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 2012/2013
11. prosince 2012

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
```

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)

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

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")
```

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)

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

Obsah

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

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
5. buď zpracovat ručně (Excel), nebo být Myšpulín a vyrobit skript (úkol s hvězdičkou)

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

```
.....
         the k(trk length) cycle values appear normally distributed at the 95 percent confide
------------------------------------------------------------------------------------------
|
| the final estimated combined collision/absorption/track-length keff = 1.00353 with an estima
|
| the estimated 68, 95, & 99 percent keff confidence intervals are 1.00329 to 1.00377, 1.0030
|
| the final combined (col/abs/tl) prompt removal lifetime = 1.0017E-04 seconds with an estimat
.....
```

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

IO.foreach("c1_1o") do |line|

end

puts keff
```

Realizace (2/5)

```
keff = nil

IO.foreach("c1_1o") do |line|
  if line.include?("final estimated combined")

  end
end

puts keff
```

Realizace (3/5)

```
keff = nil

IO.foreach("c1_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")

  end
end

puts keff
```

Realizace (4/5)

```
keff = nil

IO.foreach("c1_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

IO.foreach("c1_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")
    b = a[1].split
    keff = b[0]
  end
end

puts keff
```

Určení poloh tyčí

Ve vstupním souboru si najdeme relevantní část:

```
c -------------------------------
c polohy tyci (z-plochy)
c -------------------------------
c
67 pz 47.6000    $ dolni hranice absoberu r1
68 pz 40.4980    $ dolni hranice hlavice r1
69 pz 44.8000    $ dolni hranice absoberu r2
70 pz 37.6980    $ dolni hranice hlavice r2
```

Výroba šablon

Jak dostat polohy tyčí do vstupního souboru? Vyrobíme šablonu, tzn nahradíme

```
67 pz 47.6000     $ dolni hranice absoberu r1
```

Výroba šablon

Jak dostat polohy tyčí do vstupního souboru? Vyrobíme šablonu, tzn nahradíme

```
67 pz 47.6000     $ dolni hranice absoberu r1
```

nějakou značkou (*placeholder*):

```
67 pz %r1%     $ dolni hranice absoberu r1
```

Chytáky a zádrhele

- kromě samotné plochy konce absorbéru je nutno správně umístit i z-plochu konce hlavice o 7,102 cm níže
- obecně je na místě ohlídat si, že placeholder nebude kolidovat s ničím jiným

Doporučené nástroje jsou:

- již známá funkce `sub` pro nahrazení jednoho řetězce jiným
- pro pragmatické lenochy funkce `IO.read` načítající celý soubor do řetězce (na což nelze v Pascalu ani pomyslet)
- možno ovšem použít i `IO.readlines` (v čem je to lepší?)

Realizace

```
DELTA = 44.8000 - 37.6980

template = IO.read('template')
(0..10).each do |i1|
  (0..10).each do |i2|
    r1 = i1 * 50
    r2 = i2 * 50
    File.open("inputs/c_#{i1}_#{i2}", "w") do |f|
      s = template.sub("%r1%", r1.to_s)
      s = s.sub("%r1_%", (r1 - DELTA).to_s)
      s = s.sub("%r2%", r2.to_s)
      s = s.sub("%r2_%", (r2 - DELTA).to_s)
      f.puts template
    end
  end
end
```

Jak na to

Máme všechno, co potřebujeme:

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

Takže už to stačí jen vhodným způsobem spojit dohromady!

Realizace

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

  IO.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 funkce `to_i` (co asi dělá?)

Realizace chytré horákyně

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

  IO.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
```

A to je vše, přátelé!