Julia 入門

Julia とは

参考：Julia in a Nutshell, なぜ僕らはJuliaを作ったか

速い

LLVM を利用してネイティブコードを生成する（らしい）

簡単に書ける

動的型付けのおかげで気軽に書ける。対話環境も充実。

いろいろな構成に対応

多重ディスパッチでOOP（構造体に関数が作用する書き方）や関数型っぽく書ける

そしてオープンソース

インストール

直接

- https://julialang.org/ からダウンロード、インストール
- インストール場所はどこでもOK、パスを通そう
- `julia` で REPL（対話環境）開始

Jupyter でも使いたい人（参考：Jupyter NotebookeでJuliaを使ってみた）

- `conda` コマンドから julia をインストールする方法もあるらしい（インストールしてしまえば「直接」と同じ状況のはず）
-] キーでパッケージモードに移って `add IJulia`
- Backspace で julia モードに戻って `using IJulia; notebook()` で起動

REPL の使い方

- `exit()` か Ctrl+D で終了
-] で **パッケージモード**
- ? で **ヘルプモード**
- BackSpace で **julia モード** に戻る
- パッケージの追加：
  - (パッケージモード) `add パッケージ名`
  - (julia モード) `using Pkg; Pkg.add("パッケージ名")` (import でも同じ)
- インストールしたパッケージの確認：`Pkg.status()`
- パッケージのアップデート：`Pkg.update()`

REPL を使いこなそう

- 起動コマンドを `julia -q` とすると julia のアスキーアートが表示されない
- **Tab 変換**：`\alpha`+Tab, `\Alpha`+Tab などと入力して `α`, `A` などに置換される（`π` 以外はだいたい変数用）（TeX 記号は結構対応している）

（余談：LaTeX では \Alpha コマンドは用意されておらずアルファベットのAで代用せざるを得ないが Unicode では区別される。cf.Unicode一覧 0000-0FFF）

- ヘルプモードで記号を入力すると TeX での打ち方、演算子の場合は用例も分かる
- 定数 `π`, `▫` が用意されている(`\pi`, `\euler`+Tab)（cf. Base.Mathconstants）
- 円周率は `pi` でも同じ扱い

REPL as 電卓

- 算術、論理、比較、ビット演算子は下表以外は python とほぼ同じ

記号	意味	出し方	記号	意味	出し方
^	累乗		≠	不等(!=)	\ne +Tab
÷	整数除算	\div +Tab	≤	小なり(<=)	\le +Tab
◻	XOR	\xor +Tab	≥	大なり(>=)	\ge +Tab
//	分数		≢	不等(!==)	\nequiv +Tab

- ~~≠ は \neq ではなく \ne
- **数値リテラル係数**：係数が数値なら掛け算の '*' を略せる。 x=1; √2^2x^2+(x-1)x

いろいろ試すために

- コメントアウトは `#`。範囲コメントアウトは `#=` と `=#`。
- **`println()`** でコンソール出力

データ型

- **`typeof(1)`** などとしてデータ型を確認できる
- `1.0`, `1 // 7`, `π`, `2.0im`, `true`, `'a'`, `'␣'`, `"ABC"` の型を確認してみよう
- **型変換関数** はデータ型と同じ名前。`Float64(pi * 2)`, `BigFloat(␣)`
- `zero(x)`, `one(x)` などとすると `x` と同じ型の `0`, `1` が得られる

文字列・配列

- String は Char の配列
- **アクセス**：先頭 `[1]`, `[begin]`, 末尾 `[end]`, 真ん中（切り捨て） `[end÷2]`
- **配列のスライス**：`"hello"[2:4]` とすると `"ell"` が切り出せる
- **文字列の結合**：`string("Java","script")` とするか `"インド" * "ネシア"` とする（`+` じゃないのは非可換だからしい）
  `string` を使う方法なら文字列以外も文字列として結合できる
- **文字列の置換**：`replace("Word to vec", " to " => 2)`
- **配列の長さ**：`length("four")`
- **コードの挿入**：`print("x=$(1+2)")`
- `"""` で複数行にまたがる文字列を書ける

関数

```
function f(x,y)
    x * y # 最後の値が戻り値。return で明示するのも可
end
```

インデントはなくても動く。型も指定できる。

```
function cat(x::String, y::String) :: String
   x * y
end

function cat(x::Int64, y::Int64) :: String
   string(x) * string(y)
end
```

引数の型が違えば違う関数。

for 文、if 文、可変長引数

```
function add(x...)
  sum = 0
  for i = 1:length(x) # for i in 1:length(x) でも同じ
    if x[i] ≤ 0
      continue
    elseif sum ≥ 100
      break
    end
    sum += x[i]
  end
  sum
end

println(add(1,2,3,4,5,-6,100,200)) # 115
```

REPL上では `end` を書くまで改行できる。末尾参照の `end` とは異なる使い方。

辞書（連想配列）

```
戦いの年号集 = Dict("関ケ原" => 1600, "桶狭間" => 1560, "小牧・長久手" => 1584)
# 戦いの年号集 = Dict{String, Int32}("関ケ原" => 1600,
#     "桶狭間" => 1560, "小牧・長久手" => 1584) 型を明示

if haskey(戦いの年号集, "関ケ原")
    println(get(戦いの年号集, "関ケ原", 0))
end
```

リスト内法表記のような書き方も可能。

```
Dict(i => i ^ 3 for i = 1:10)
```

順番はばらばら。

少し発展的

- `[(2i-1)^3 for i=1:5]` の代わりに `[1:2:9;].^3` としても同じ。
  - `(start):(step):(end)` で範囲を生成できる。`;` をつけると数列になる
  - **ドット演算子**：`.` で各要素に関数を作用させる。`sin.([0.5pi, pi, 1.5pi])`
- ? でヘルプモードに移って `|>` と入力してみよう
  - **ラムダ式**：9p. の関数は `f = (x, y) -> x * y` とも書ける
  - `|>`：変数に関数を次々と作用させる。
- **`do`**：無名関数を作って第1引数として渡す。`map(1:2:9) do x; x^3; end`
- ローカルスコープからグローバル変数に書き込むときは `global` をつける（スコープ内で1度でOK）
- 演算子は上書きできる Juliaでのユーザー定義演算子の種類

グラフをプロット

-] でパッケージモードに移って `add Plots` で Plots パッケージをインストール（数分かかる）

```
x = -5.0:0.1:5.0 # -5から5まで0.1刻み
f(x) = 1 ./ (1 .+ exp.(-x)) # xの各要素にシグモイド関数を作用させる
y = f(x)

using Plots
plot(x, y)
savefig("Sigmoid.png") # 画像保存
```

なぜか画像保存以外ではグラフを見られない（Jupyter なら可能）

アニメーション

```
using Plots

x = -5.0:0.1:5.0
f(x,a) = 1 ./ (1 .+ exp.(-a.*x)) # a はゲイン

anim = Animation()
for a in 1.0:0.1:2.0
  # plot でラベル、軸範囲なども設定できる
  plt = plot(x, f(x, a), label = "gain:$a",
    xlims = (-5,5), ylims = (0,1), xlabel = "x", ylabel = "f(x)")
  frame(anim, plt)
end
gif(anim, "sigmoid_anim.gif", fps = 5)
```

行列演算

- `using LinearAlgebra`
- `A = [1 2;3 4]` で $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
- `A*B` で行列積、`A'` でエルミート共役、`A.*B` でアダマール積（成分ごとの積）
- `B=copy(A)` でコピー（配列も同じ）
- エルミート行列 H に対して `λ,U=eigen(H)` で固有値たち λ とユニタリUが求まる
- 行列からベクトルを切り出す：`u1 = U[:,1]` は列を切り出す（列も行も切り出すと列ベクトル）
- 逆行列は `inv(A)`、行列式は `det(A)`、トレースは `tr(A)`、指数行列は `exp(A)`

素数

- `add Primes`
- `primes(2, 97)` で $2 \sim 97$ の素数
- `isprime()`：素数判定、`factor()`：素因数分解
- Prime number functions 詳細。公式。

フィッティング

```
using LsqFit, Plots
xdata = [5:0.2:10.8;]
ydata = [1.9, 2.2, 2.5, 2.1, 2.8, 2.9, 2.2, 2.6, 2.4, 3.6,
         5.5, 9.2, 25, 110, 340, 550, 220, 51, 13, 5.4, 8.9,
         5.7, 2.5, 5.1, 5.6, 3.2, 3.0, 2.5, 2.1, 1.8].*1e16
function func(x, prm) # フィッティングのモデル
  @. prm[1] * exp(-(x - prm[2]) ^ 2 / (2prm[3] ^ 2)) + prm[4]
end
parameter_initial = [1e18, 8, 3, 1e15] # 係数の初期値
fit = curve_fit(func, xdata, ydata, parameter_initial)
println("param->", fit.param, ", covar->", estimate_covar(fit),
  ", stderr->", stderror(fit)) # 係数、共分散、標準偏差
plot(xdata, ydata, seriestype = :scatter)
plot!([5:0.02:10.8;], func([5:0.02:10.8;], fit.param))
savefig("lsqfit-sample.png")
```

詳しく -> LsqFit.jl

さらに詳しく

- Julia 1.0 ドキュメント 有志による一部日本語化ドキュメント。最新は 1.6.4 なので内容は古いかもしれないが、基本的な言語仕様は同じ？
- Julia 1.6 Documentation 困ったら公式。英語。
- Juliaで数値計算　その1：コードサンプル〜基本的計算編〜 紹介しきれていない入門的な事項も書かれている。

参考文献

- Julia言語プログラミング入門
- REPL (julia コマンド) の使い方