatné prvky pomocí zadaného **oddělovače**, výstupem je seznam obsahující jednotlivé prvky uložené jako string
- funkce má dva parametry:
  - oddělovač: výchozím oddělovačem je **mezera**, lze však použít libovolný znak či kombinaci znaků
  - definice počtu rozdělení, které se mají realizovat, zadává se jako integer; výchozí hodnota = -1, split se tedy realizuje pro všechny výskyty oddělovače

https://www.w3schools.com/python/ref_string_split.asp

Python - split()

```
retezec = 'Dobrý den. Jak se máte?'
seznam = retezec.split() #pokud nespecifikuji oddělovač, použije se výchozí = mezera
print(seznam)
```

výstup:

['Dobrý', 'den.', 'Jak', 'se', 'máte?']

```
retezec2 = '20220811;08:40;18.3;989.5;2.3;315;0.0'
seznam2 = retezec2.split(';') #použiji středník jako oddělovač
print(seznam2)
```

výstup:

['20220811', '08:40', '18.3', '989.5', '2.3', '315', '0.0'] #prvky v seznamu jsou DT string!

Python - strip()

- Po načtení obsahu textového souboru jsou součástí získaného obsahu (funkce read -> string, funkce readlines -> seznam) i znaky zalomení konce řádků (\n)
- Pokud použijeme funkci split() bez specifikace oddělovače, dojde k automatickému odstranění konců řádků, viz příklad dále
- Pokud použijeme funkci split() s definicí oddělovače, nedojde k automatickému odstranění konců řádků
- Můžeme však uplatnit funkci strip(), která slouží k odstranění zadaného textového řetězce ze začátku a konce volaného textového řetězce

https://www.w3schools.com/python/ref_string_strip.asp

Python – split() a strip()

- Příklad: načteme textový soubor s jednoduchým obsahem:

První řádek
Druhý řádek
Třetí řádek
Čtvrtý řádek

Při aplikaci příkazu readlines() získáme

seznam = ['První řádek\n', 'Druhý řádek\n', 'Třetí řádek\n', 'Čtvrtý řádek\n']

Python – split() a strip()

1, Chceme pracovat s každým řádkem a slovem zvlášť, definujeme proto cyklus a použijeme split()

```
for radek in seznam:
    radek_split = radek.split() #nedefinujeme oddělovač, použije se mezera a odstraní se \n
    print(radek_split)
```

Výsledek: konce řádků se odstraní

['První', 'řádek']
['Druhý', 'řádek']
['Třetí', 'řádek']
['Čtvrtý', 'řádek']

Python – split() a strip()

2, Chceme pracovat s každým řádkem a slovem zvlášť, definujeme proto cyklus a použijeme split(' ') = vepíšeme mezeru jako parametr funkce split()

```
for radek in seznam:
    radek_split = radek.split(' ') #definujeme mezeru jako oddělovač
    print(radek_split)
```

Výsledek: konce řádků se neodstraní

```
['První', 'řádek\n']
['Druhý', 'řádek\n']
['Třetí', 'řádek\n']
['Čtvrtý', 'řádek\n']
```

Python – split() a strip()

3, Chceme pracovat s každým řádkem a slovem zvlášť, definujeme proto cyklus a použijeme split(' ') = vepíšeme mezeru jako parametr funkce split(), plus použijeme funkci strip() pro odstranění konce řádků

```
for radek in seznam:
    radek_bez_konce = radek.strip('\n') #odstraníme znak konce řádku
    radek_split = radek_bez_konce.split(' ') #definujeme mezeru jako oddělovač
    print(radek_split)
```

Výsledek: konce řádků se odstraní

['První', 'řádek']
['Druhý', 'řádek']
['Třetí', 'řádek']
['Čtvrtý', 'řádek']

Zkrácená varianta:

```
for radek in seznam:
    radek_split = radek.strip('\n').split(' ')
    print(radek_split)
```

Děkuji za pozornost

Michal Kačmařík

michal.kacmarik@vsb.cz

www.vsb.cz