Informatika pro moderní fyziky (6) výstupní a vstupní soubory pro výpočetní programy, základy OOP

František HAVLŮJ

e-mail: haf@ujv.cz

ÚJV Řež
oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2022/2023, 2. listopady 2022

Obsah

- práci s datovými strukturami (zejm. hash)
- opakování tvorby grafů

Obsah

Co kdybychom využili příležitost a naučili se trochu OOP

- když už jsme se pustili do definice vlastních funkcí a použití `require` ...
- ... zkusíme se rovnou posunout o level dál a budeme používat objektově orientované postupy

```
f = MCNPOutput.new("c1_1o")
puts f.keff
puts f.keff_uncertainty
```

Obsah

Co je OOP (1)

- zatím jsme používali tzv. procedurální programování – máme data a pak procedury/funkce
- Ruby je nicméně čistě objektový jazyk, i když jsme se tomu zatím spíš vyhýbali
- objekt je entita, která má *vlastnosti* (properties) a *metody* (methods) a poskytuje okolnímu světu nějaké *rozhraní* (interface)

Co je OOP (2)

- většinou se jako hlavní důvody pro OOP uvádí polymorfismus a dědičnost (inheritance), ale to hlavní je posun uvažování od dat a operací nad nimi k “inteligentním” objektům – a spolu s tím zapouzdření (encapsulation)
- je potřeba si uvědomit, že všechna ‘zázračná’ paradigmata v programování jsou pouze odlišné formalismy, může nám to hodně pomoct a stojí za to se tomu věnovat, na druhou stranu je potřeba se z toho nezbláznit a nedělat z toho náboženství

Třída

- ‘typ’ objektu - definuju, jak se objekty chovají
- neboli definuju metody
- zjednodušeně řečeno jsou to jen metody a ne data
- potkali jsme třeba `File`, `CSV` (a aniž bychom to věděli, tak `Fixnum`, `Array`, `Hash`)

Instance

- konkrétní objekt – má svoje data
- většinou vytváříme pomocí `Class.new`, resp `Class.new(arg1, arg2, ...)`
- k datům přistupuju pomocí *instance variables* – `@data` – ta nikdo jiný zvenku nevidí
- můžu definovat `attr_reader`, `attr_writer`, `attr_accessor` pro přístup k instance variables zvenčí
- metody se definují podobně jako normální funkce a volají se obvyklým způsobem `obj.method`
- leckdy se hodí speciální metoda `initialize`, tzv. *konstruktor* – volá se při `new` a nastavují se zde výchozí hodnoty vlastností

Class method

- můžu mít i metody, které nepatří k žádné instanci – nepracují s žádnými daty
- dávat je do třídy má pak smysl pouze organizační, nemá to pak žádnou reálnou výhodu proti obyčejným funkcím
- např `File.foreach` nebo `CSV.read`

Jak to vypadá v Ruby

```
class Table
  def initialize
    @data = {}
  end
  def get(key)
    @data[key]
  end
  def set(key, value)
    @data[key] = value
  end
  def print
    @data.each do |key, value|
      puts "#{key} #{value}"
    end
  end
end

t = Table.new
t.set("a", 123)
t.print
```

Úkol: implementujte třídu MCNPFile z příkladu

```
f = MCNPOutput.new("c1_1o")
puts f.keff
puts f.keff_uncertainty
```

Obsah

HELIOS

Tabulka výstupů:

```
List name         : list
List Title(s)  1) This is a table
               2) of some data
               3) in many columns
               4) and has a long title!
```

	bup	kinf	ab	ab	u235	
0001	0.00E+00	1.16949	9.7053E-03	7.6469E-02	1.8806E-04	7.
0002	0.00E+00	1.13213	9.7478E-03	7.9058E-02	1.8806E-04	7.
0003	1.00E+01	1.13149	9.7488E-03	7.9070E-02	1.8797E-04	7.
0004	5.00E+01	1.13004	9.7521E-03	7.9093E-02	1.8760E-04	7.
0005	1.00E+02	1.12826	9.7559E-03	7.9218E-02	1.8714E-04	7.
0006	1.50E+02	1.12664	9.7594E-03	7.9407E-02	1.8668E-04	7.
0007	2.50E+02	1.12399	9.7657E-03	7.9869E-02	1.8577E-04	7.
0008	5.00E+02	1.12007	9.7812E-03	8.1065E-02	1.8351E-04	7.
0009	1.00E+03	1.11561	9.8203E-03	8.3169E-02	1.7914E-04	7.
0010	2.00E+03	1.10542	9.9329E-03	8.6731E-02	1.7088E-04	7.
0011	3.00E+03	1.09354	1.0067E-02	8.9717E-02	1.6316E-04	7.

Co bychom chtěli

- mít načtené jednotlivé tabulky (zatím jen jednu, ale bude jich víc)
- asi po jednotlivých sloupcích, sloupec = pole (hodnot po řádcích)
- sloupce se nějak jmenují, tedy použijeme `Hash`
- `table["kinf"]`
- pozor na `ab`, asi budeme muset vyrobit něco jako `ab1`, `ab2` (ale to až za chvíli)

Nástrahy, chytáky a podobně

- tabulka skládající se z více bloků
- více tabulek
- tabulky mají jméno `list name` a popisek `list title(s)`

Jak uspořádat data?

- pole s tabulkami + pole s názvy + pole s titulky?

Jak uspořádat data?

- pole s tabulkami + pole s názvy + pole s titulky?
- co hashe tabulky[název] a titulky[název]?

Jak uspořádat data?

- pole s tabulkami + pole s názvy + pole s titulky?
- co hashe tabulky[název] a titulky[název]?
- nejchytřeji: `{"a"=>{:title => "Table title", :data => {"kinf"=>...}}}`

Jak uspořádat data?

- pole s tabulkami + pole s názvy + pole s titulky?
- co hashe tabulky[název] a titulky[název]?
- nejchytřeji: `{"a"=>{:title => "Table title", :data => {"kinf"=>...}}}`
- "nová" syntaxe: `{"a"=>{title: "Table title", data: {"kinf"=>...}}}`

Z příkazové řádky

- a co takhle z toho udělat skript, který lze pustit s argumentem = univerzální
- `ruby read_helios.rb helios1.out`
- vypíše seznam všech tabulek, seznam jejich sloupců, počet řádků
- pole `ARGV` – seznam všech argumentů
- vylepšení – provede pro všechny zadané soubory: `ruby read_helios.rb helios1.out helios2.out` (tip: využívejte vlastní metody, kde to jen jde)

Obsah

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 `File.read` načítající celý soubor do řetězce
- možno ovšem použít i `File.readlines` (v čem je to lepší?)

Realizace

```
delta = 44.8000 - 37.6980

template = File.read("template")
(0..10).each do |i1|
  (0..10).each do |i2|
    r1 = i1 * 50
    r2 = i2 * 50
    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)
    File.write("inputs/c_#{i1}_#{i2}", s)
  end
end
```

Obsah

A co takhle trocha zobecnění?

- když budu chtít přidat další tyče nebo jiné parametry, bude to děsně bobtnat
- funkce `process("template", "inputs/c_#{i1}_#{i2}", {"r1" => r1, "r2" => r2, .....})`
- všechno víme, známe, umíme...
- rozšiřte tak, že třeba tyč B1 bude mezi R1 a R2, B2 mezi R1 a B1, B3 mezi R2 a dolní hranicí palivového článku (Z = 1 cm)

A to je vše, přátelé!