Informatika pro moderní fyziky (2)
základy Ruby, zpracování textu

František HAVLŮJ

e-mail: haf@ujv.cz

ÚJV Řež
oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2016/2017
12. října 2016

1. Co jsme se naučili minule
2. Úvod do jazyka Ruby
   - Ještě chvilku v IRb
   - Pole
   - Vstup a výstup
   - Problém č. 2: jehla v kupce sena
3. Rozšíření Ruby: RubyGems a Bundler
4. Vytvoříme excelovskou tabulku

Obsah

1. Co jsme se naučili minule
2. Úvod do jazyka Ruby
3. Rozšíření Ruby: RubyGems a Bundler
4. Vytvoříme excelovskou tabulku

- základní principy automatizace
- CSV soubory a Gnuplot
- příkazový řádek / terminál
- dávkové (BAT) soubory
- představení skriptovacích jazyků
- interpret Ruby a IRb
- letem světem Ruby

Obsah

1 Co jsme se naučili minule

2 Úvod do jazyka Ruby
- Ještě chvilku v IRb
- Pole
- Vstup a výstup
- Problém č. 2: jehla v kupce sena

3 Rozšíření Ruby: RubyGems a Bundler

4 Vytvoříme excelovskou tabulku

OOP - volání metod

Klasickým příkladem je například počet znaků v řetězci.

procedurální jazyky

```
strlen("retezec")
```

OOP - volání metod

Klasickým příkladem je například počet znaků v řetězci.

procedurální jazyky

```
strlen("retezec")
```

Můžeme místo toho nahlížet na řetězec jako na objekt:

OOP - volání metod

Klasickým příkladem je například počet znaků v řetězci.

procedurální jazyky

```
strlen("retezec")
```

Můžeme místo toho nahlížet na řetězec jako na objekt:

objektově orientované jazyky

```
"retezec".length
```

Práce s řetězci

Délka řetězce

```
"krabice".length
"kocour".size
```

Práce s řetězci

Délka řetězce

```
"krabice".length
"kocour".size
```

Ořezání mezer a konce řádku

```
"   hromada ".strip
" koleso   ".lstrip
```

Triky s řetězci

Hledání

```
"koleno na kole".include?("kole")
```

Triky s řetězci

Hledání

```
"koleno na kole".include?("kole")
```

Nahrazení

```
"volej kolej".sub("olej", "yber")
"baba a deda".gsub("ba", "ta")
```

Dokumentace

GIYF: Google is your friend

```
ruby api string
```

API dokumentace

```
http://ruby-doc.org/core-2.3.1/String.html
```

Pole je seznam

Literál, přiřazení

```
a = []
a << 1
a << "string"
b = []
```

Pole je seznam

Literál, přiřazení

```
a = []
a << 1
a << "string"
b = []
```

Délka, řazení, vytřídění, převracení

```
["a", "bb", "CCC"].size
[4, 2, 6].sort
[2, 5, 3, 3, 4, 1, 2, 1].uniq.sort
[4, 2, 6].reverse
```

Čtení z pole

Indexace

```
a = [1,2,3]
a[1]
a[3]
```

Čtení z pole

Indexace

```
a = [1,2,3]
a[1]
a[3]
```

Záporný index a rozsahy

```
a = [1,2,3,4,5,6]
a[-1]
a[-2]
a[0..3]
```

Pole vs řetězce

Řetězec je skoro pole znaků

```
"kopr"[2]
"mikroskop"[0..4]
```

Pole vs řetězce

Řetězec je skoro pole znaků

```
"kopr"[2]
"mikroskop"[0..4]
```

Leccos funguje!

```
"abcd".reverse
[1,2,3].size
```

Rozdělit dle potřeby

```
"a b  c d".split
"a b,c d".split(",")
```

Operátor a operatér

Malé bezvýznamné plus

```
"alfa" + "beta"
[1, 2] + [3, 4]
```

Operátor a operatér

Malé bezvýznamné plus

```
"alfa" + "beta"
[1, 2] + [3, 4]
```

Násobilka

```
"kolo" * 5
[1, 2, 3] * 3
["a", "b", "c"] * ","
```

Převod mezi typy

```
"123".to_i
1250.to_s
"0.6".to_f
```

Převod mezi typy

```
"123".to_i
1250.to_s
"0.6".to_f
```

Rozsahy

```
(1..4)
(0...10)
(1..5).to_a
```

Rozsahy

```
(1..4)
(0...10)
(1..5).to_a
```

Řetězce a symboly

```
"letadlo"
:letadlo
```

Logické hodnoty a spol.

Jednoduchá porovnání

```
2 + 2 < 5
"alfa" != "beta"
(x == y) and (y == z)
```

(pozor na = versus ==)

Logické hodnoty a spol.

Jednoduchá porovnání

```
2 + 2 < 5
"alfa" != "beta"
(x == y) and (y == z)
```

(pozor na = versus ==)

Chytré metody s otazníkem

```
[1, 2, 3].include?(3)
"abc".include?("bc")
```

Úlohy

Konverze II

- vyzkoumejte, jak se chová `to_f` a `to_i` pro řetězce, které nejsou tak úplně číslo

Úlohy

Konverze II

- vyzkoumejte, jak se chová `to_f` a `to_i` pro řetězce, které nejsou tak úplně číslo

Palindrom

- z libovolného řetězce vyrobte palindrom (osel → oselleso)
- z libovolného řetězce vyrobte palindrom s lichým počtem znaků (osel → oseleso)

Úlohy

Palindrom / řešení

```
s = "osel"

puts s + s.reverse
puts s[0..-2] + s.reverse
```

Výpis na terminál

Print vs puts

```
print "jedna"
puts "dve"
```

Výpis na terminál

Print vs puts

```
print "jedna"
puts "dve"
```

Na všechno platí inspect

```
puts "2 + 2 = #{2+2}"
puts [1,2,3].inspect
```

Iterátory

Jednoduchý rozsah

```
(1..5).each do
  puts "Cislo"
end
```

Iterátory

Jednoduchý rozsah

```
(1..5).each do
  puts "Cislo"
end
```

S polem a proměnnou

```
[1, 2, 3].each do |i|
  puts "Cislo #{i}"
end
```

Ještě jedna věc: formátovaný výstup

- často potřebuju něco vytisknout ‘hezky’, zarovnané, se správným počtem desetinných míst apod.
- v C na to je funkce `sprintf` a alternativa v Ruby funguje podobně
- je na to operátor `%`: *formát* `%` *data*

```
"%10s" % "kolo"
"%-6d" % a
"%8.3f +- %8.3f" % b
```

Úlohy

- vypište prvních deset druhých mocnin (1 * 1 = 1, 2 * 2 = 4 atd.)
- vypište malou násobilku
- vypište prvních N členů Fibonacciho posloupnosti (1, 1, 2, 3, 5, 8 ...)
- vypište všechna prvočísla menší než N (kdo použije Erathostenovo síto, má plus)

Úlohy

Mocniny / řešení

```
(1..10).each do |x|
  print x
  print " * "
  print x
  print " = "
  puts x*x
end
```

Úlohy

Mocniny / řešení

```
(1..10).each do |x|
  print x
  print " * "
  print x
  print " = "
  puts x*x
end
```

Mocniny / lepší řešení

```
(1..10).each do |x|
  puts "#{x} * #{x} = #{x*x}"
end
```

Úlohy

Násobilka / řešení

```
(1..10).each do |a|
  (1..10).each do |b|
    puts "#{b} * #{a} = #{a*b}"
  end
end
```

Úlohy

Násobilka / řešení

```
(1..10).each do |a|
  (1..10).each do |b|
    puts "#{b} * #{a} = #{a*b}"
  end
end
```

Násobilka / hezké řešení

```
(1..10).each do |a|
  (1..10).each do |b|
    puts "%2d * %2d = %3d" % [b, a, a * b]
  end
end
```

Úlohy

Fibonacci / řešení

```
a, b = 1, 1
20.times do
  puts a
  c = a + b
  a = b
  b = c
end
```

Fibonacci / jiné řešení

```
a, b = 1, 1
20.times do
  puts a
  a, b = b, a + b
end
```

Úlohy

Erathostenes / řešení

```
n = 100

ary = (2..n).to_a
ary.each do |x|
  y = x
  while y <= n
    y += x
    ary.delete(y)
  end
end

puts ary.inspect
```

Čtení ze souboru

Šikovný iterátor po řádcích

```
File.foreach("data.txt") do |line|
  ...
end
```

Čtení ze souboru

Šikovný iterátor po řádcích

```
File.foreach("data.txt") do |line|
  ...
end
```

Celý soubor najednou

```
string = File.read("data.txt")
```

V podmínce

If nebo Unless

```
if "velikost".include?("kost")
  puts "s kosti"
end
unless 7 > 8
  puts "poporadku"
end
```

V podmínce

If nebo Unless

```
if "velikost".include?("kost")
  puts "s kosti"
end
unless 7 > 8
  puts "poporadku"
end
```

Přirozený jazyk

```
puts "je tam!" if "podvodnik".include? "vodnik"
puts "pocty" unless 2 + 2 == 5
a = [1]
a << a.last * 2 while a.size < 10
```

Úlohy

V souboru `data/text_1.txt`:

- spočítejte všechny řádky
- spočítejte všechny řádky s výskytem slova kapr
- spočítejte počet výskytů slova kapr (po řádcích i v kuse)

Úlohy

Kapři / řešení

```
n, n_kapr, nn_kapr = 0, 0, 0
File.foreach("../data/text_1.txt") do |line|
  n += 1
  n_kapr += 1 if line.include?("kapr")
  nn_kapr += line.scan("kapr").size
end

nn_kapr_bis = File.read("../data/text_1.txt").scan("kapr").size

puts "Celkem radku: #{n}"
puts "Radku s kaprem: #{n_kapr}"
puts "Celkem kapru: #{nn_kapr}"
puts "         nebo: #{nn_kapr_bis}"
```

Zápis do katastru

Soubor se otevře a pak už to známe

```
f = File.open("text.txt", 'w')
f.puts "Nazdar!"
f.close
```

Zápis do katastru

Soubor se otevře a pak už to známe

```
f = File.open("text.txt", 'w')
f.puts "Nazdar!"
f.close
```

The Ruby way

```
File.open("text.txt", 'w') do |f|
  f.puts "Nazdar!"
end
```

Úlohy

Z dat v souboru `data/data_two_1.csv`:

- vyberte pouze druhý sloupec
- sečtěte oba sloupce do jednoho
- vypočtěte součet obou sloupců
- vypočtěte průměr a RMS druhého sloupce

S hvězdičkou:

- použijte soubory `*multi*`
- proveďte pro všechny čtyři CSV soubory

Úlohy

CSV(1) / řešení

```
File.open("druhy_sloupec.csv", 'w') do |f|
  File.foreach("../data/data_two_1.csv") do |line|
    f.puts line.strip.split[1]
  end
end
```

Úlohy

CSV(2) / řešení

```
File.open("sectene_sloupce.csv", 'w') do |f|
  File.foreach("../data/data_two_1.csv") do |line|
    data = line.strip.split
    f.puts data[0].to_f + data[1].to_f
  end
end
```

Úlohy

CSV(3) / řešení

```
x0 = 0
x1 = 0
n = 0
File.foreach("../data/data_two_1.csv") do |line|
  data = line.strip.split
  x0 += data[0].to_f
  x1 += data[1].to_f
  n += 1
end
puts "Prvni sloupec: soucet #{x0}"
puts "Druhy sloupec: soucet #{x1}"
```

Úlohy

CSV(4) / řešení

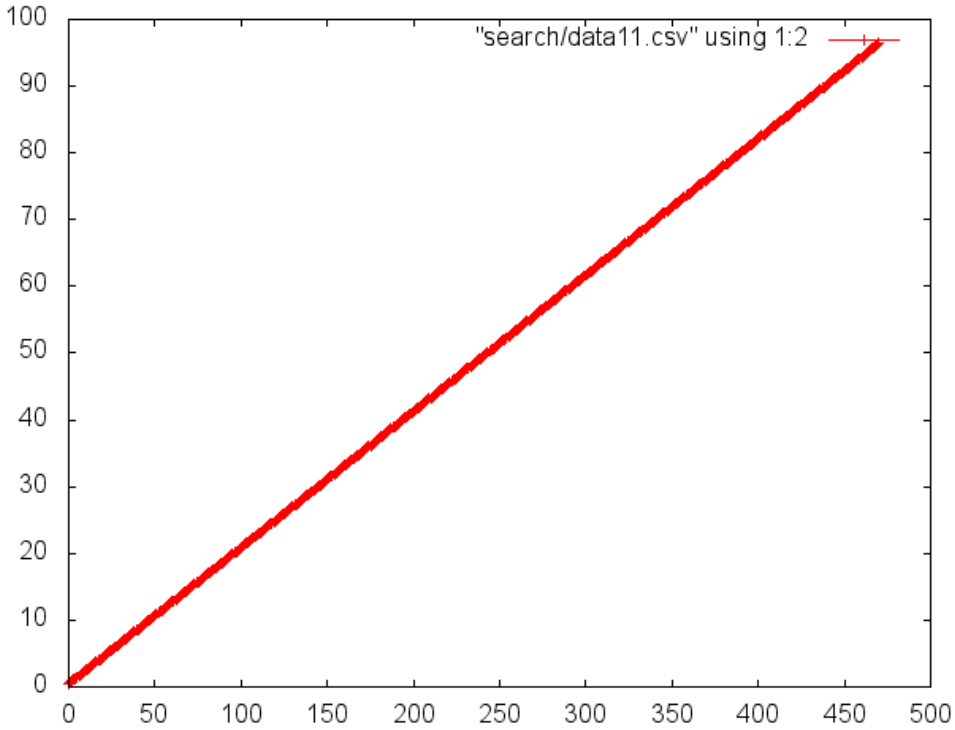
```
...
a0 = x0 / n
a1 = x1 / n

rms0 = 0
rms1 = 1
File.foreach("../data/data_two_1.csv") do |line|
  data = line.strip.split
  rms0 += (data[0].to_f - a0) ** 2
  rms1 += (data[1].to_f - a1) ** 2
  n += 1
end
rms0 = (rms0 / n) ** 0.5
rms1 = (rms1 / n) ** 0.5
puts "Prvni sloupec: RMS #{rms0}"
puts "Druhy sloupec: RMS #{rms1}"
```

Zadání

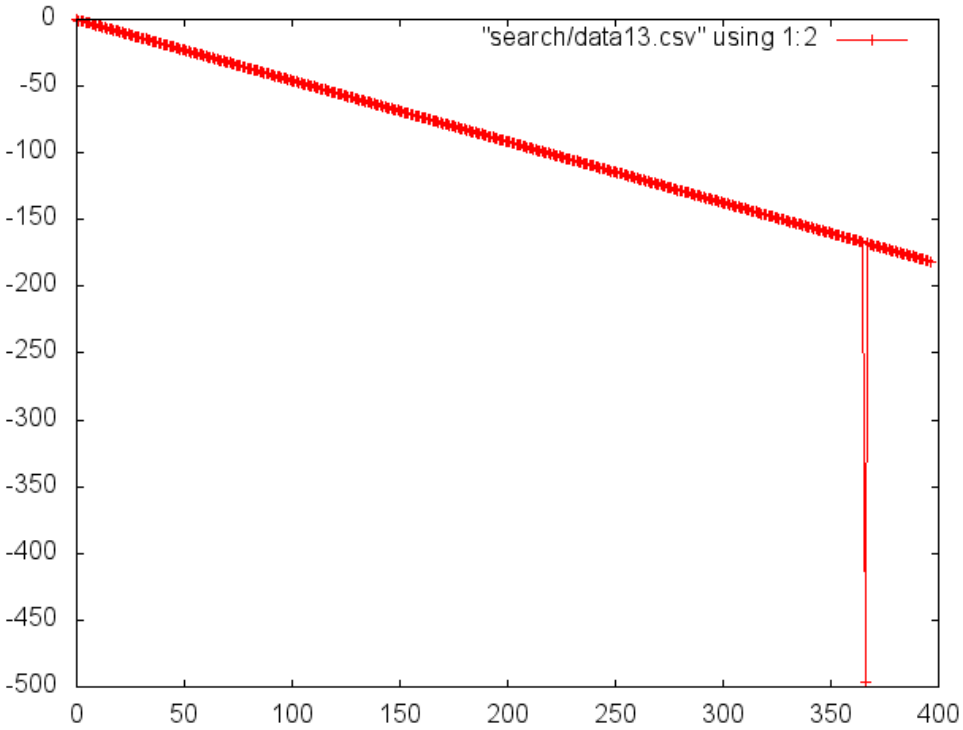
2

Adresář plný CSV souborů (stovky souborů) obsahuje data, která jsou záznamy signálů s lineární závislostí.
V pěti z nich jsou ale poruchy - data ležící zcela mimo přímku.
Kde?

Příklad - dobrý signál



Příklad - špatný signál



Řešení

- stačí vykreslit grafy pro všechny
- `Dir` pro najití souborů
- připravit a spustit `gnuplot`

Znovu a lépe

- pořád je to ještě spousta práce; navíc co když bude souborů tisíckrát víc?
- nabízí se několik řešení, od těžkopádných a robustních (LLS) přes chytré (selská regrese) až po jednoduché (detekce delta-y)
- hurá do toho, už je to jenom práce a skvělé cvičení

Obsah

1 Co jsme se naučili minule

2 Úvod do jazyka Ruby

3 Rozšíření Ruby: RubyGems a Bundler

4 Vytvoříme excelovskou tabulku

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
- otestume bundler: `bundle -v`
- pokud bundler není, doinstalujeme `gem install bundler`

Jak na to

- najdu si, která knihovna mě zajímá (třeba na 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 --path vendor/bundle` (ten parametr stačí poprvé, bundler si to pak pamatuje v konfiguráku `bundle/.config`)

Jak použít?

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

Obsah

1 Co jsme se naučili minule

2 Úvod do jazyka Ruby

3 Rozšíření Ruby: RubyGems a Bundler

4 Vytvoříme excelovskou tabulku

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
- naoprvé navedu do začátku:

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

Jednoduché cvičení

- použijte soubor `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

A to je vše, přátelé!