

# Informatika pro moderní fyziky (3) práce s tabulkami, hromadné zpracování dat, rozšíření RubyGems, ladění programů

František HAVLŮJ

*e-mail: haf@ujv.cz*

ÚJV Řež

oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2022/2023, 19. října 2022

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
  - Vytvoříme excelovskou tabulku
- 3 Kde je chyba?
- 4 Zpracování textu
  - Obecný rozbor
  - Načítání výstupního souboru
  - Zápis všech výsledků do tabulky

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
- 3 Kde je chyba?
- 4 Zpracování textu

- zpracování většího množství dat
-

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler**
  - Vytvoříme excelovskou tabulku
- 3 Kde je chyba?
- 4 Zpracování textu

## Knihovny (gemy) jsou základ

- existují mnohá rozšíření, tzv. knihovny – v ruby se jim říká **rubygems**
- aktuálně nás zajímá něco na práci s excelovskými soubory
- gemy jdou sice instalovat na systémové úrovni, ale z toho je pak zase jenom neštěstí
- použijeme radši **bundler**, správce gemů pro každého: vyřeší za nás závislosti a postará se o snadnou instalaci

## Máme bundler?

- otestujeme rubygems: `gem -v`
- pokud není, zapláčeme, protože jsme asi špatně nainstalovali Ruby
- otestujeme bundler: `bundle -v`
- pokud bundler není, doinstalujeme `gem install bundler`

## Jak na to

- najdu si, která knihovna mě zajímá (třeba na [rubygems.org](http://rubygems.org) nebo kdekoli jinde): my bychom rádi **rubyXL**  
`https://github.com/weshatheleopard/rubyXL`
- vytvořím si prázdný Gemfile – tam se specifikuje, které gemy chci používat: `bundle init`
- do gemfilu – je to normální Ruby skript! – dopíšu `gem "rubyXL"`
- nainstaluju: `bundle install`



## Jak použít?

- na začátku svého skriptu pak musím nahrát bundler:
- `require "bundler/setup"`
- a teď už můžu nahrát jakýkoli gem:
- `require "rubyXL"`

## RTFM, RTFM, RTFM

- na stránkách `rubyXL` se nachází spousta příkladů a návodů – <https://github.com/weshatheleopard/rubyXL>
- kromě toho má i slušnou dokumentaci (GIYF / "rubyxl docs") – <http://www.rubydoc.info/gems/rubyXL/3.3.15>
- napoprvé navedu do začátku:

```
workbook = RubyXL::Workbook.new
worksheet = workbook[0]
worksheet.add_cell(0, 0, "A1")
workbook.write("data.xlsx")
```

## Kompletní test

... se nachází v data/bundler.

### Gemfile

```
source "https://rubygems.org"  
gem "rubyXL"
```

### Ruby skript

```
require "bundler/setup"  
require "rubyXL"  
  
workbook = RubyXL::Workbook.new  
worksheet = workbook[0]  
worksheet.sheet_name = "Data test"  
worksheet.add_cell(0, 0, "A1")  
workbook.write("data.xlsx")
```

## Jednoduché cvičení

- použijte soubor `numbers/data_two_1.csv`
- vytvořte excelovský soubor se dvěma listy, na obou bude sloupec 1, sloupec 2 a součet
- na jednom součet bude jako číslo (sečte to váš skript)
- na druhém bude součet jako excelovský vzorec

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
- 3 Kde je chyba?**
- 4 Zpracování textu

## Ladění programů

- v každém programu je aspoň jedna chyba
- není důležité nedělat chyby, ale je nutné je umět najít
- když si program/Ruby na něco stěžuje, tak si to přečtěte, jinak se nic nedozvíte
- pokud nepoznám, v čem je chyba, jsem bezezbytku ztracen
- následují tři úlohy, kde je úkolem najít všechny chyby
- z didaktických důvodů postupujte metodou tupého spouštění a postupného opravování

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
- 3 Kde je chyba?
- 4 Zpracování textu**
  - Obecný rozbor
  - Načítání výstupního souboru
  - Zápis všech výsledků do tabulky

## Problém č. 2: mnoho výpočtů, inženýrova smrt

### Zadání

Při přípravě základního kritického experimentu je pomocí MCNP potřeba najít kritickou polohu regulační tyče R2. Jak se tato poloha změní při změně polohy tyče R1?



## Co máme k dispozici?

### MCNP

Pokud připravíme vstupní soubor (v netriviální formě obsahující polohy regulačních tyčí R1 a R2), spočítá nám keff.

Potřebovali bychom ale něco na:

- 1 vytvoření velkého množství vstupních souborů
- 2 extrakci keff z výstupních souborů
- 3 popřípadě na vyhodnocení získaných poloh tyčí a keff

## Pracovní postup

- 1 načíst keff z výstupního souboru MCNP

## Pracovní postup

- 1 načíst keff z výstupního souboru MCNP
- 2 vygenerovat potřebné vstupní soubory

## Pracovní postup

- 1 načíst keff z výstupního souboru MCNP
- 2 vygenerovat potřebné vstupní soubory
- 3 vyrobit BAT soubor na spuštění výpočtů

## Pracovní postup

- 1 načíst keff z výstupního souboru MCNP
- 2 vygenerovat potřebné vstupní soubory
- 3 vyrobit BAT soubor na spuštění výpočtů
- 4 načíst výsledky ze všech výstupních souborů do jedné tabulky

## Nejprve najdeme, kde je ve výstupu z MCNP žádané keff:

.....

```
the k(trk length) cycle values appear normally distributed at the 95 percent confidence
```

```
-----  
|  
| the final estimated combined collision/absorption/track-length keff = 1.00353 with an estimated  
|  
| the estimated 68, 95, & 99 percent keff confidence intervals are 1.00329 to 1.00377, 1.00300 to  
|  
| the final combined (col/abs/tl) prompt removal lifetime = 1.0017E-04 seconds with an estimated  
|
```

.....

# Algoritmus

- 1 najít řádek s keff

# Algoritmus

- 1 najít řádek s keff
- 2 vytáhnout z něj keff, takže například:



# Algoritmus

- 1 najít řádek s keff
- 2 vytáhnout z něj keff, takže například:
- 3 rozdělit podle rovnítka

# Algoritmus

- 1 najít řádek s keff
- 2 vytáhnout z něj keff, takže například:
- 3 rozdělit podle rovnítka
- 4 druhou část rozdělit podle mezer

# Algoritmus

- 1 najít řádek s keff
- 2 vytáhnout z něj keff, takže například:
- 3 rozdělit podle rovnítka
- 4 druhou část rozdělit podle mezer
- 5 vzít první prvek

## Realizace (1/5)

```
keff = nil

File.foreach("cl_1o") do |line|

end

puts keff
```

## Realizace (2/5)

```
keff = nil

File.foreach("cl_1o") do |line|
  if line.include?("final estimated combined")

    end
end

puts keff
```

## Realizace (3/5)

```
keff = nil

File.foreach("cl_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")

    end
end

puts keff
```

## Realizace (4/5)

```
keff = nil

File.foreach("cl_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")
    b = a[1].split

  end
end

puts keff
```

## Realizace (5/5)

```
keff = nil

File.foreach("cl_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")
    b = a[1].split
    keff = b[0]
  end
end

puts keff
```



## Jak na to

Máme všechno, co potřebujeme:

- načtení keff z jednoho výstupního souboru (`File.foreach`, `include a split`)
- procházení adresáře (`Dir.each`)
- zápis do souboru (`File.open` s parametrem `w` anebo `File.write`)

Takže už to stačí jen vhodným způsobem spojit dohromady!

## Realizace

```
Dir["*o"].each do |filename|
  keff = nil

  File.foreach(filename) do |line|
    if line.include?("final estimated combined")
      a = line.split("=")
      b = a[1].split
      keff = b[0]
    end
  end

  puts "#{filename} #{keff}"
end
```

## Výstup

Výsledkem je perfektní tabulka:

```
outputs/c_0_0o 0.94800  
outputs/c_0_10o 0.99800  
outputs/c_0_1o 0.94850  
outputs/c_0_2o 0.95000  
outputs/c_0_3o 0.95250  
outputs/c_0_4o 0.95600  
...
```

Hloupé je, že nikde nemáme tu polohu tyčí.

## Chytrá horákyně

... by jistě vyrobila toto:

```
0 0 0.94800
0 10 0.99800
0 1 0.94850
0 2 0.95000
0 3 0.95250
0 4 0.95600
...
```

Nápovědou je funkce `split` (podle podtržítka) a metoda `to_i`

## Realizace chytré horákyně

```
Dir["outputs/*o"].each do |filename|
  keff = nil

  File.foreach(filename) do |line|
    if line.include?("final estimated combined")
      a = line.split("=")
      b = a[1].split
      keff = b[0]
    end
  end

  s = filename.split("_")
  puts "#{s[1].to_i} #{s[2].to_i} #{keff}"
end
```

Co jsme se naučili minule  
Rozšíření Ruby: RubyGems a Bundler  
Kde je chyba?  
Zpracování textu

Obecný rozbor  
Načítání výstupního souboru  
Zápis všech výsledků do tabulky

A to je vše, přátelé!

