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

オブジェクト指向言語や関数型言語の書き方ができる

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.installed()`
- 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(`!=`)	コピペ
`÷`	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` が用意されておらず、ヘルプモードが示す `=\not` もエラー
- **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
  `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")`

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）

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）

行列演算

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。

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