Solutions for
CSC 372 Ruby Mid-term Exam
April 11, 2014

Problem 1: (21 points) Mean and median: 17.6, 19

In this problem you are to write a Ruby program `tail.rb` *that prints the last N lines of standard input, just like the UNIX* `tail` *command. ...*

The solution I wrote when timing myself was this:

```
if ARGV.length != 1 || (not ARGV[0] =~ /^-\d+$/)
    puts "Usage: tail -N"
    exit 1
end

n = -ARGV[0].to_i

lines = []
while line = STDIN.gets
    lines << line
end

if n > lines.size
    n = lines.size
end

(-n).upto(-1) { |i| puts lines[i] }
```

With some time to think and some good ideas from student solutions, I offer this simpler version:

```
if ARGV.length != 1 || (not ARGV[0] =~ /^-\d+$/)
    puts "Usage: tail -N"
    exit 1
end

n = -ARGV[0].to_i

lines = STDIN.readlines

n = lines.size if n > lines.size

puts lines[-n..-1]
```

Problem 2: (17 points) **Mean and median: 13.5, 15.5**

In this problem you are to write a method `load_facts(file_name)` *that reads Prolog facts from the specified file and returns a Ruby hash that holds a representation of the facts.*

```
def load_facts(file_name)
    facts = Hash.new []

    f = File.open(file_name)

    while line = f.gets
        line.chomp!
        if line = ~ /^(\w+)\((.*)\)\.$/
            facts[$1] += [$2.split(",")]
        end
    end

    f.close
    facts
end
```

The combination of initializing the hash facts with `Hash.new []` and using += allows avoiding a check to see whether `facts[$1]` already exists. (Using << instead of += would break it. Try it and be the first to explain why on Piazza!)

Because the input can be assumed to be valid, only processing lines that start with `/\w/` suffices to ignore empty and blank lines.

Common mistakes, shown as variations of the above, were these:

```
facts[$1] = $2.split(",")      # Doesn't build an array of arrays

facts[$1] += [$2]              # Doesn't split
```

Some students used various splits but here's the good one:

```
parts = "food(just,a,test,here)".split(/[,()]/)
```

The functor ends up in `parts[0]`; the terms are in `parts[1..-1]`.

Let's combine that with a variation of parallel assignment that we didn't study but that is similar to the mechanism for handling varying numbers of arguments:

```
>> head, *tail = "food(just,a,test,here)".split(/[,()]/)

>> head   => "food"
>> tail   => ["just", "a", "test", "here"]
```

Problem 3: (15 points) <u>**Mean and median: 9.3, 10.5**</u>

This problem is a partner to `load_facts`*. You are to write a Ruby method* `print_preds` *that takes the hash produced by* `load_facts` *(the previous problem) and prints a list of predicate indicators.*

My draft solution is below but Dennis noticed a bug in it. Cover the portion of the page the follows the code and play exam grader for a minute to see if you see it!

```
def print_preds h
    for k in h.keys.sort
        lens = h[k].collect {|a| a.size }
        for n in (lens.min)..(lens.max)
            puts "#{k}/#{n}"
        end
    end
    nil
end
```

The problem is that it assumes a contiguous range of "arities". For example, given `x(a).` and `x(a,b,c).`, it'll output `x/1`, `x/2` (oops!), and `x/3`. That was true with the sample data I provided but not true in general.

If had recognized the potential of noncontiguous arities, I would have mentioned `Array#uniq`, which is similar to UNIX's `uniq` command, discarding non-unique values:

```
>> [3,1,2,3,1,3].uniq    => [3, 1, 2]
```

`uniq` enables this concise solution:

```
def print_preds h
    for k in h.keys.sort
        for n in lens = h[k].collect {|a| a.size }.uniq.sort
            puts "#{k}/#{n}"
        end
    end
    nil
end
```

Without `uniq` you need some other way to compute the set of arities. The first thing that comes to mind based on what you've seen is using the `|` operator to form the union of two arrays:

```
>> [3,1]|[1]|[2] => [3, 1, 2]
```

The keys of a hash also constitute a set. A couple of students took that observation further and used the whole predicate indicator, like "`food/1`", as the key! Here's a solution based on that idea:

```
def print_preds h
    indicators = {}
    h.each { |k, arrays|
        arrays.each { |e| indicators["#{k}/#{e.size}"] = 1 }
        }
    puts indicators.keys.sort
    nil
end
```

There's a slight bug with that because, for example, "`t/10`" collates before "`t/2`". With these facts,

```
t(x).
t(x,x).
t(x,x,x,x,x,x,x,x,x,x).
```

you'll this output:

```
t/1
t/10
t/2
```

Also, I never mentioned it in lecture but Ruby does have a `Set` class:

```
>> require 'set'          => true

>> [3,1,3,4,2,1].to_set  => #<Set: {3, 1, 4, 2}>
```

I apologize for the unintended difficulty due to my error when writing my own solution, but everybody was in the same boat.

Problem 4: (5 points) **<u>Mean and median: 4.0, 4.5</u>**

Write a method `multstring(spec, s)` that behaves like this:

```
>> multstring("3,1,5", "x")
=> "xxx,x,xxxxx"
```

Solution:

```
def multstring(spec,s)
    spec.split(",").map {|e| s*e.to_i} * ","
end
```

Problem 5: (4 points) <u>Mean and median: 2.4, 2.5</u>

Write a method `addrev` *such that after calling* `addrev`*, the unary minus operator applicatively reverses a string.*

```
def addrev
    eval("class String; def -@; self.reverse; end; end")
end
```

I didn't expect students to know it but that class declaration must be wrapped in an `eval(...)`. The error "`class definition in method body`" is produced by this version:

```
def addrev
    class String
          def -@; self.reverse; end; end
end
```

Problem 6: (11 points) (One point each unless otherwise indicated) **<u>Mean and median: 7.2, 8.5</u>**

(a) *Write a regular expression that is equivalent to* `/x+/` *but that doesn't use the* `+` *operator.*

```
/xx*/
```

(b) *Write a regular expression that is equivlent to* `/ax|bx|cx/` *but that doesn't use the | operator.*

```
/[abc]x/
```

(c) *Describe in English the strings matched by* `/a*b+c$/`

Strings containing zero or more "`a`"s immediately followed by one or more "`b`"s immediately followed by a "`c`" at the end of the string.

(d) *(3 points) Write a regular expression that matches only strings that are binary constants like these:*

```
"0b0"
"0B11111"
"0b01010101"
```

The first two characters are a "`0`" and an upper- or lower-case "`B`". One or more "`1`"s or "`0`"s follow.

Here are two <u>non</u>-matches: "`00b1`" (extra leading zero), "`0b12`" (trailing non-0/1).

```
/^0[bB][01]+$/
```

(e) *(5 points) Write a regular expression that matches a string if and only if it is a comma-separated sequence of integers **<u>greater than or equal to 10</u>**. Examples of matches:*

```
"10, 11, 12"
```

```
"30,200,500"
"100"
```

One optional space may appear after each comma but that is the only place a space may appear. ***Leading zeros are not permitted!***

Here are three non-matches: "`10, 020`" (has a leading zero), "`7,11`" (7 is less than 10), "`1x`" (has a trailing "`x`").

```
/^[1-9][0-9]+(, ?[1-9][0-9]+)*$/
```

Problem 7: (7 points) Mean and median: 4.7, 6

Write an iterator `take_while(a)` that calls its block with each element of the array `a` in turn. As long as the block returns true, array elements are accumulated. When the block first returns a non-true value or when the array elements are exhausted, an array of the accumulated elements is returned.

```
def take_while a
    r = []
    a.each { |e|
        if yield e
            r << e
        else
            return r
        end
        }
    return r
end
```

Problem 8: (12 points) Mean and median: 7.9, 8.5

(a) *Who invented Ruby? (1 point)*

"Matz" is the answer I was hoping for, but "some Japanese guy" was counted as correct, too. Even "Some whack-a-doo! Monkey patching?!? really?" got a point!

(b) *Consider a Ruby method `last(x)` that returns the last element of a string or array. Why would `last` be awkward to write and use in Java? (2 points)*

Quite a few students pointed out that you'd need to do length-1 to compute the position of the last element but I was looking for something a little deeper.

I wish I'd stipulated "without overloading" but since I didn't, pointing out that you'd need overloaded methods was counted as correct. However, `char last(String s)` is easy enough but how about the array case? `int last(int a[])` only covers `int`s, of course. Can it be done with generics?

(c) *What's the effect of adding "`include Enumerable`" to a class definition? (2 points)*

Assuming that the class implements `each`, including `Enumerable` gives your class all the methods in `Enumerable`, as you saw with `XString`.

(d) *Cite one way in which Ruby's `if-then-else` is like Haskell's `if-then-else` and one way in which it is different. (2 points)*

In both Haskell and Ruby `if-then-else` is an expression, producing a value. Haskell requires an `else` clause but Ruby does not.

(e) *A Java newcomer to Ruby might think that `private` and `attr_reader` are keywords but they aren't. What are they? (2 points)*

Methods

(f) *What's the essential difference between dynamic typing and static typing? (3 points)*
Hint: If your answer contains `/(interprete|compile)[rd]?/` there's a fair chance it'll be wrong.

In my mind the essential difference is that with static typing you can detect a particular class of type errors without executing the code but with dynamic typing you cannot.

A number of students had a successful answer that included *`/(interprete|compile)[rd]?/`* and that was fine but I suggested not using that because I feared answers like "Dynamically typed languages must be interpreted and statically typed languages must be compiled.", which is completely wrong!

Problem 9: (8 points) Mean and median: 4.8, 6

In this problem you are to create a subclass of `Shape` named `Ring`. An instance of `Ring` represents the region between two concentric circles (the black area in the drawing to the right). Along with a label that is passed to `Shape`'s constructor, `Ring`'s constructor takes the radii of the two circles but they may specified be in either order. If the two circles have the same radius, the area of the ring is zero.

Your implementation of `Ring` should have five methods: a constructor, `area`, `inner`, `outer`, and `inspect`. The methods `area`, `inner`, and `outer` return the area of the ring, the inner radius and the outer radius, respectively. `inspect` displays the inner (smaller) radius followed by the outer radius.

```
class Ring < Shape
    def initialize(label, r1, r2)
        super(label)
        if r1 < r2
            r1, r2 = r1, r2
        end
        @outer, @inner = r1, r2
    end

    attr_reader :inner, :outer
```

```
          def area
              @outer**2*Math::PI - @inner**2*Math::PI
          end

          def to_s
              "Ring #{label} (#{@inner}-#{@outer})"
          end
      end
```

Extra Credit Section (½ point each unless otherwise noted) Mean and median: 3.0, 3.25

(a) *What does whm consider to be the all-time greatest Original Thought ever put forth by one of his 372 students?*

Joe Astier, in my Fall 1996 class, likened a Prolog predicate to a motor: If you apply current to a motor, it turns the rotor; if you turn the rotor, current is produced.

Technical note: My understanding is that it must be a DC motor with a magnet.

(b) *Based on the Prolog we've covered, is Prolog statically typed or dynamically typed? (And why?)*

Dynamically typed

(c) *What thing in Ruby is closest to a Prolog atom?*

Symbols

(d) Which is the oldest of Java, Haskell, and Ruby?

Haskell

(e) *The term "mixin" came from a business that sold what to college students?*

The business was Steve's Ice Cream but I imagine that Steve sold other stuff, too.

(f) *Regular expressions are Type* 4 *languages in the* Chomsky *hierarchy of languages.*

(g) *Predict the median score on this exam. (The median is the "middle" value in a range of values.)*

Here are the predictions: (mean = 74.4, mode = 75, median = 75)

```
50, 50, 60, 65, 65, 67, 67, 70, 70, 70, 70, 70, 72, 72, 72, 72.71,
74, 74, 75, 75, 75, 75, 75, 75, 78, 78, 79, 79, 80, 80, 81, 82,
82, 82, 83, 83, 84, 84, 84.726953281, 85, 85
```

Here are the actual scores:

```
104, 103, 100.5, 100.5, 99, 98.5, 96, 96, 94.5, 93, 91.5, 91.5,
90, 90, 89, 89, 88.5, 88, 88, 86, 86, 85.5, 84, 83.5, 82.5, 82,
81.5, 79, 79, 78.5, 78.5, 78.5, 78.5, 78, 78, 78, 77, 75, 73.5,
70.5, 69.5, 69.5, 69, 68, 66.5, 65.5, 64.5, 63, 62.5, 60.5, 57,
53, 48.5, 44, 38.5, 38, 31.5, 28.5, 15.5
```

Simple statistics:

```
N = 59
mean = 75.8729
median = 78.5
```

If I had more time I'd compile the interesting answers from the following but alas I don't have that time to spare at the moment. If time permits, I'll issue an update.

One man's trash is another man's treasure so as you'd expect, what seemed stupidest to some, like monkey patching, seemed best to others.

(h) *If you only remember one thing about Ruby, what will it be? (Ok to be funny!)*

(i) *What's the stupidest thing in Ruby? Can't be the same as (h) or (j). (1 point)*

(j) *What's the best thing in Ruby? Can't be the same as (h) or (i). (1 point)*

(k) *Implement* `attr_reader`*. (1 point)*

```
def attr_reader *attrs
    attrs.each { |attr|
        eval("def #{attr}; @#{attr}; end")
        }
end
```

The real `attr_reader` is a method of `Module` but the above works as-is.