# バイオインフォマティクス演習
# スクリプトプログラミング（２）

## Rubyのライブラリ->BioRuby

東京大学医科学研究所
ヒトゲノム解析センター
ゲノムデータベース分野
片山俊明><k@bioruby.org>

# 前回>(Ruby入門)

- プログラミング
  - 様々なプログラミング言語
  - プログラムの作成と実行
  - オブジェクト指向
  - 身につけるには？→目的意識、実際に試す

- Ruby>とは
  - 国産でフリーのオブジェクト指向スクリプト言語
  - プログラムの制御構造
    - while,>if,>case,>イテレータ,>...
  - 組み込みクラスの便利なメソッド
    - 数値、文字列、配列、ハッシュ、正規表現、...

# なぜRuby

- Rubyはどのような場面に向いているか
  - Perl同様、文字列処理や正規表現が得意
  - 遺伝子の配列やアノテーションなど文字情報が多い生物学のデータを扱うのにも便利>→>Perlの成功

- Ruby>の代わりに>Perl,>Python>でも全く問題ない
  - 普及率と使いやすさ（慣れ？）のトレードオフ
  - 使いたいライブラリがあるかどうか
  - JavaScript,>PHP,>SQL>なども知っていると便利

- C,>C++,>Java>などはコンパイルが必要
  - プログラムが長くなりがち→手軽ではない
  - 高速な処理が可能、GUI>アプリなどにも向く

# Perlは？

- スクリプト言語の草分けでたぶん最も普及している
  - ユーザ数が多く、ライブラリも豊富
  - BioPerlがある

- でもデータ構造に弱い
  - 配列とハッシュの組み合わせ
  - Perl5以降のオブジェクト指向機能でカバー
    - 記号やオマジナイの増加>(リファレンス,>bless,>@ISA ,>...)

- さらに基本的に使い捨てプログラム用の言語なので
  - 他人のPerlスクリプトは読みにくい
    - しばらく前に自分が書いたプログラムでも、、

# 最初に何を選ぶか

- どれか１つの言語をちゃんと覚えれば他もわかる
- 普及しているもの、身の回りに使っている人がいるもの
- 自分が改良して使いたいソフトが書かれている言語
- 末永く使えるもの
  - マニアックな言語、特定の製品に特化したものは？
- コンパイル言語（スピード）vs>スクリプト言語（開発効率）
  - 情報系→C,>Java、生物系→Ruby,>Perl>とか
  - RubyもCなどで高速な拡張ライブラリが書ける

# 分かりやすい言語から入ると

- 無駄なことを覚えなくてよいのでキレイに書く癖
  - 何をやっているかロジックの見えるコード
- 理解が速いのですぐ使えるようになる
  - スピードには２種類
    - プログラムの実行が速い
    - プログラムの開発が速い
  - アルゴリズムが悪いと実行が速くても意味ない
- オブジェクト指向はRubyならよく分かったり
  - 後から>Perl5,>Java,>C++>を見ると読める、、

# なぜオブジェクト指向か

- そんなに大げさなものではなく、単に便利だから
- オブジェクトはデータのいれもの（構造体）
  - さらにデータの操作方法（メソッド）を定義
    - 化合物オブジェクトには名前、構造式、機能説明などのデータ（情報）が入っている
    - 組成や分子量を計算するメソッドなど
- オブジェクトに共通の特徴を括り出したものがクラス
  - 抽象化（モノをどう捉えるか、理解するか）
  - 複数のクラスに共通の特徴を括り出したらスーパークラス（これを各クラスで継承し差分を書く）

# なぜオブジェクト指向か

- そんなに大げさな
- オブジェクトは
  - さらにデータ
    - 化合物オ
      などのデ
    - 組成や分
- オブジェクト
  - 抽象化（モ
  - 複数のクラス
    クラス（これ

```
class Cpd
  MW = { 'C'=>12.011, 'H'=>1.00794, 'N'=>14.00674,
         'O' => 15.9994, 'P' => 30.973762 }

  def initialize
    @comp = Hash.new
  end
  attr_accessor :name, :definition, :formula

  # formula から組成を抽出するメソッド
  def composition
    @formula.scan(/([A-Z]+)(\d+)/) do |a, b|
      @comp[a] = b.to_i
    end
    return @comp
  end

  # composition から分子量を計算するメソッド
  def mol_weight
    total = 0.0
    composition.each do |elem, i|
      total += MW[elem] * i
    end
    return total
  end
end

cpd = Cpd.new
cpd.name = "ATP"
cpd.definition = "Adenosine 5'-triphosphate"
cpd.formula = "C10H16N5O13P3"

p cpd.composition
p cpd.mol_weight
```

# ライブラリとは

- 言語に様々な機能を追加する
  - CGIライブラリ→RubyでCGIが使えるようになる
  - GDライブラリ→Rubyで絵が描けるようになる
  - DBIライブラリ→RubyでRDBにアクセスできる
- 車輪の再発明をしないですむ（特にオープンソース）
  - 先人が苦労して作ってくれた成果を共有
  - 多くの人が使っているものはバグが少なく安定
- RAA(Ruby Application Archive)
  - http://raa.ruby-lang.org/
  - PerlならCPAN、Cだと、、sf.net?

# Rubyでライブラリを読み込む

- ライブラリのファイル名を調べて require する
  - /usr/local/lib/ruby/1.8
  - /usr/local/lib/ruby/site_ruby/
- マニュアルを読んで使ってみる
  - http://localhost/hello.html

```
<html>
<head><title>hello</title></head>

<body>

<form method="GET" action="/cgi-bin/hello.cgi">
What is your name?
<input type=text name="name">
<input type=submit>
</form>

</body>
</html>
```

```
#!/usr/bin/env ruby

# これで CGI クラスが使える
require 'cgi'

cgi = CGI.new

name = cgi.params['name']

cgi.out do
  "Hello #{name}!<br>"
end
```

# shbang

- Rubyプログラムなどスクリプトの一行目の「#!」

```
#!/usr/bin/ruby
#!/usr/local/bin/ruby
#!/usr/bin/env ruby
```

- シェルが>#!>に続くプログラムを起動、中身を渡す

- env>は環境変数を返すので、PATH>から続くコマンドを探して実行してくれることになる

```
k@hoihoi:~% env
USER=k
HOME=/Users/k
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin:/usr/local/bin:.
SHELL=/bin/zsh
  :
```

# よく使うRubyのライブラリ（標準添付）

- CGI, URI, Net::HTTP
  - ウェブ関係（URLの処理やCGIプログラム作成）
- getoptlong
  - コマンドライン引数を扱う
- SOAP4R (Ruby 1.8)
  - ウェブサービスへのアクセス
- REXML (Ruby 1.8)
  - XMLファイルの処理
- YAML (Ruby 1.8)
  - オブジェクトをダンプ／リストア
- MD5, ftools, open3, pp, profile
  - ツール類

# よく使うRubyのライブラリ （後で追加）

- XMLParser
  - 高速なXMLファイルの処理
- DBI,>DBD(mysql,>postgres>etc.)
  - 関係データベース(RDB)へのアクセス
- GD
  - 絵を描く

```
#!/usr/bin/env ruby

require 'GD'

filename = ARGV.shift

image = GD::Image.new(100,100)

bgcolor = image.colorAllocate("#000000")
fgcolor = image.colorAllocate("#ff0000")

image.line(10, 10, 90, 90, fgcolor)

File.open(filename, "w+") do |file|
  image.png(file)
end
```

# BioRubyとは

- バイオインフォマティクスのための>Ruby>ライブラリ
  - Bio::Sequence,>Bio::Location,>Bio::Feature
    - 配列操作、スプライシング、ウィンドウサーチなど
  - Bio::Registry,>Bio::SQL,>Bio::Fetch,>Bio::FlatFile
    - OBDAによるデータベースへのアクセス
  - Bio::DB
    - >GenBank,>SwissProt,>KEGG>などデータベースのパーザ
  - Bio::Blast,>Bio::Fasta>(他>HMMER,>MAFFT,>Sosui,>Psort>etc.)
    - 解析アプリケーションの実行と結果のパーザ
  - Bio::DAS,>Bio::KEGG::API,>Bio::DDBJ::XML
    - ウェブサービス
  - Bio::Pathway,>Bio::Relation
    - グラフアルゴリズム
  - Bio::PubMed,>Bio::Reference
    - 文献情報

# Bio::Sequence->配列操作

```
#!/usr/bin/env ruby

require 'bio'

seq = "atgcacgggaactaa"

dna = Bio::Sequence::NA.new(seq)

puts dna.subseq(3,6)                  # "gcac"
puts dna.complement                   # "ttagttcccgtgcat"
puts dna.composition                  # {"a"=>6, "c"=>3, "g"=>4, "t"=>2}
puts dna.gc                           # 46.7

protein = dna.translate

puts protein                          # "MHGN*"
puts protein.molecular_weight         # 439.5
```

```
% ruby seq.rb
```

# Bio::CodonTable->コドン表

```
#!/usr/bin/env ruby

require 'bio'

# make a DNA sequence including 'tga'
seq = "atgggcccatgaaaaggcttggagtaa"
dna = Bio::Sequence::NA.new(seq)

# translate from the frame 1 with
# codon table 10 (Euplotid Nuclear)
ct10 = Bio::CodonTable[10]
protein = seq.translate(1, ct10)

# print out the protein
puts protein        # => "MGPCKGLE*"

# compared to the universal table
puts seq.translate  # => "MGP*KGLE*"
```

- 1 - Standard (Eukaryote)
- 2 - Vertebrate Mitochondrial
- 3 - Yeast mitochondorial
- 4 - Mold, Protozoan, Coelenterate Mitochondrial and Mycoplasma/Spiroplasma
- 5 - Invertebrate Mitochondrial
- 6 - Ciliate Macronuclear and Dasycladacean
- 9 - Echinoderm Mitochondrial
- 10 - Euplotid Nuclear
- 11 - Bacteria
- 12 - Alternative Yeast Nuclear
- 13 - Ascidian Mitochondrial
- 14 - Flatworm Mitochondrial
- 15 - Blepharisma Macronuclear
- 16 - Chlorophycean Mitochondrial
- 21 - Trematode Mitochondrial
- 22 - Scenedesmus obliquus mitochondrial
- 23 - Thraustochytrium Mitochondrial Code

# OBDA (Open Bio* Database Access)

- BioHackathon (2002, 2003) で制定
  - データベースのエントリを取得する仕組みを BioPerl, BioPython, BioJava, BioRuby で共通化
- 設定ファイル
  - ~/.bioinformatics/seqdatabase.ini
  - /etc/bioinformatics/seqdatabase.ini
  - http://open-bio.org/registry/seqdatabase.ini

```
VERSION=1.00

[embl]
protocol=biofetch
location=http://www.ebi.ac.uk/cgi-bin/dbfetch
dbname=embl

[swissprot]
protocol=biosql
location=db.bioruby.org
dbname=biosql
driver=mysql
biodbname=sp
```

# データベースへのアクセス

```
#!/usr/bin/env ruby

require 'bio'

serv = Bio::Registry.new

db = serv.get_database("swissprot")

entry = db.get_by_id("TETW_BUTFI")

puts entry
```

```
#!/usr/bin/env ruby

require 'bio'

serv = Bio::KEGG::API.new

entry = serv.bget("sp:TETW_BUTFI")

puts entry
```

**bget コマンドの作成**

```
#!/usr/bin/env ruby

require 'bio'

serv = Bio::KEGG::API.new

if /:/.match(ARGV[0])
  list = ARGV
else
  db = ARGV.shift
  list = ARGV.map {|entry| "#{db}:#{entry}"}
end

puts serv.get_entries(list)
```

# GenBank

```
#!/usr/bin/env ruby

require 'bio'

file = "gbbct1.seq"

Bio::GenBank.open(file).each do |entry|
  entry.each_gene do |feat|
    pos = feat.position
    seq = entry.seq.splicing(pos)
    desc = entry.entry_id + " " + pos
    puts seq.to_fasta(desc, 60)
  end
end
```

- Feature>Table

```
FEATURES             Location/Qualifiers
     source          1..5842
                     /organism="Streptococcus agalactiae"
                     /db_xref="taxon:1311"
                     /clone="pGB3634"
     gene            join(5804..5842,1..240)
                     /gene="copR"
     CDS             join(5804..5842,1..240)
                     /gene="copR"
                     /note="circular"
                     /codon_start=1
                     /transl_except=(pos:5804..5806,aa:Met)
                     /transl_table=11
                     /protein_id="CAA50904.1"
                     /db_xref="GI:2673854"
                     /db_xref="SWISS-PROT:P24716"
                     /translation="MELAFRESLKKMRGTKSKEKFSQE
                     KTLEQIVKLTNSTLVVDLIPNEPTEPEPETEQVTLELE
```

# Bio::FlatFile

```
#!/usr/bin/env ruby

require 'bio'

Bio::FlatFile.auto(ARGF) do |ff|
  ff.each do |entry|
    puts entry.entry_id
    puts entry.definition
  end
end
```

```
% ruby flatfile.rb gbbct1.seq
% ruby flatfile.rb e.coli.pep
```

# Blast

```
#!/usr/bin/env ruby
require 'bio'

prog = ARGV.shift# 'blastp'
db = ARGV.shift  # 'e.coli.pep'
blast = Bio::Blast.local(prog, db)

Bio::FlatFile.auto(ARGF) do |ff|
  ff.each do |entry|
    puts entry.entry_id
    report = blast.query(ff.raw)
    report.each do |hit|
      puts hit.target_id
      hit.each do |hsp|
        puts [
          hsp.query_id,
          hsp.target_id,
          hsp.evalue
        ].join("?t")
      end
    end
  end
end
```

```
% ruby blast.rb blastp e.coli.pep query.f
```

# KEGG>API

```
#!/usr/bin/env ruby

require 'bio'
require 'uri'
require 'net/http'

# KEGG API のサーバに接続し WSDL から利用可能なメソッドを取得
serv = Bio::KEGG::API.new

# 指定したパスウェイに載っている遺伝子名のリストを取得
path = ARGV.shift || 'path:eco00010'
genes = serv.get_genes_by_pathway(path)

# LinkDB を用いて、各遺伝子から PDB データベースへのリンクを探す
results = Hash.new
genes.each do |gene|
  print gene
  if links = serv.get_linkdb_by_entry(gene, 'pdb')
    # リンクがあれば、順番にリンク先のエントリ名を表示する
    links.each do |link|
      results[gene] ||= Array.new
      results[gene] << link.entry_id2
      print "\t" + link.entry_id2
    end
  end
  puts
end
```

# KEGG>API

```
# リンクがあった遺伝子のリストを渡して枠に色付けした画像を生成する
url, = serv.mark_pathway_by_genes(path, results.keys)
STDERR.puts url

# 画像の URL が返されるので、実際の画像ファイルを取得し、ファイルに保存する
host = URI.parse(url).host
path = URI.parse(url).path

filename = 'pdb_on_' + File.basename(path)

File.open(filename, File::CREAT|File::TRUNC|File::RDWR) do |file|
  file.print Net::HTTP.get_response(host, path).body
end
```

# 参考文献

- http://bioruby.org/
- Code>Reading
- Ruby>Way