Ruby in the context of scientific computing

Shahrooz Afsharipour

16 January 2014

Overview

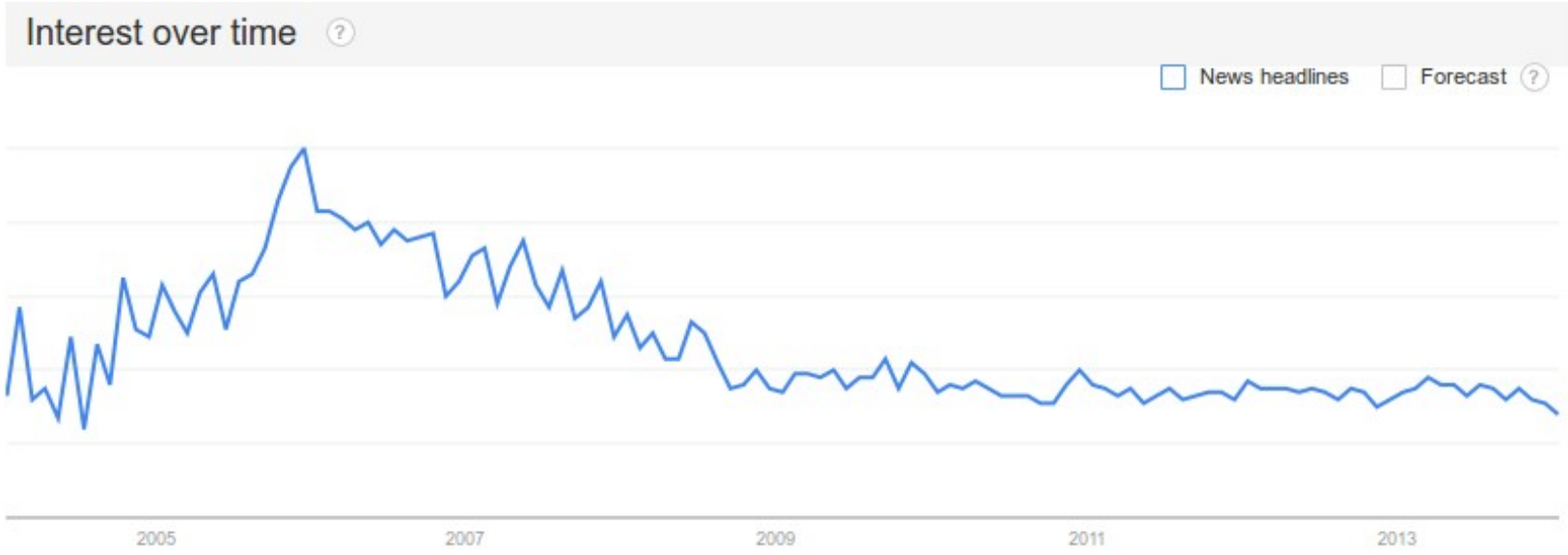
- Introduction
- Characteristics and Features
  - Closures
- Ruby and Scientific Computing
  - SciRuby
  - Bioruby
- Conclusion
- References

Introduction

Ruby

A Programmer's Best Friend

- Created by Yukihiro "Matz" Matsumoto
- First public release in 1995
- Specification published in 2010
- Was mostly popularized by Ruby on Rails (a web framework)



Google searches for "Ruby programming" according to Google Trends

Introduction

- Main objective: to make developers happy :)

  “The biggest goal of Ruby is developer friendliness, and productivity of application development and intuitive description of program behaviors take precedence over brevity of the language specification itself and ease of implementation.” - Specification document

- RubyGems: package management platform for sharing Ruby programs and libraries

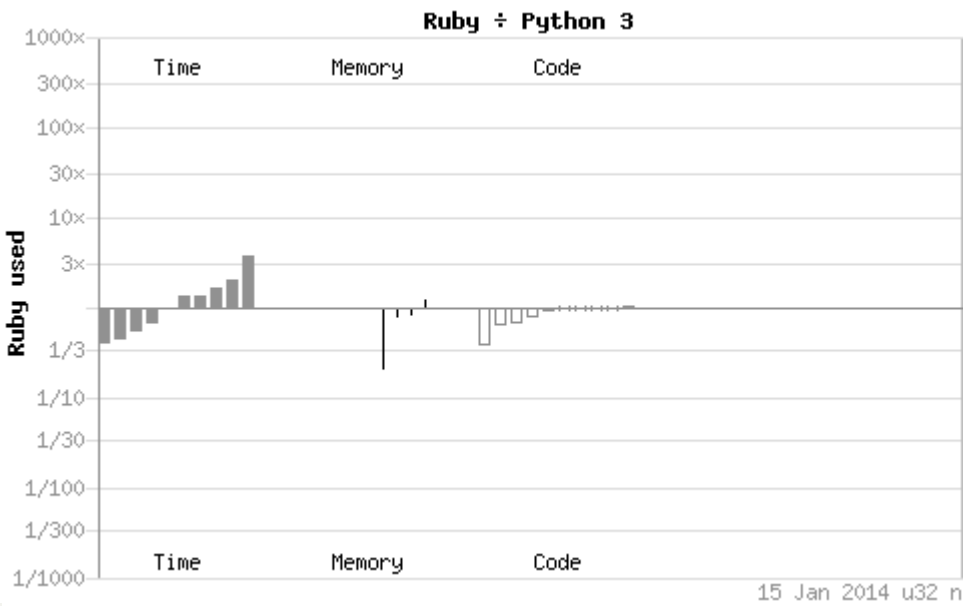
```
gem install myGem
```

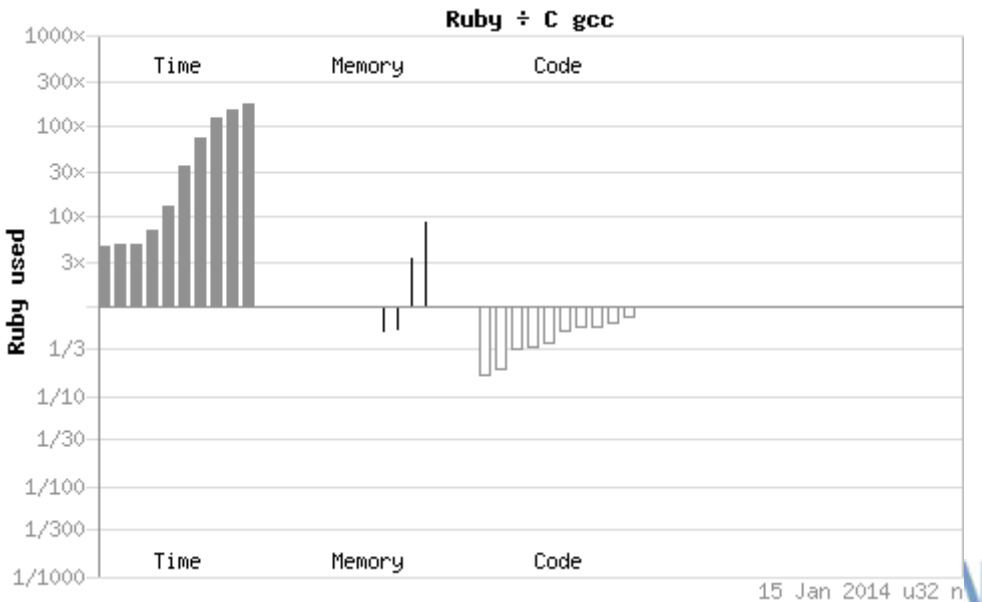
Implementations

- Many implementations: Ruby MRI (CRuby), YARV, JRuby, Rubinius, IronRuby, Ruby .NET, etc.
- Ruby MRI: *Matz's Ruby Interpreter*, also known as CRuby
  - Reference implementation
- YARV merged with MRI since 1.9
- This presentation's focus: Ruby MRI
  - Many plugins and gems might not work with other implementations

Performance

- Interpreted, dynamically typed
- Perforamce in the same order of magnitude as Python and PHP
- ...and not as good as compiled, statically-typed languages (C/C++, Java, C#, etc.)





Programming paradigms

- Supports multiple paradigms
  - Procedural: Procedural-style code outside classes is in the Object class scope
  - Object Oriented: (Almost) everything is an object
    - Even classes
    - E.g., binary operators are just syntactic sugar for method calls: `1 + 2` is the same as `1.+(2)`
  - Functional
    - Closures
    - Higher-order functions
    - Implicit return

Classes and Modules

- Classes can be reopened and modified at any point
  - The same can be done with instance objects

```
class String
  def append_exclamation
    self + "!"
  end
end
```

- No multiple inheritance, no *Interfaces*
- Mixins via Modules

```
module Constants
  PI = 3.1416
  E = 2.7183

  def pi_squared
    return PI ** 2
  end
end
```

```
class Calculator
  include Constants
end
```

Type system

- Dynamic and strong typing
  - Duck typing (“if it walks like a duck an quacks like a duck...”)
  - It is the object's methods and properties, not its class or inheritance status, that determine semantics

```
words = ["This", "is", "a", "sentence"]
if words.respond_to?("join")
  words.join(" ") + "."
end

=> "This is a sentence."
```

Closures

- Associated with functional programming
- (Old) definition from Wikipedia:

  “In computer science, a closure is a first-class function with free variables that are bound in the lexical environment.”

- Closures are functions that
  - can be passed around like objects
  - are bound to the scope where they were created
- Ruby makes heavy use of closures (via Blocks, Procs and Lambdas)

Blocks

- Blocks
  - Enclosed with either {} or do/end

```
fibonacci = [1, 1]

10.times do
  fibonacci << fibonacci[-2] + fibonacci[-1]
end

=> [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

- Methods can be made to accept Blocks as parameters using the *yield* keyword

```
def pass_arg_to_block(arg)
  yield arg
end

pass_arg_to_block(5) { |n| n * 10 }
=> 50
```

Blocks, Procs, Lambdas

- Blocks are not objects, e.g. can't get passed around
  - Solution: Procs and Lambdas

- Lambdas are the same as Procs, but:
  - more strict argument checking
  - different handling of return calls

```
multiply_by_ten = Proc.new { |n| n * 10 }
# OR              = proc     { |n| n * 10 }
# OR              = lambda   { |n| n * 10 }

pass_arg_to_block(5, &multiply_by_ten)
=> 50

[1, 2, 3].map(&multiply_by_ten)
=> [10, 20, 30]
```


RWTH AACHEN UNIVERSITY

Threads

- Green threads in 1.8
- Native threads in 1.9
  - Global Interpreter Lock: only one thread executed at a time
  - No true concurrency
  - Better integrity protection (e.g. many extensions are not thread safe)
  - BUT: doesn't atuomatically stop you from writing thread-unsafe code!
- Other implementations deal with threads differently

Ruby and Scientific Computing

- Due to historical reasons, Python grew into a widely-used language in scientific computing (SciPy/NumPy, matplotlib, etc.), while Ruby didn't
  - Google's promotion of Python
  - Ruby community being mainly focused on Rails

- ...but such a thing as “scientific Ruby” does exist:
  - SciRuby
  - BioRuby

SciRuby

- Collection of libraries for various scientific tasks
- First (Git) commit in 2011
- Might not be ready for mission critical tasks yet

"Word to the wise: These gems have been tested, but are not battle-hardened. If you're thinking of using NMatrix (or other SciRuby components) to write mission critical code, such as for a self-driving car or controlling an ARKYD 100 satellite, you should expect to encounter a few bugs — and be prepared for them." - sciruby.com

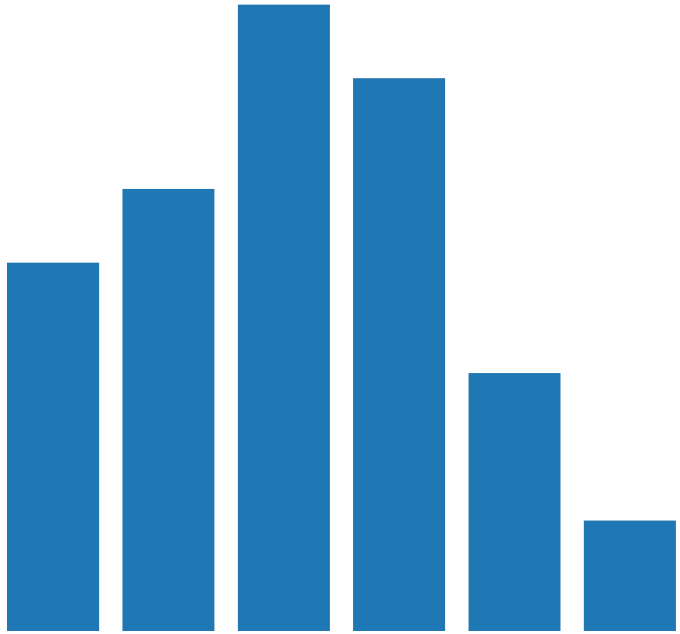
SciRuby

- Visualization
  - Rubyvis
  - (Plotrb?)
- Statistics and probability
  - Statsample
  - Distribution
- Numeric
  - Minimization
  - Integration
  - Nmatrix

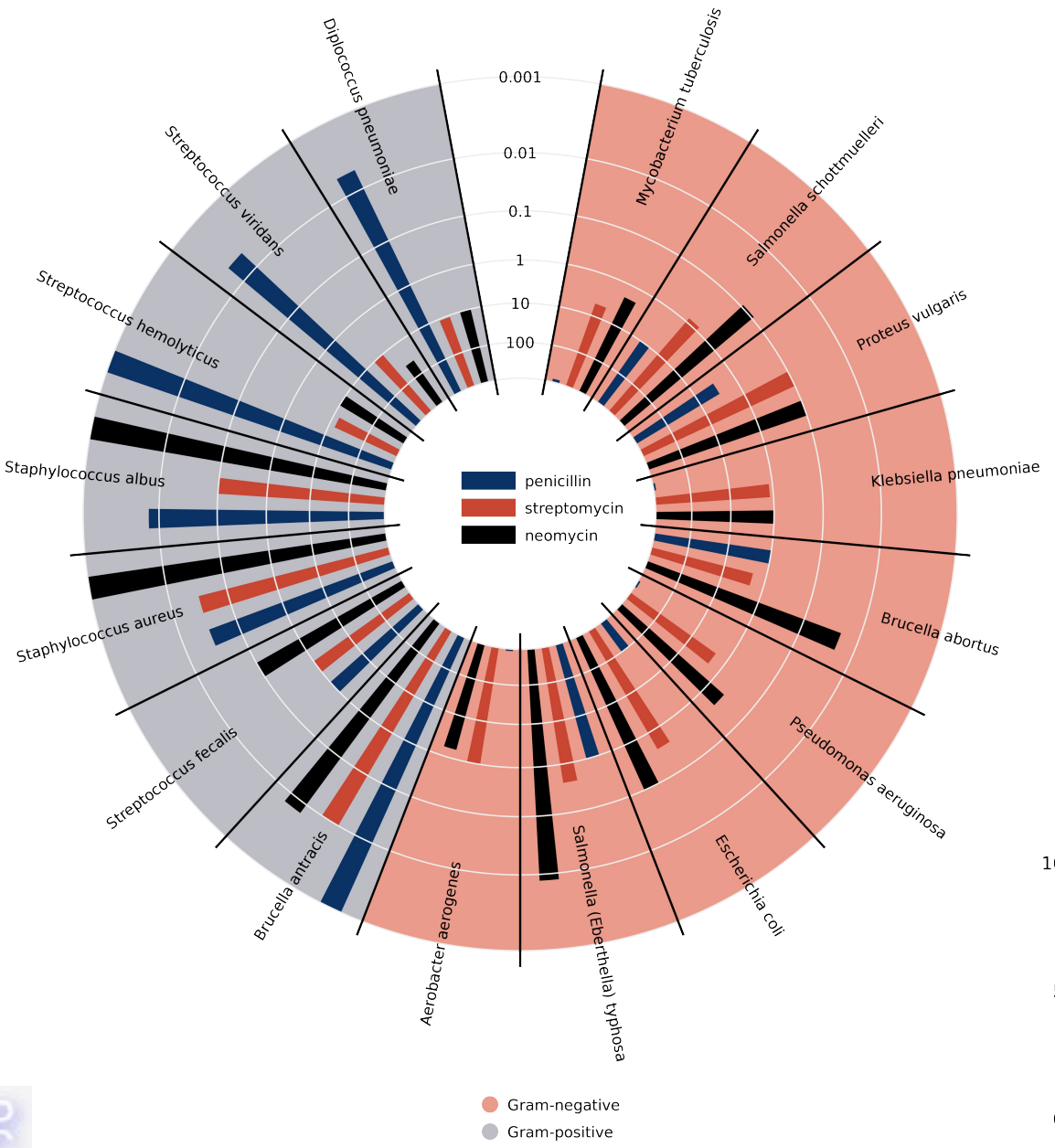
RWTH AACHEN UNIVERSITY

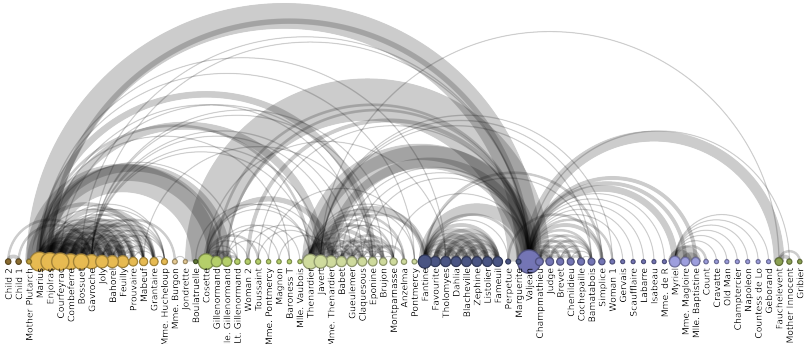
Rubyvis Plotting

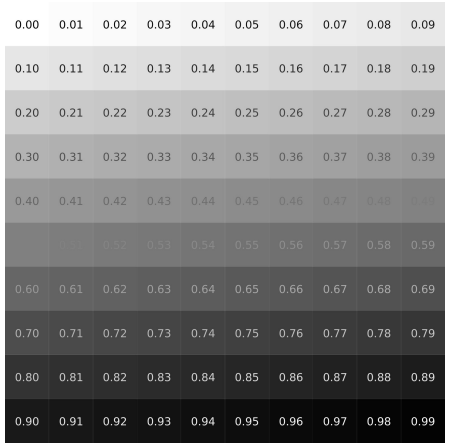
```ruby
require 'rubyvis'

vis = Rubyvis::Panel.new do
  width 150
  height 150
  bar do
    data [1, 1.2, 1.7, 1.5, 0.7, 0.3]
    width 20
    height {|d| d * 80}
    bottom(0)
    left {index * 25}
  end
end

vis.render
outfile = open('rv-out.svg', 'w')
outfile.write (vis.to_svg)
outfile.close()
```

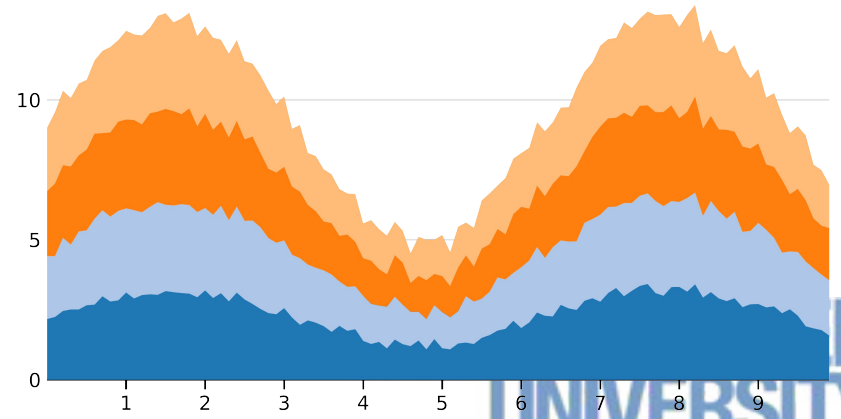


Rubyvis: More Plotting





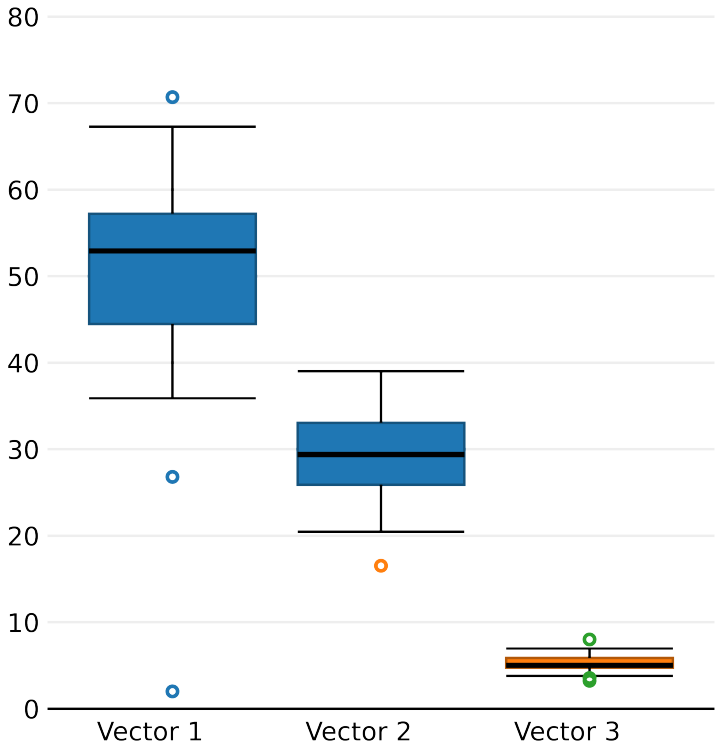




Statsample

- A statistics suite for Ruby
- Sample code:

```
require 'statsample'

ss_analysis(Statsample::Graph::Boxplot) do

  n=30

  a=rnorm(n-1, 50, 10)

  b=rnorm(n, 30, 5)

  c=rnorm(n, 5, 1)

  a.push(2)

  boxplot(:vectors=>[a,b,c],
:width=>300, :height=>300, :groups=>
%w{first first second}, :minimum=>0)

end

Statsample::Analysis.run # Open svg file
on *nix application defined
```



Minimization and Integration

- Minimization

```
require 'minimization'

d = Minimization::Brent.new(-1000,
20000, proc {|x| x**2})

d.iterate

puts d.x_minimum

puts d.f_minimum


# Output:

# 4.005934284325451e-31

# 1.604750949033406e-61
```

- Integration

```
require 'integration'

puts Integration.integrate(1, 2,
{:tolerance=>1e-10,:method=>:simpson})
{|x| x**2}

normal_pdf=lambda {|x|
(1/Math.sqrt(2*Math::PI))*Math.exp(-(x**2/
2))}

puts
Integration.integrate(Integration::MInfini
ty, 0, {:tolerance=>1e-10}, &normal_pdf)

puts Integration.integrate(0,
Integration::Infinity,
{:tolerance=>1e-10}, &normal_pdf)


# Output:

# 2.333333333333333

# 0.5

# 0.5
```

BioRuby

- A project that aims to provide a set of tools for tasks in computationial biology and bioinformatics (sequence analysis, pathway analysis, sequence alignment, etc.)

"Bioinformatics is an interdisciplinary field that develops and improves on methods for storing, retrieving, organizing and analyzing biological data. A major activity in bioinformatics is to develop software tools to generate useful biological knowledge." - Wikipedia

- A major task in bioinformatics is analyzing character sequences. BioRuby helps with this.
- Biogem: Tool that aids bioinformaticians in writing "plugins" and sharing them with the rest of the BioRuby community

Image source: http://www.medicine.uiowa.edu

BioRuby: Example

```
bioruby> seq = Bio::Sequence::NA.new("atgcatgcaaaa")
==> "atgcatgcaaaa"
bioruby> seq.complement
==> "ttttgcatgcat"
bioruby> seq.subseq(3,8) # gets subsequence of positions 3 to 8 (starting from 1)
==> "gcatgc"
bioruby> seq.gc_percent
==> 33
bioruby> seq.composition
==> {"a"=>6, "c"=>2, "g"=>2, "t"=>2}
bioruby> seq.translate # each three nucleotides represent an amino acid
==> "MHAK"
bioruby> seq.translate.codes # (standard) amino acids have abbreviations
==> ["Met", "His", "Ala", "Lys"]
bioruby> seq.translate.names
==> ["methionine", "histidine", "alanine", "lysine"]
bioruby> seq.translate.molecular_weight
==> 485.605
bioruby> counts =
{'a'=>seq.count('a'),'c'=>seq.count('c'),'g'=>seq.count('g'),'t'=>seq.count('t')}
==> {"a"=>6, "c"=>2, "g"=>2, "t"=>2}
bioruby> randomseq = Bio::Sequence::NA.randomize(counts)
==> "aaacatgaagtc"
```

Conclusion

- Feature-rich, developer-friendly multi-paradigm language
- Powerful mixture of object-oriented and functional programming
- Scientific libraries available
- Lower performance but higher programmer productivity vs. compiled/statically-typed languages
  - Good for prototyping?
- Fun to use!

Refrences

- Official Ruby website

  https://www.ruby-lang.org/en/about/
- Ruby documentation

  http://ruby-doc.org/
- Computer Language Benchmarks Game

  http://benchmarksgame.alioth.debian.org
- Closures and Higher-Order Functions

  http://weblog.raganwald.com/2007/01/closures-and-higher-order-functions.html
- Closures – A Simple Explanation (Using Ruby)

  http://www.skorks.com/2010/05/closures-a-simple-explanation-using-ruby/
- SciRuby

  http://sciruby.com/
- BioRuby

  http://bioruby.open-bio.org/