# Informatika pro moderní fyziky (5)
# Tvorba textových dokumentů

František HAVLŮJ

*e-mail: haf@ujv.cz*

ÚJV Řež
oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2013/2014
12. listopadu 2013

# Obsah

- komplexnější řešení úlohy na zpracování dat, vykreslování sady grafů
- organizace jednoduchých skriptů - rake
- rychlý úvod do LaTeXu

# Obsah

# Klávesnice a myš

- myš je dobrá na grafiku a jako alternativa k tabletu
- taky se hodí tam, kde se potřebuju přesouvat mezi položkami, které nemají jednoznačné pořadí nebo prostorový vztah (neseřazené ikony na ploše, rozhraní s mrakem oken atd.)
- případně ještě na použití menu pro úkony, které dělám jednou za uherský rok
- naopak na programování je nejlepší na myš vůbec nešahat a používat skoro jenom klávesnici
- extrémní školy dokonce brojí proti kurzorovým šipkám, protože (na velké klávesnici) nutí měnit polohu rukou, což je pomalé

# Přepínání jazyků

- je dobré se mu vyhnout, protože to opravdu trochu otravuje (i když se s tím dá docela dobře žít, pokud máte dobrou klávesovou zkratku)
- rozhodně stojí za to zjistit – například pro psaní v LaTeXu – kde na české klávesnici máte potřebné speciální znaky (v tomto případě zejména backslash a složené závorky)
- chytré editory mají různé pochystávky a makra, která vám umožní se těmto speciálním znakům defacto vyhnout

# Klávesové zkratky

- jako s programováním – musím se něco naučit / zapamatovat, ale pak mi to ušetří hromadu času
- minimálně základní sadu stojí za to se naučit
- často jdou ručně editovat, ale většinou to není nutné (a je to stejně na houby, pokud zrovna nesedíte u svého počítače)
- jako s hudebním nástrojem – za čas už neznáte ty zkratky, ale prostě je umíte zmáčknout bez přemýšlení
- hodně jich je sdílených napříč programy a editory

# Klávesové zkratky - MS Windows

- copy-paste
- undo
- přepínání aplikací
- přepínání oken v rámci aplikace

# Klávesové zkratky – Notepad++

- pohyb v textu po slovech a stránkách
- uložení, otevření, zavření
- změna odsazení bloku `Tab / Shift+Tab`
- přepínání mezi soubory
- zakomentovat/odkomentovat `Ctrl+Q`

# Obsah

# Úkol na dnešek

- pro jeden blok JE mám provozní data - v určitých dnech hodnotu koncentrace kyseliny borité a axiálního ofsetu - pro několik kampaní (blíže neurčený počet)
- chci vyrobit přehledové PDF, které bude hezky prezentovat grafy obou veličin pro každou kampaň a k tomu i tabulky
- data pro jednotlivé kampaně mám v CSV souborech, každý má tři sloupce (datum, cB, AO)

# Rozbor

- načíst tabulky a vykreslit grafy umíme
- převést tabulky v CSV na tabulky v LaTeXu se záhy naučíme
- vložit obrázek do latexu taky umíme
- předem neznámý počet souborů nás netrápí (`Dir["*.csv"]`)

# Tak nejdřív ty grafy

- to už je vážně obehraná písnička, ale tady je aspoň trochu změna
- potřebujeme vybrat, které dva sloupce použít - parametr `using`
- `plot "data.csv" using 1:2`
- datum na vodorovné ose – potřeba načíst ve správném formátu atd.
- `set xdata time`
- `set timefmt "%m/%d/%Y"`
- s hvězdičkou: nastavit nadpis a popisky os

# Ne tak úplně chytře

```
Dir["*.csv"].each do |fn|
  base = fn.split(".").first

  File.open("#{base}_bc.gp", 'w') do |f|
    f.puts "set terminal png"
    f.puts "set xdata time"
    f.puts "set timefmt \"%m/%d/%Y\""
    f.puts "set output \"#{base}_bc.png\""
    f.puts "plot \"#{base}.csv\" using 1:2"
  end
  `gnuplot #{base}_bc.gp`

  File.open("#{base}_ao.gp", 'w') do |f|
    f.puts "set terminal png"
    f.puts "set output \"#{base}_ao.png\""
    f.puts "set xdata time"
    f.puts "set timefmt \"%m/%d/%Y\""
    f.puts "plot \"#{base}.csv\" using 1:3"
  end
  `gnuplot #{base}_ao.gp`
end
```

# DRY

- základní paradigma: DRY = don't repeat yourself
- nemá cenu psát dvě věci stejně
- použití `copy and paste` při programování je varovný signál
- pokud nejsou stejně, ale skoro stejně, je potřeba trochu chytrosti
- připomeňme si hash: `{'a' => 1, 'b' => 2}`
- přes hash se dá iterovat:

```
{'a' => 1, 'b' => 2}.each do |key, value|
```

# Grafy – DRY

```
Dir["*.csv"].each do |fn|
  base = fn.split(".").first
  {'bc' => 2, 'ao' => 3}.each do |var, col|
    File.open("#{base}_#{var}.gp", 'w') do |f|
      f.puts "set terminal png"
      f.puts "set xdata time"
      f.puts "set timefmt \"%m/%d/%Y\""
      f.puts "set output \"#{base}_#{var}.png\""
      f.puts "plot \"#{base}.csv\" using 1:#{col}"
    end
    `gnuplot #{base}_#{var}.gp`
  end
end
```

# Jak na tabulky

- tabulky budou dost rozsáhlé a montovat je přímo nějak do latexových vstupů je asi spíš nepraktické, naštěstí to jde i jinak
- naštěstí má LaTex příkaz `\input`, kterým můžeme prostě vložit do dokumentu nějaký externí soubor
- takže si nejdřív přichystáme soubory s tabulkami a pak se na ně budeme už jenom odkazovat

# Jak na tabulky v LaTeXu (1)

Základem tabulky je prostředí `tabular` s definicí počtu a zarovnání sloupců:

```
\begin{tabular}{lrr}
  ...
\end{tabular}
```

# Jak na tabulky v LaTeXu (2)

Uvnitř tabulky se sloupce oddělují ampersandem a řádky dvojitým backslashem:

```
\begin{tabular}{lrr}
  Data 1 & a & 1.0 \\
  Data 2 & b & 2.0 \\
  Data 3 & c & 3.0 \\
\end{tabular}
```

# Jak na tabulky v LaTeXu (3)

Přidání mřížky je nesnadné, leč proveditelné a vlastně docela dobře vymyšlené - přidáváme jednotlivé čáry po sloupcích a řádcích:

```
\begin{tabular}{|l|r|r|}
  \hline
  Data 1 & a & 1.0 \\
  \hline
  Data 2 & b & 2.0 \\
  \hline
  Data 3 & c & 3.0 \\
  \hline
\end{tabular}
```

# Úkol na teď: výroba tabulek

- vyrobit z CSV souboru (tři sloupce) dvě LaTeX tabulky (po dvou sloupcích)
- postarat se, aby byly hezké

# Obsah

# Úskalí šablon

- snadno umíme nahradit jeden řetězec druhým
- trochu méně pohodlné pro větší bloky textu
- navíc by se hodila nějaká logika (cyklus) přímo v šabloně
- naštěstí jsou na to postupy

# ERb (Embedded Ruby)

- lepší šablona - “aktivní text”
- používá se například ve webových aplikacích
- hodí se ale i na generování latexových dokumentů, resp. všude, kde nám nesejde na whitespace
- poměrně jednoduchá syntax, zvládne skoro všechno (viz předmět MAA3)

# Základní syntaxe ERb (1)

Jakýkoli Ruby příkaz, přiřazení, výpočet ...

```
<% a = b + 5 %>
<% list = ary * ", " %>
```

# Základní syntaxe ERb (2)

Pokud chci něco vložit, stačí přidat rovnítko

```
<%= a %>
<%= ary[1] %>
<%= b + 5 %>
```

# Základní syntaxe ERb (3)

Radost je možnost použít bloky a tedy i iterátory apod. v propojení s vkládaným textem:

```
<% (1..5).each do |i| %>
Number <%= i %>
<% end %>
<% ary.each do |x| %>
Array contains <%= x %>
<% end %>
```

# ERb – shrnutí

- dobrý sluha, ale špatný pán
- můžu s tím vyrobit hromadu užitečných věcí na malém prostoru
- daň je velké riziko zamotaného kódu a nízké přehlednosti (struktura naprosto není patrná na první pohled, proto je namístě ji držet maximálně jednoduchou)

# Důležité upozornění

- oddělení modelu a view
- přestože lze provádět zpracování dat a výpočty přímo v ERb, je to nejvíc nejhorší nápad
- je chytré si všechno připravit v modelu (tj. v Ruby skriptu, kterým data chystáme)
- a kód ve view (tj. v ERb šabloně) omezit na naprosté minimum

# Jak ze šablony udělat výsledek

## Příklad překladu ERb

```
require 'erb'

erb(template, filename, {:x => 1, :y => 2})
```

# Příklad – kreslení grafů z minula

## template.gp

```
set terminal png
set output "plot_<%=n%>.png"
plot "data_<%=n%>.csv"
```

```
(1..10).each do |i|
  erb("template.gp", "plot_#{i}.gp", {:n => i})
end
```

# A to je vše, přátelé!