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 2013/2014
22. října 2013

Obsah

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

Obsah

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
```

Hrátky s řetězci

Délka řetězce

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

Hrátky s řetězci

Délka řetězce

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

Ořez mezer

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

Hrátky s řetězci

Hledání

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

Hrátky s řetězci

Hledání

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

Nahrazení

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

Dokumentace

Google is your friend

```
ruby api string
```

API dokumentace

```
http://www.ruby-doc.org/core-1.9.3/String.html
```

Ztracen v poli

Literál, přiřazení

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

Ztracen v poli

Literál, přiřazení

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

Délka, řazení, vypletí, převracení

```
[4, 2, 6].sort
[2, 5, 3, 3, 4, 1, 2, 1].uniq.sort
[4, 2, 6].reverse
```

Ztracen v poli

Indexace

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

Ztracen v poli

Indexace

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

Do mínusu, odkud kam

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

Pole z řetězů

Řetězec, pole znaků

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

Pole z řetězů

Řetězec, pole znaků

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

Leccos funguje!

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

Sekáček na maso

```
"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"] * ","
```

Převádět přes ulici

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

Převádět přes ulici

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

Hash / slovník

```
a = {}
a["xyz"] = 555
b = {'a' => 4, 'c' => 6}
```

Vocaď pocaď

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

Vocaď pocaď

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

Symbolika

```
"letadlo"
:letadlo
```

Boolean nebolí

Jednoduchá porovnání

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

(pozor na = versus ==)

Boolean nebolí

Jednoduchá porovnání

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

(pozor na = versus ==)

Chytré metody

```
[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 z účtu

Tiskem

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

Výpis z účtu

Tiskem

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

Inspektor Clouseau

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

Cyklistika

Jednoduchý rozsah

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

Cyklistika

Jednoduchý rozsah

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

S polem a proměnnou

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

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 ...)
- metodou Erathostenova síta nalezněte prvočísla menší než N

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 / jiné 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
  c = a + b
  puts a
  a = b
  b = c
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
```

Česko čte dětem

Šikovný iterátor

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

Česko čte dětem

Šikovný iterátor

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

V kuse

```
string = IO.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
IO.foreach("../data/text_1.txt") do |line|
  n += 1
  n_kapr += 1 if line.include?("kapr")
  nn_kapr += line.count("kapr")
end

nn_kapr_bis = IO.read("../data/text_1.txt").count("kapr")

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|
  IO.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|
  IO.foreach("../data/data_two_1.csv") do |line|
    f.puts line.strip.split[0].to_f + line.strip.split[1].to_f
  end
end
```

Úlohy

CSV(3) / řešení

```
x0 = 0
x1 = 0
n = 0
IO.foreach("../data/data_two_1.csv") do |line|
  x0 += line.strip.split[0].to_f
  x1 += line.strip.split[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
IO.foreach("../data/data_two_1.csv") do |line|
  rms0 += (line.strip.split[0].to_f - a0) ** 2
  rms1 += (line.strip.split[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}"
```

A to je vše, přátelé!