

# Lekce 7: Textové soubory I – Jednoduchá tabulka

Python pro GIS - Čtení a zápis souborů

Vojtěch Barták, FŽP ČZU Praha

2025-10-31

## Table of contents

<b>1</b>	<b>Cíle lekce</b>	<b>3</b>
<b>2</b>	<b>Proč pracovat se soubory?</b>	<b>3</b>
2.1	Motivace . . . . .	3
2.2	Kde použijeme práci se soubory? . . . . .	3
<b>3</b>	<b>Čtení textových souborů</b>	<b>4</b>
3.1	Příprava - ukázkový soubor . . . . .	4
3.2	Otevření souboru - funkce open() . . . . .	4
3.2.1	Windows cesty - tři správné způsoby . . . . .	4
3.2.2	Proč je raw string důležitý? . . . . .	5
3.2.3	Časté chyby . . . . .	6
3.2.4	Relativní cesty (doporučeno pro projekty) . . . . .	6
3.3	Čtení celého souboru - read() . . . . .	7
3.4	Čtení po řádcích - readlines() . . . . .	7
3.5	with statement - bezpečnější způsob . . . . .	8
<b>4</b>	<b>Zpracování textu - strip() a split()</b>	<b>8</b>
4.1	Odstranění bílých znaků - strip() . . . . .	8
4.2	Rozdělení řetězce - split() . . . . .	9
4.2.1	split() s konkrétním separátorem . . . . .	9
<b>5</b>	<b>Praktická úloha - zpracování tabulky měst</b>	<b>9</b>
5.1	Zadání . . . . .	9
5.2	Řešení krok za krokem . . . . .	10
5.2.1	Krok 1: Načtení a výpis . . . . .	10

5.2.2	Krok 2: Zpracování každého řádku . . . . .	10
5.2.3	Krok 3: Filtrace měst nad 200 000 . . . . .	11
5.2.4	Krok 4: Celková populace . . . . .	12
5.2.5	Krok 5: Město s nejvyšší populací . . . . .	12
5.2.6	Kdy použít jaký přístup? . . . . .	13
5.2.7	Cvičení: Přepište na přímé procházení . . . . .	14
<b>6</b>	<b>Zápis do souborů</b>	<b>15</b>
6.1	Vytvoření nového souboru . . . . .	15
6.2	Přidání na konec - režim “a” . . . . .	15
6.3	Praktický příklad - export výsledků . . . . .	15
<b>7</b>	<b>Propojení s funkcemi z math_utils</b>	<b>16</b>
7.1	Příklad: Prvočísla ze souboru . . . . .	16
<b>8</b>	<b>Praktická cvičení</b>	<b>17</b>
8.1	Cvičení 1: Statistiky měst . . . . .	17
8.2	Cvičení 2: Filtrace a export . . . . .	18
<b>9</b>	<b>Shrnutí</b>	<b>18</b>
9.1	Co jsme se naučili . . . . .	18
9.2	Co bude příště? . . . . .	19
<b>10</b>	<b>Domácí úkol</b>	<b>19</b>
10.1	Varianta A (základní) . . . . .	19
10.2	Varianta B (pokročilá) . . . . .	19
10.3	Varianta C (výzva) . . . . .	20
10.4	Varianta D (výzva) . . . . .	20
<b>11</b>	<b>Cheatsheet</b>	<b>22</b>
<b>12</b>	<b>Poznámky pro vyučujícího</b>	<b>24</b>
12.1	Běžné chyby studentů . . . . .	24
12.2	Časový plán (90 min) . . . . .	24
12.3	Klíčové momenty . . . . .	25
12.3.1	with statement (15-30 min): . . . . .	25
12.3.2	strip() a split() (30-50 min): . . . . .	25
12.3.3	Encoding (v průběhu): . . . . .	25
12.3.4	Propojení s math_utils (80-90 min): . . . . .	25
12.4	Rizika . . . . .	26
12.5	Tipy . . . . .	26
12.6	Materiály k přípravě . . . . .	26

# 1 Cíle lekce

Po absolvování této lekce budete umět:

- Otevírat textové soubory pomocí `open()`
- Číst obsah souborů (`read()`, `readlines()`)
- Zapisovat do souborů
- Používat `with` statement pro bezpečnou práci se soubory
- Zpracovávat řádky textu pomocí `.strip()` a `.split()`
- Pracovat s encoding (UTF-8)
- Aplikovat funkce z `math_utils` na data ze souborů

**Časová dotace:** 90 minut

---

## 2 Proč pracovat se soubory?

### 2.1 Motivace

Dosud jsme pracovali s daty, která jsme **psali přímo do kódu**:

```
mesta = ["Praha", "Brno", "Ostrava"]
```

**Problém:** - Co když máme 100 měst? 1000? - Co když chceme data aktualizovat? - Co když data přicházejí odjinud (Excel, databáze, web)?

**Řešení:** Načítat data ze **souborů**!

### 2.2 Kde použijeme práci se soubory?

1. **Načítání dat** - tabulky, seznamy, konfigurace
2. **Export výsledků** - uložení analýz, reportů
3. **Logování** - záznam průběhu programu
4. **GIS workflow** - export atributových tabulek, import dat do ArcGIS

#### Preview ArcPy

V GIS často: - Exportujete atributovou tabulku do CSV - Zpracujete ji v Pythonu - Importujete zpět do ArcGIS  
Toto je běžný workflow!

---

## 3 Čtení textových souborů

### 3.1 Příprava - ukázkový soubor

Vytvořte soubor `mesta.txt` s tímto obsahem:

```
Praha 1300000  
Brno 380000  
Ostrava 290000  
Plzeň 170000  
Liberec 103000
```

Každý řádek obsahuje: **název města** (mezera) **populace**

### 3.2 Otevření souboru - funkce `open()`

```
soubor = open("mesta.txt", "r")
```

**Parametry:** - První parametr: **cesta k souboru** - Druhý parametr: **režim** - "r" = read (čtení) - "w" = write (zápis) - "a" = append (přidání na konec)



Pozor na cestu k souboru!

Pokud neuvedete plnou cestu, Python hledá soubor v **aktuálním adresáři** (kde je váš .py soubor).

---

#### 3.2.1 Windows cesty - tři správné způsoby

##### 1. Raw string (DOPORUČENO):

```
soubor = open(r"C:\Projekty\data\mesta.txt", "r")
```

Písmeno **r** před uvozovkami = **raw string**

**Co je raw string?**

- Python **ignoruje** escape sekvence (`\n`, `\t`, ...)

- Zpětné lomítko \ se bere doslovně
- Ideální pro Windows cesty!

## 2. Lomítka místo zpětných lomítek:

```
soubor = open("C:/Projekty/data/mesta.txt", "r")
```

Lomítka / fungují i ve Windows! Python je automaticky převede.

## 3. Dvojitě zpětné lomítko:

```
soubor = open("C:\\Projekty\\data\\mesta.txt", "r")
```

Každé \ musíte zdvojit: \\

---

### 3.2.2 Proč je raw string důležitý?

Problém bez raw stringu:

```
# CHYBA - \t se interpretuje jako TABULÁTOR!
cesta = "C:\temp\data.txt"
print(cesta)
```

Výstup:

```
C:  emp\data.txt
  ↑ zde je tabulátor místo \t
```

Správně s raw stringem:

```
# SPRÁVNĚ - r"..." vypne interpretaci escape sekvencí
cesta = r"C:\temp\data.txt"
print(cesta)
```

Výstup:

```
C:\temp\data.txt
```

---

### 3.2.3 Časté chyby

```
# CHYBA - \n na konci = nový řádek!
cesta = "C:\\projekty\\nové\\data.txt"
print(cesta) # C:\\projekty
              # ové\\data.txt (na dvou řádcích!)

# SPRÁVNĚ
cesta = r"C:\\projekty\\nové\\data.txt"
```

---

### 3.2.4 Relativní cesty (doporučeno pro projekty)

**Výhoda:** Projekt funguje i po přesunu na jiné PC.

```
# V aktuální složce
soubor = open("mesta.txt", "r")

# V podsložce
soubor = open("data/mesta.txt", "r")

# 0 složku výš, pak data/
soubor = open("../data/mesta.txt", "r")

# Dvě složky výš
soubor = open("../../data/mesta.txt", "r")
```

**Struktura projektu:**

```
MujProjekt/
  main.py          # Váš skript
  mesta.txt        # open("mesta.txt")
  data/
    mesta.txt      # open("data/mesta.txt")
    teploty.txt
```

---

**Doporučené postupy**

**Pro Windows cesty:**

```
with open(r"C:\Data\mesta.txt", "r", encoding="utf-8") as f:
    # Vždy používejte r"..."
```

**Pro relativní cesty:**

```
with open("data/mesta.txt", "r", encoding="utf-8") as f:
    # Lomítka / fungují všude (Windows, Linux, Mac)
```

**Zlaté pravidlo:**

- Absolutní cesta Windows → `r"C:\..."`
- Relativní cesta → `"data/soubor.txt"` (lomítka)

### 3.3 Čtení celého souboru - `read()`

```
soubor = open("mesta.txt", "r")
obsah = soubor.read()
print(obsah)
soubor.close() # DŮLEŽITÉ - zavřít soubor!
```

**Výsledek:**

```
Praha 1300000
Brno 380000
Ostrava 290000
Plzeň 170000
Liberec 103000
```

### 3.4 Čtení po řádcích - `readlines()`

```
soubor = open("mesta.txt", "r")
radky = soubor.readlines()
soubor.close()

print(radky)
```

**Výsledek:**

```
['Praha 1300000\n', 'Brno 380000\n', 'Ostrava 290000\n', 'Plzeň 170000\n', 'Liberec 103000\n']
```

💡 Všimněte si `\n`

`\n` je znak pro **nový řádek**. Budeme ho muset odstranit pomocí `.strip()`!

### 3.5 with statement - bezpečnější způsob

**Problém:** Můžete zapomenout zavřít soubor pomocí `.close()`

**Řešení:** Použijte `with` statement:

```
with open("mesta.txt", "r") as soubor:
    radky = soubor.readlines()

# Tady je soubor už automaticky zavřený!
print(radky)
```

**Výhody:** - Soubor se **automaticky zavře** na konci bloku - Bezpečnější - soubor se zavře i při chybě - Čitelnější kód

❗ Doporučení

**Vždy používejte `with` statement** pro práci se soubory!

---

## 4 Zpracování textu - `strip()` a `split()`

### 4.1 Odstranění bílých znaků - `strip()`

```
radek = "Praha 1300000\n"
cisty_radek = radek.strip()
print(cisty_radek) # "Praha 1300000"
```


**Co `strip()` odstraňuje:** - `\n` - nový řádek - `\t` - tabulátor - Mezery na začátku a konci



## 4.2 Rozdělení řetězce - split()

```
radek = "Praha 1300000"
casti = radek.split() # Rozdělí podle mezer
print(casti) # ['Praha', '1300000']

mesto = casti[0]      # "Praha"
populace = casti[1]   # "1300000" (STRING!)
```

 split() vrací STRINGY!

```
populace = casti[1]      # "1300000" (string)
populace_cislo = int(casti[1]) # 1300000 (int)
```

Pokud chcete číslo, musíte převést pomocí int() nebo float()!

### 4.2.1 split() s konkrétním separátorem

```
# CSV (hodnoty oddělené čárkou)
radek = "Praha,1300000,CZ"
casti = radek.split(",")
print(casti) # ['Praha', '1300000', 'CZ']

# Oddělení středníkem
radek = "Praha;1300000;CZ"
casti = radek.split(";")
```

---

## 5 Praktická úloha - zpracování tabulky měst

### 5.1 Zadání

Máte soubor mesta.txt:

Praha 1300000  
Brno 380000  
Ostrava 290000  
Plzeň 170000  
Liberec 103000

a soubor `mesta_all.txt` (ke stažení v Moodle).

### Úkoly:

1. Načíst soubor
2. Vypsát všechna města s populací  $> 200\,000$
3. Spočítat celkovou populaci všech měst
4. Najít město s nejvyšší populací
5. Totéž provést se souborem `mesta_all.txt` (pozor: zde jsou sloupce oddělené tabulátorem!)

## 5.2 Řešení krok za krokem

### 5.2.1 Krok 1: Načtení a výpis

```
with open("mesta.txt", "r", encoding="utf-8") as soubor:  
    radky = soubor.readlines()  
  
for radek in radky:  
    print(radek.strip())
```

**i** encoding="utf-8"

**UTF-8** je standard pro kódování českých znaků (čšřžý...).  
Vždy přidávejte `encoding="utf-8"` při práci s českými texty:

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

### 5.2.2 Krok 2: Zpracování každého řádku

```

with open("mesta.txt", "r", encoding="utf-8") as soubor:
    radky = soubor.readlines()

for radek in radky:
    radek = radek.strip() # Odstranit \n

    casti = radek.split() # Rozdělit
    mesto = casti[0]
    populace = int(casti[1]) # Převést na int!

    print(f"{mesto}: {populace} obyvatel")

```

**Výsledek:**

```

Praha: 1300000 obyvatel
Brno: 380000 obyvatel
Ostrava: 290000 obyvatel
Plzeň: 170000 obyvatel
Liberec: 103000 obyvatel

```

### 5.2.3 Krok 3: Filtrace měst nad 200 000

```

with open("mesta.txt", "r", encoding="utf-8") as soubor:
    radky = soubor.readlines()

for radek in radky:
    radek = radek.strip() # Odstranit \n

    casti = radek.split() # Rozdělit
    mesto = casti[0]
    populace = int(casti[1]) # Převést na int!

    if populace > 200000:
        print(f"{mesto}: {populace}")

```

**Výsledek:**

```

Praha: 1300000
Brno: 380000
Ostrava: 290000

```

#### 5.2.4 Krok 4: Celková populace

```
celkova_populace = 0

with open("mesta.txt", "r", encoding="utf-8") as soubor:
    radky = soubor.readlines()

for radek in radky:
    radek = radek.strip()

    casti = radek.split()
    populace = int(casti[1])
    celkova_populace += populace

print(f"Celková populace: {celkova_populace}")
```

#### 5.2.5 Krok 5: Město s nejvyšší populací

```
max_mesto = ""
max_populace = 0

with open("mesta.txt", "r", encoding="utf-8") as soubor:
    radky = soubor.readlines()

for radek in radky:
    radek = radek.strip()

    casti = radek.split()
    mesto = casti[0]
    populace = int(casti[1])

    if populace > max_populace:
        max_populace = populace
        max_mesto = mesto

print(f"Největší město: {max_mesto} ({max_populace})")
```

### 💡 Procházení souboru přímo

V ukázkách výše jsme vždy **nejprve načetli řádky pomocí readlines()** a následně **procházeli seznam**. Alternativně je možné **procházet přímo otevřený soubor**:

```
with open("cisla.txt", "r", encoding="utf-8") as soubor:
    for radek in soubor:
        print(radek.strip())
```

**Pozor na kurzor!** Po průchodu souborem kurzor zůstává na konci, takže další průchod už nefunguje:

```
with open("cisla.txt", "r", encoding="utf-8") as soubor:
    # První průchod - funguje
    for radek in soubor:
        print(radek.strip())

    # Druhý průchod - NEFUNGUJE! (nic se nevypíše)
    for radek in soubor:
        print(radek.strip())
```

### 5.2.6 Kdy použít jaký přístup?

Metoda	Kdy použít
<code>readlines()</code>	Potřebujete procházet data vícekrát nebo použít indexy
<b>Přímé procházení</b>	Jednorázové zpracování, šetří paměť u velkých souborů

### 5.2.7 Cvičení: Přepište na přímé procházení

Přepište předchozí úlohu tak, aby používala přímé procházení místo `readlines()`. Příklad: **Původní verze:**

```
with open("mesta.txt", "r", encoding="utf-8") as soubor:
    radky = soubor.readlines()

for radek in radky:
    radek = radek.strip() # Odstranit \n

    casti = radek.split() # Rozdělit
    mesto = casti[0]
    populace = int(casti[1]) # Převést na int!

    print(f"{mesto}: {populace} obyvatel")
```

**Verze s přímým procházením:**

```
with open("mesta.txt", "r", encoding="utf-8") as soubor:
    for radek in soubor:
        radek = radek.strip() # Odstranit \n

        casti = radek.split() # Rozdělit
        mesto = casti[0]
        populace = int(casti[1]) # Převést na int!

        print(f"{mesto}: {populace} obyvatel")
```

**Výhoda:** Jednodušší a paměťově efektivnější!

## 6 Zápis do souborů

### 6.1 Vytvoření nového souboru

```
with open("vysledky.txt", "w", encoding="utf-8") as soubor:
    soubor.write("Výsledky analýzy měst\n")
    soubor.write("=" * 30 + "\n")
    soubor.write("Praha: 1300000\n")
```

 Režim "w" PŘEPÍŠE soubor!

Pokud soubor existuje, "w" ho **VYMAŽE** a začne nový:

```
# První spuštění:
with open("test.txt", "w") as f:
    f.write("První text\n")

# Druhé spuštění - PŘEPÍŠE!
with open("test.txt", "w") as f:
    f.write("Druhý text\n")

# Výsledek: "Druhý text" (první text je pryč!)
```

### 6.2 Přidání na konec - režim "a"

```
with open("vysledky.txt", "a", encoding="utf-8") as soubor:
    soubor.write("Brno: 380000\n")
```

### 6.3 Praktický příklad - export výsledků

```
# Analýza
velka_mesta = []

with open("mesta.txt", "r", encoding="utf-8") as soubor:
    for radek in soubor:
        radek = radek.strip()
```

```

    if radek:
        casti = radek.split()
        mesto = casti[0]
        populace = int(casti[1])

        if populace > 200000:
            velka_mesta.append((mesto, populace))

# Export výsledků
with open("velka_mesta.txt", "w", encoding="utf-8") as soubor:
    soubor.write("Města s populací nad 200 000\n")
    soubor.write("=" * 40 + "\n\n")

    for mesto, populace in velka_mesta:
        soubor.write(f"{mesto}: {populace} obyvatel\n")

print("Výsledky uloženy do velka_mesta.txt")

```

---

## 7 Propojení s funkcemi z math\_utils

Nyní spojíme práci se soubory a funkce z minulé lekce!

### 7.1 Příklad: Prvočísla ze souboru

Soubor `cisla.txt`:

```

2
17
20
23
100

```

**Program:**



```

import math_utils

# Načíst čísla ze souboru
cisla = []
with open("cisla.txt", "r") as soubor:
    for radek in soubor:
        radek = radek.strip()
        if radek:
            cisla.append(int(radek))

# Najít prvočísla
prvocisla = []
for cislo in cisla:
    if math_utils.je_prvocislo(cislo):
        prvocisla.append(cislo)

# Uložit výsledky
with open("prvocisla.txt", "w") as soubor:
    soubor.write("Prvočísla ze vstupu:\n")
    for p in prvocisla:
        soubor.write(f"{p}\n")

print(f"Nalezeno {len(prvocisla)} prvočísel")

```

---

## 8 Praktická cvičení

### 8.1 Cvičení 1: Statistiky měst

Použijte soubor `mesta.txt` (případně `mesta_all.txt`) a vytvořte program, který:

1. Spočítá průměrnou populaci
2. Najde medián (použijte `math_utils.bubble_sort()` pro seřazení!)
3. Uloží statistiky do souboru `statistiky.txt` ve formátu:

Statistiky měst

=====

Počet měst: 5

Celková populace: 2,243,000

Průměrná populace: 448,600  
Medián: 290,000  
Největší město: Praha (1,300,000)  
Nejmenší město: Liberec (103,000)

## 8.2 Cvičení 2: Filtrace a export

Vytvořte program, který:

1. Načte `mesta.txt`
2. Vytvoří DVA nové soubory:
  - `velka_mesta.txt` - města nad 200 000
  - `mala_mesta.txt` - města pod 200 000
3. Každý soubor má formát:

Velká města (populace > 200 000)  
=====

Praha:	1,300,000
Brno:	380,000
...	

---

## 9 Shrnutí

### 9.1 Co jsme se naučili

**open()** - otevírání souborů s různými režimy  
**read()**, **readlines()** - čtení obsahu  
**write()** - zápis do souborů  
**with statement** - bezpečná práce se soubory  
**strip()** - odstranění bílých znaků  
**split()** - rozdělení řetězce  
**encoding="utf-8"** - správné kódování češtiny  
**Propojení s funkcemi** - použití `math_utils` na data ze souborů

## 9.2 Co bude příště?

V příští lekci:

- **CSV formát** - Comma-Separated Values
  - **Modul csv** - profesionální práce s CSV
  - **csv.reader()** a **csv.DictReader()**
  - **Složitější tabulky** - více sloupců, hlavičky
  - **Propojení s GIS** - atributové tabulky jako CSV
- 

## 10 Domácí úkol

### 10.1 Varianta A (základní)

Vytvořte soubor `teploty.txt`:

```
Pondělí 15
Úterý 18
Středa 22
Čtvrtek 19
Pátek 16
```

Napište program, který:

1. Načte data
2. Spočítá průměrnou teplotu
3. Najde den s nejvyšší teplotou
4. Uloží výsledky do `vysledky_teploity.txt`

### 10.2 Varianta B (pokročilá)

Vytvořte soubor `okresy.txt`:

```
Praha 1300000 496
Brno 380000 230
Ostrava 290000 214
Plzeň 170000 261
```

Formát: **Název Populace Rozloha** (všechny hodnoty oddělené mezerami)

**Napište program, který:**

1. Pro každý okres vypočítá hustotu obyvatel (populace/rozloha)
2. Použije `bubble_sort()` z `math_utils` pro seřazení okresů podle hustoty
3. Uloží výsledky do `hustota_okresu.txt`:

```
Okresy seřazené podle hustoty obyvatel
=====
Praha: 2,621 obyvatel/km2
Brno: 1,652 obyvatel/km2
...
```

### 10.3 Varianta C (výzva)

Zpracujte úkoly z varianty A pro **Tabulku meteorologických údajů** (ke stažení v Moodleu jako soubor `7770_all_variables.txt`).

#### Warning

**Pozor!** V souboru je třeba vyřešit:

- Soubor má hlavičku (první řádek obsahuje názvy sloupců, ne hodnoty!)
- Sloupce jsou oddělené tabulátorem!
- Tabulka obsahuje i jiné proměnné než teplotu (údaj o proměnné je ve sloupci **Variable**). Je třeba vyfiltrovat **jen teploty!**
- Tabulka obsahuje jak měsíční, tak roční hodnoty (ve sloupci **Month** hodnota **Annual**). Pracujte pouze s **ročními hodnotami!**

### 10.4 Varianta D (výzva)

Vytvořte vlastní modul `soubory_utils.py` s funkcemi:

```
def nacti_cisla(nazev_souboru):
    """Načte čísla ze souboru (jedno číslo na řádek)."""
    # ...

def uloz_cisla(nazev_souboru, cisla):
    """Uloží seznam čísel do souboru (jedno na řádek)."""
    # ...
```

```
def nacti_tabulku(nazev_souboru):  
    """Načte tabulku hodnot oddělených mezerami.  
    Vrátí seznam seznamů.  
    """  
    # ...
```

**Použijte tyto funkce** v hlavním programu pro zpracování dat!

---

## 11 Cheatsheet

```
# === OTEVŘENÍ SOUBORU ===
# Čtení
with open("soubor.txt", "r", encoding="utf-8") as f:
    obsah = f.read()

# Zápis (přepíše soubor)
with open("soubor.txt", "w", encoding="utf-8") as f:
    f.write("text\n")

# Přidání na konec
with open("soubor.txt", "a", encoding="utf-8") as f:
    f.write("další text\n")

# === ČTENÍ ===
# Celý soubor jako string
obsah = f.read()

# Seznam řádků
radky = f.readlines()

# Procházení řádek po řádku
for radek in f:
    # zpracování řádku

# === ZPRACOVÁNÍ TEXTU ===
radek = "  Praha 1300000\n "

radek.strip()          # "Praha 1300000" (bez \n a mezer)
radek.split()          # ['Praha', '1300000']
radek.split(",")       # Rozdělení čárkou

# === PŘEVODY ===
cislo_text = "1300000"
cislo = int(cislo_text)          # int
desetinne = float("3.14")       # float

# === FORMÁTOVÁNÍ VÝSTUPU ===
f.write(f"Město: {mesto}\n")
f.write(f"Populace: {populace}\n") # S oddělovači tisíců
```

```
# === KONTROLA PRÁZDNÉHO ŘÁDKU ===  
if radek:                # Není prázdný  
if radek.strip():        # Není prázdný ani po strip()  
  
# === ENCODING ===  
# VŽDY používejte pro české texty:  
open("soubor.txt", "r", encoding="utf-8")
```

---

## 12 Poznámky pro vyučujícího

### 12.1 Běžné chyby studentů

```
# 1. Zapomínají strip()
radek = "Praha 1300000\n"
casti = radek.split() # ['Praha', '1300000\n'] - \n je tam!
# Správně:
radek = radek.strip()
casti = radek.split() # ['Praha', '1300000']

# 2. Zapomínají převést na int
populace = casti[1] # "1300000" (string!)
vysledek = populace + 1000 # CHYBA!
# Správně:
populace = int(casti[1])

# 3. Zapomínají encoding
with open("soubor.txt", "r") as f: # Chyba s českými znaky!
# Správně:
with open("soubor.txt", "r", encoding="utf-8") as f:

# 4. Používají "w" místo "a"
with open("log.txt", "w") as f: # Každé spuštění VYMAŽE soubor!
# Správně pro přidání:
with open("log.txt", "a") as f:

# 5. Zapomínají uzavřít soubor (bez with)
f = open("soubor.txt", "r")
obsah = f.read()
# Zapomněli f.close()!

# 6. Rozdělení prázdného řádku
radek = "\n"
casti = radek.split() # [] - prázdný seznam!
mesto = casti[0] # CHYBA - IndexError!
# Řešení: kontrolovat if radek.strip():
```

### 12.2 Časový plán (90 min)



Čas	Obsah
0-15 min	Motivace, <code>open()</code> , <code>read()</code>
15-30 min	<code>readlines()</code> , <code>with</code> statement
30-50 min	<code>strip()</code> , <code>split()</code> , zpracování
50-70 min	Praktická úloha - města společně
70-80 min	Zápis do souborů
80-90 min	Propojení s <code>math_utils</code> , zadání úkolu

## 12.3 Klíčové momenty

### 12.3.1 `with` statement (15-30 min):

- **DŮLEŽITÉ:** Zdůraznit, že je to bezpečnější způsob
- Ukázat, co se stane při chybě (soubor se stejně zavře)
- Říct, že v praxi se VŽDY používá `with`

### 12.3.2 `strip()` a `split()` (30-50 min):

- **Prakticky ukázat** na projektoru
- Nechat studenty experimentovat
- Ukázat častou chybu: zapomenutí `strip()` před `split()`

### 12.3.3 Encoding (v průběhu):

- Zmínit při první práci se souborem
- Vysvětlit: “UTF-8 = standard pro češtinu”
- Říct: “Vždy to přidávejte, až to budete potřebovat, připomenu”

### 12.3.4 Propojení s `math_utils` (80-90 min):

- Ukázat, jak použít `je_prvocislo()` na data ze souboru
- Zdůraznit: “Takhle budete kombinovat ArcPy s jinými funkcemi!”

## 12.4 Rizika

### 1. `strip()` a `split()` mohou být matoucí (20 min → 25 min)

- Řešení: Praktické ukázky, nechat je experimentovat
- Mít připravené příklady na projektoru

### 2. Encoding problémy

- Řešení: Hned na začátku říct “vždy používejte utf-8”
- Pokud narazí na chybu, ukázat jak opravit

### 3. Studenti nestihnou cvičení

- Řešení: Cvičení 1 společně, Cvičení 2 jako domácí úkol
- Mít připravené řešení

## 12.5 Tipy

- Vytvořte soubor **mesta.txt** předem a sdílejte ho se studenty
- Na projektoru ukazujte **obsah souboru** vedle kódu
- Používejte **print()** často - ať vidí, co se děje
- Zdůrazněte: “CSV příští týden - tohle je příprava!”
- **Propojení s GIS:** “Atributové tabulky v ArcGIS exportujete jako CSV, zpracujete Pythonem, importujete zpět!”

## 12.6 Materiály k přípravě

- ☐ Soubor **mesta.txt** s ukázkovými daty
- ☐ Soubor **teploty.txt** pro domácí úkol
- ☐ Řešení všech cvičení
- ☐ Diagram: otevření → zpracování → zavření souboru