ÖGOR Summer-Workshop for PhD-candidates and Post-Docs

An introduction to Julia and JuMP for Operations Research

Prof. Dr. Xavier Gandibleux

Nantes Université – France
Département Informatique – Faculté des Sciences et Techniques

August 2023

Short bio


NANTES
Caen
Lisieux
Evreux
PARIS
Versailles
Dreux
Chartres
Alençon
Fougères
Rennes
Laval
Le Mans
Orléans
Blois
Tours
Angers
Saumur
Cholet
Châtellerault
Poitiers
Châteauroux
Niort
La Roche-sur-Yon
Saint-Nazaire
Vannes
Lorient
Quimper
Brest
Morlaix
Saint-Brieuc
Dinan
Dinard
Avranches
Coutances
Senlis
Meaux
Melun
Fontainebleau
Evry
Créteil
St-Germain-en-Laye
St-Denis
Guéret

Short bio

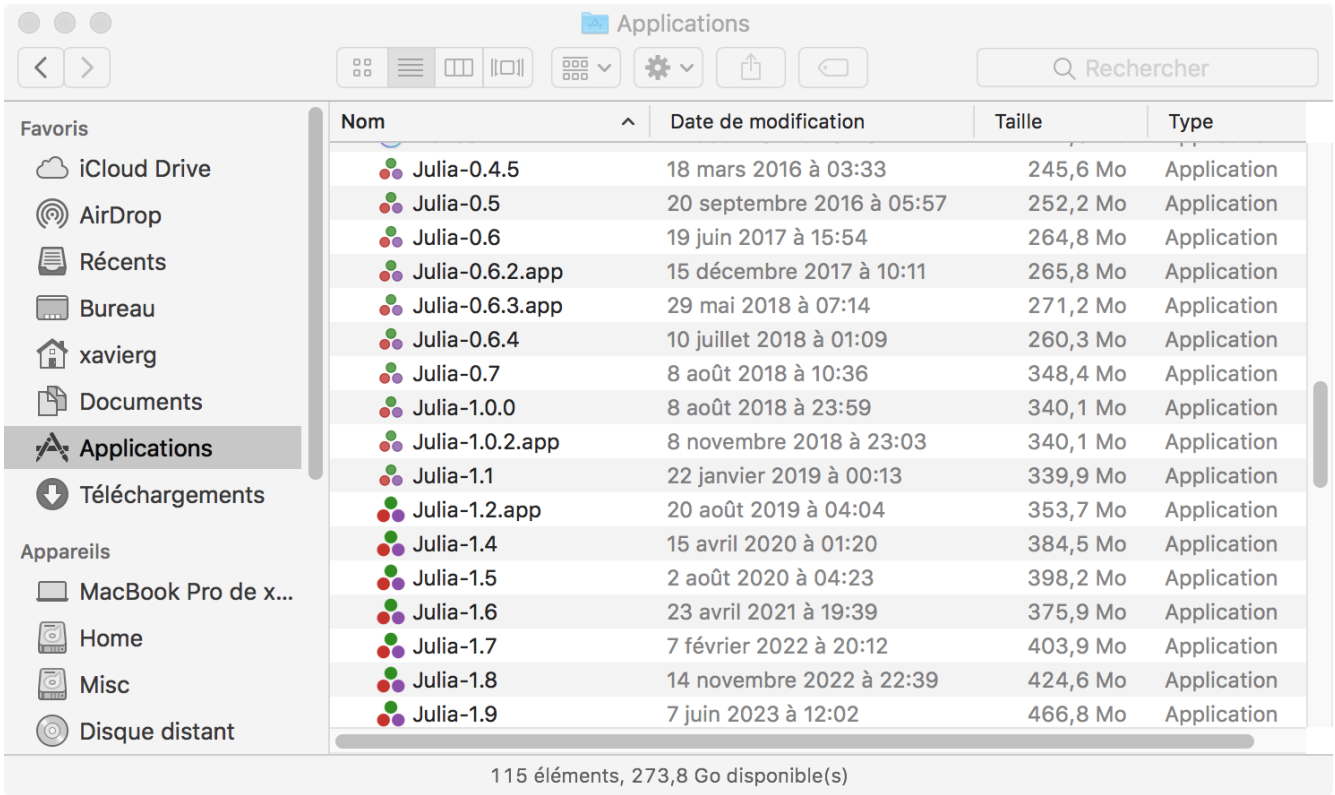
1. Teaching areas
   - Optimization: operations research; discrete and combinatorial optimization; multi-objective optimization.
   - Artificial intelligence: metaheuristics; multi-objective metaheuristics.
   - Computer science: Introduction to computer concepts; Julia programming language.
2. Research areas
   - Optimization: operations research; multi-objective optimization.
   - Artificial intelligence: metaheuristics.
   - Scientific computing: MIP solvers.
   - Applications: railway transportation, autonomous vehicles.

Short bio

1. Teaching areas
   - Optimization: operations research; discrete and combinatorial optimization; multi-objective optimization.
   - Artificial intelligence: metaheuristics; multi-objective metaheuristics.
   - Computer science: Introduction to computer concepts; Julia programming language.
2. Research areas
   - Optimization: operations research; multi-objective optimization.
   - Artificial intelligence: metaheuristics.
   - Scientific computing: MIP solvers.
   - Applications: railway transportation, autonomous vehicles.

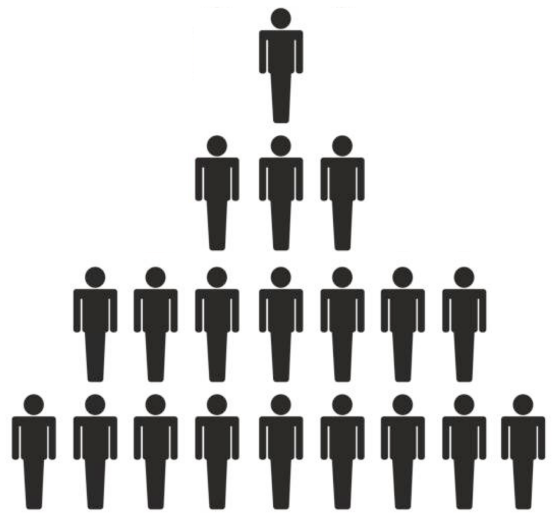
Julia

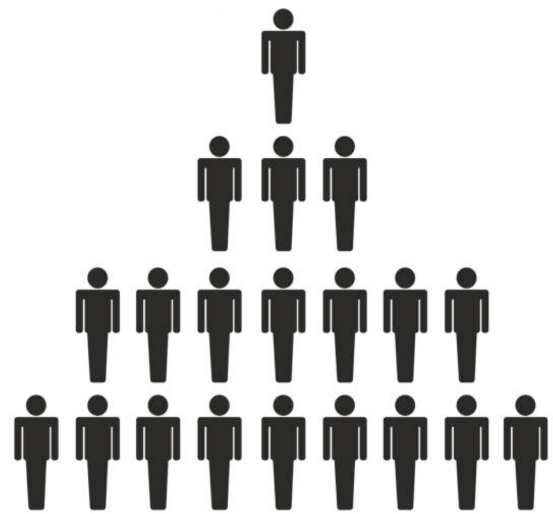
My story with Julia and JuMP: ROADEF'2016 (Feb 2016)



Julia

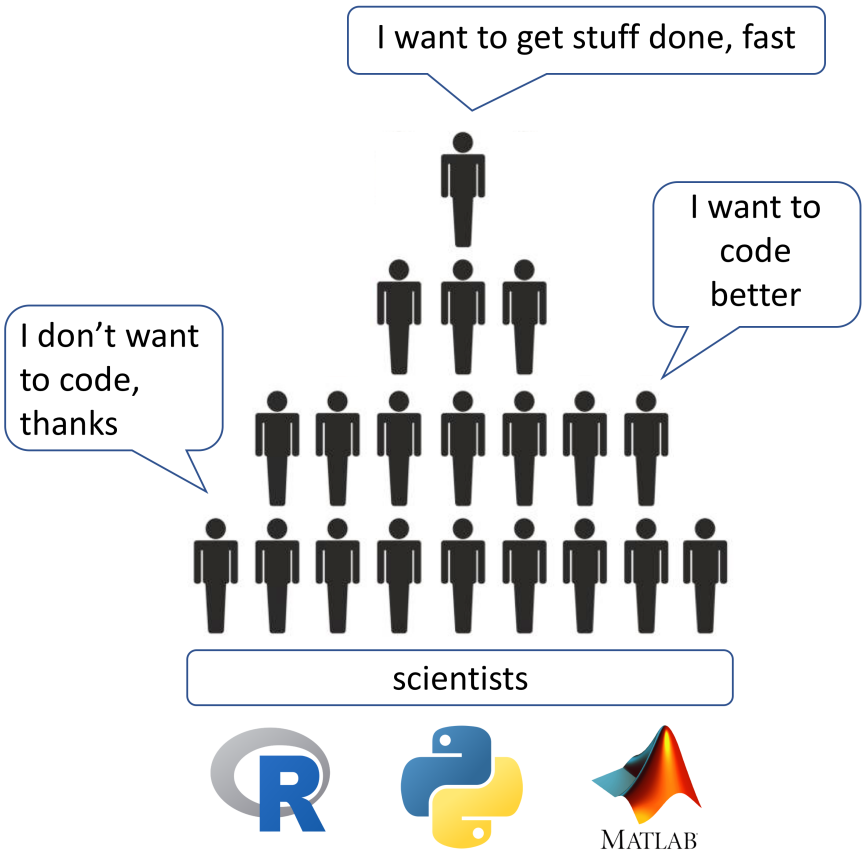
A programming language for optimization?

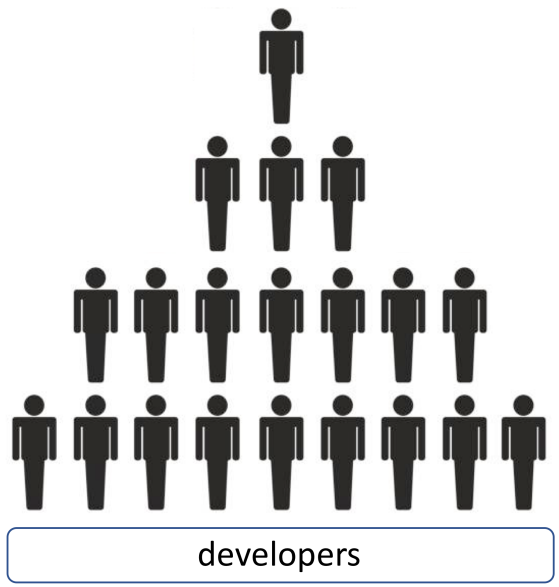




Julia

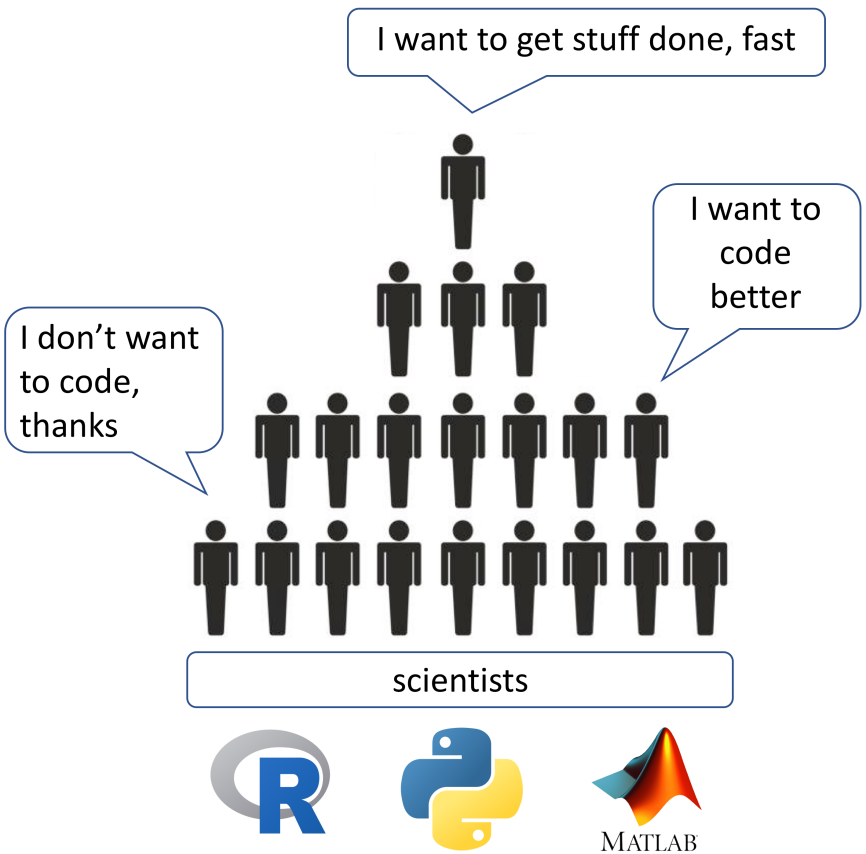
A programming language for optimization?

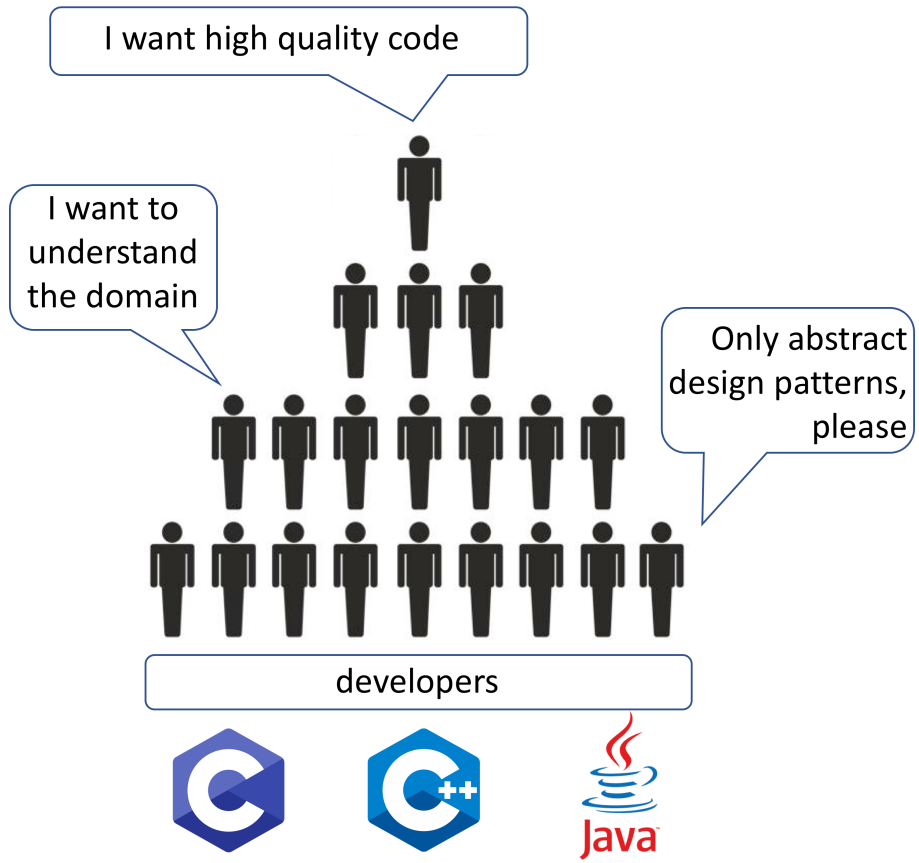




Julia

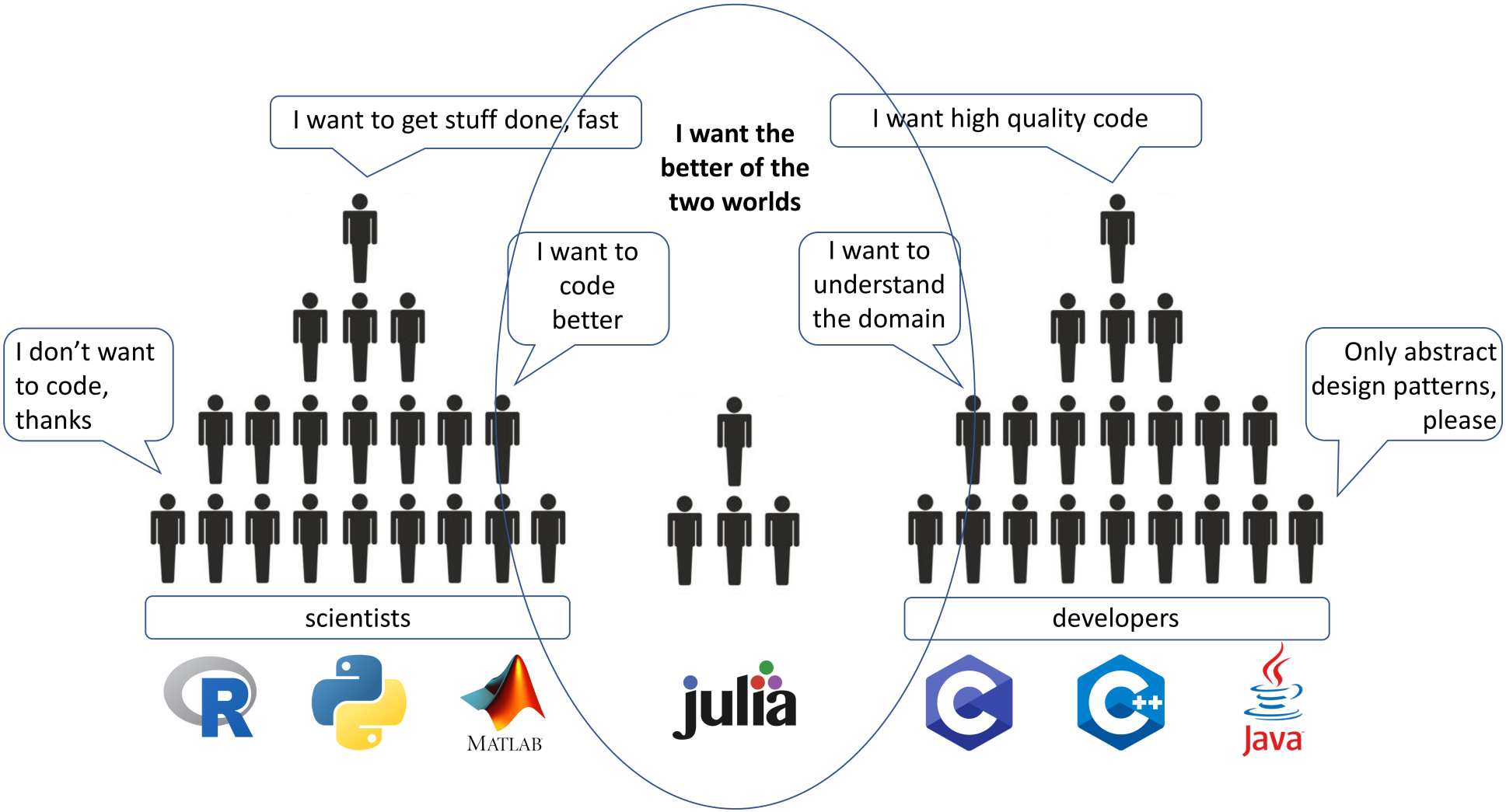
A programming language for optimization?





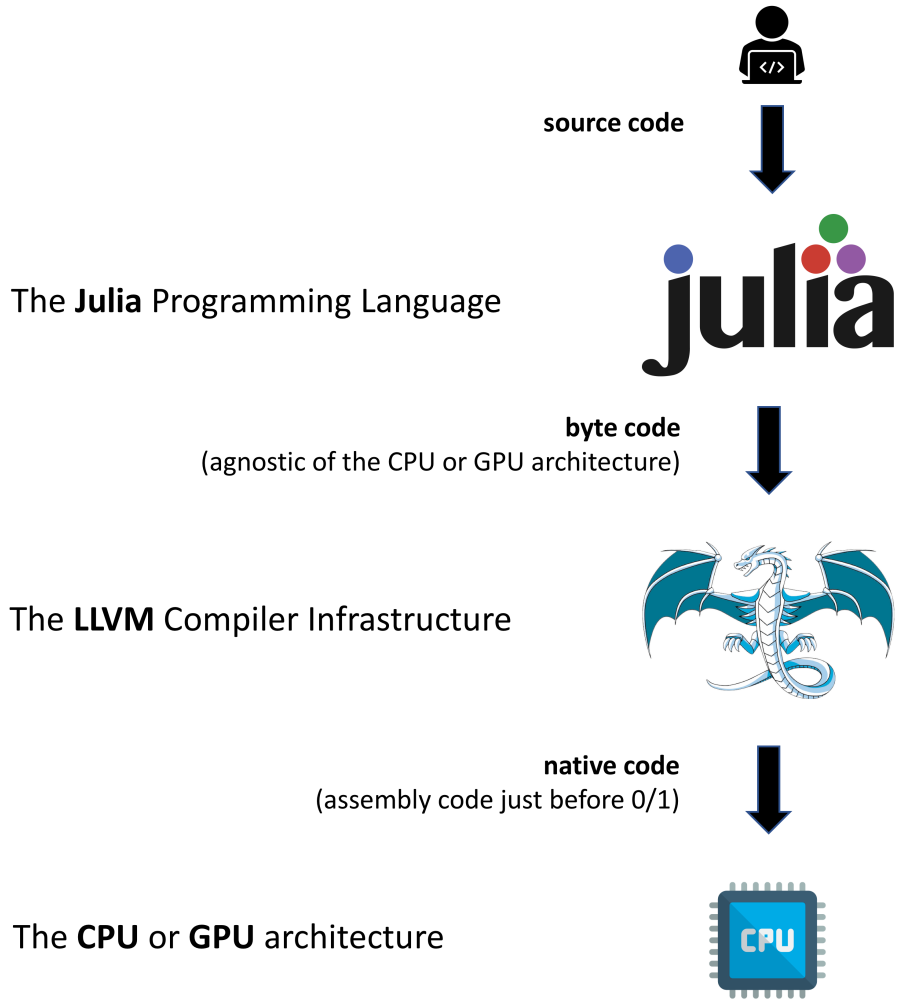
Julia

A programming language for optimization?



Julia

LLVM: middleman between source code and compiled native code




```
julia> 1+2
3
```

```
julia> @code_llvm 1+2
;  @ int.jl:87 within `+`
define i64 @"julia_+_115"(i64 signext %0, i64 signext %1) #0 {
top:
  %2 = add i64 %1, %0
  ret i64 %2
}
```

```
julia> @code_native 1+2
        .section        __TEXT,__text,regular,pure_instructions
        .build_version macos, 13, 0
        .globl  "_julia_+_150"                  ; -- Begin function julia_+_150
        .p2align        2
"_julia_+_150":                         ; @"julia_+_150"
; ┌ @ int.jl:87 within `+`
        .cfi_startproc
; %bb.0:                                ; %top
        add     x0, x1, x0
        ret
        .cfi_endproc
; └
                                        ; -- End function
.subsections_via_symbols
```

```
01101100
01101111
01110110
01100101
```

Julia

Example: how to get the column sums of a matrix . . .

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$V = \begin{pmatrix} 4 & 6 \end{pmatrix}$$

. . . in C/C++ ?

Julia

Example: how to get the column sums of a matrix?

Assembler

C/C++

```c
#include <stdio.h>
#define nLin 2
#define nCol 2

int main(void)
{
    int A[nLin][nCol] = {{1,2} ,
                         {3,4}};
    int V[nCol];
    int i, j;

    for (j = 0; j < nCol; j++)
    {
        V[j] = 0;
        for (i = 0; i < nLin; i++)
            V[j] += A[i][j];
        printf("%3d ", V[j]);
    }
    printf("\n");
    return 0;
}

```

Julia

Julia

Example: how to get the column sums of a matrix?

Assembler

```
26          str     x8, [sp]
27          ldr     x8, [x0]
28          str     x8, [sp, #8]
29          mov     x8, sp
30          str     x8, [x0]
31
32          ldr     x1, [x20]
33          mov     x0, x19
34          mov     x2, #0
35          bl      _j_reducedim_initarray
36          mov     x20, x0
37          str     x0, [sp, #16]
38          mov     x8, #16720
39          movk    x8, #6111, lsl #16
40          movk    x8, #1, lsl #32
41          mov     x9, #38080
42          movk    x9, #6109, lsl #16
43          movk    x9, #1, lsl #32
44
45          stp     x8, x9, [sp, #32]
46          stp     x0, x19, [sp, #48]
47          add     x1, sp, #32
48          mov     x0, #13456
49          movk    x0, #6286, lsl #16
50          movk    x0, #1, lsl #32
51          mov     w2, #4
52          bl      "_j1__mapreducedim!_1977"
53          ldr     x8, [sp, #8]
54          str     x8, [x21]
55
56          mov     x0, x20
57          ldp     x29, x30, [sp, #96]
58          ldp     x20, x19, [sp, #80]
59          ldp     x22, x21, [sp, #64]
60          add     sp, sp, #112
61          ret
```

C/C++

```c
#include <stdio.h>
#define nLin 2
#define nCol 2

int main(void)
{
    int A[nLin][nCol] = {{1,2} ,
                         {3,4}};
    int V[nCol];
    int i, j;

    for (j = 0; j < nCol; j++)
    {
        V[j] = 0;
        for (i = 0; i < nLin; i++)
            V[j] += A[i][j];
        printf("%3d ", V[j]);
    }
    printf("\n");
    return 0;
}

```

Julia

```julia
A = [1 2; 3 4]
V = sum(A, dims=1)
println(V)

```

version 1.9.x and later

Why Julia?

1. Runs like C, walks like Python, talks like Lisp
   - Same speed as C or FORTRAN code
   - Development dynamic like writing python code
   - Less object-oriented more function-oriented (like Lisp)

   ⇒ High-level and high-performance!

   ⇒ Extremely good ratio between coding time and running time.

   ⇒ Familiar for the users of Matlab, Fortran, Python, C/C++, etc.

2. Additional value
   - Open source: MIT license, source code available on GitHub
   - Multi-platform (x86, ARM, Apple Silicon, etc.)
   - Cross-language support: C, FORTRAN, python, R ... from Julia
   - Rich set of packages: Optimisation, Plotting, Visualisation, Statistics, Data science, GPU, etc.

   ⇒ Ideal for the development of efficient open-source app

Why Julia?

1. Runs like C, walks like Python, talks like Lisp
   - Same speed as C or FORTRAN code
   - Development dynamic like writing python code
   - Less object-oriented more function-oriented (like Lisp)

   ⇒ High-level and high-performance!

   ⇒ Extremely good ratio between coding time and running time.

   ⇒ Familiar for the users of Matlab, Fortran, Python, C/C++, etc.

2. Additional value
   - Open source: MIT license, source code available on GitHub
   - Multi-platform (x86, ARM, Apple Silicon, etc.)
   - Cross-language support: C, FORTRAN, python, R ... from Julia
   - Rich set of packages: Optimisation, Plotting, Visualisation, Statistics, Data science, GPU, etc.

   ⇒ Ideal for the development of efficient open-source app

Why Julia?

1. Runs like C, walks like Python, talks like Lisp
   - Same speed as C or FORTRAN code
   - Development dynamic like writing python code
   - Less object-oriented more function-oriented (like Lisp)

   ⇒ High-level and high-performance!

   ⇒ Extremely good ratio between coding time and running time.

   ⇒ Familiar for the users of Matlab, Fortran, Python, C/C++, etc.

2. Additional value
   - Open source: MIT license, source code available on GitHub
   - Multi-platform (x86, ARM, Apple Silicon, etc.)
   - Cross-language support: C, FORTRAN, python, R ... from Julia
   - Rich set of packages: Optimisation, Plotting, Visualisation, Statistics, Data science, GPU, etc.

   ⇒ Ideal for the development of efficient open-source app

Why Julia?

1. Runs like C, walks like Python, talks like Lisp
   - Same speed as C or FORTRAN code
   - Development dynamic like writing python code
   - Less object-oriented more function-oriented (like Lisp)

   $\Rightarrow$ High-level and high-performance!

   $\Rightarrow$ Extremely good ratio between coding time and running time.

   $\Rightarrow$ Familiar for the users of Matlab, Fortran, Python, C/C++, etc.

2. Additional value
   - Open source: MIT license, source code available on GitHub
   - Multi-platform (x86, ARM, Apple Silicon, etc.)
   - Cross-language support: C, FORTRAN, python, R ... from Julia
   - Rich set of packages: Optimisation, Plotting, Visualisation, Statistics, Data science, GPU, etc.

   $\Rightarrow$ Ideal for the development of efficient open-source app

A high-level, high-performance programming language for scientific computing

https://julialang.org/

- Timeline:
  2009...2012 @ MIT, USA
  version 1.0.0 in 2018; version 1.9.1 (June 7, 2023)
- Communities:
  Over 9,100 Julia packages https://julialang.org/packages/
  Optimization https://jump.dev/
- JuliaCon...ferences:
  every year since 2014 https://juliacon.org/
  2023: Cambridge, USA. July 25th - 29th, 2023
- Structures:
  JuliaComputing http://juliacomputing.com
  JuliaLab http://julia.mit.edu

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah: Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

A high-level, high-performance programming language for scientific computing

https://julialang.org/

- Timeline:
  2009...2012 @ MIT, USA
  version 1.0.0 in 2018; version 1.9.1 (June 7, 2023)
- Communities:
  Over 9,100 Julia packages https://julialang.org/packages/
  Optimization https://jump.dev/
- JuliaCon...ferences:
  every year since 2014 https://juliacon.org/
  2023: Cambridge, USA. July 25th - 29th, 2023
- Structures:
  JuliaComputing http://juliacomputing.com
  JuliaLab http://julia.mit.edu

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah: Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

A high-level, high-performance programming language for scientific computing

https://julialang.org/

- Timeline:
  - 2009...2012 @ MIT, USA
  - version 1.0.0 in 2018; version 1.9.1 (June 7, 2023)
- Communities:
  - Over 9,100 Julia packages — https://julialang.org/packages/
  - Optimization — https://jump.dev/
- JuliaCon...ferences:
  - every year since 2014 — https://juliacon.org/
  - 2023: Cambridge, USA. July 25th - 29th, 2023
- Structures:
  - JuliaComputing — http://juliacomputing.com
  - JuliaLab — http://julia.mit.edu

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah: Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

A high-level, high-performance programming language for scientific computing

https://julialang.org/

- Timeline:
  - 2009...2012 @ MIT, USA
  - version 1.0.0 in 2018; version 1.9.1 (June 7, 2023)
- Communities:
  - Over 9,100 Julia packages https://julialang.org/packages/
  - Optimization https://jump.dev/
- JuliaCon...ferences:
  - every year since 2014 https://juliacon.org/
  - 2023: Cambridge, USA. July 25th - 29th, 2023
- Structures:
  - JuliaComputing http://juliacomputing.com
  - JuliaLab http://julia.mit.edu

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah: Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

Key concepts

- Julia has an LLVM-based Just-in-time compilation (JIT)
- Compilation down to native code (running on CPU and GPU, ...)
- Julia is dynamically typed, provides multiple dispatch
- Julia has many built-in mathematical functions
- A built-in package manager
- Designed for parallelism and distributed computation

Resources

Some URL to know:

- Official website
  `https://julialang.org/`
- Official documentation (online and pdf)
  `https://docs.julialang.org/`
- Julia channel on *youtube*
  `https://www.youtube.com/channel/UC9IuUwwE2xdjQUT_LMLONoA/featured`
- Questions and discussion on *discourse*
  `https://discourse.julialang.org/`
- Book online « Think Julia: How to Think Like a Computer Scientist »
  `https://benlauwens.github.io/ThinkJulia.jl/latest/book.html`

Resources

Some URL to know:

- Official website
  `https://julialang.org/`
- Official documentation (online and pdf)
  `https://docs.julialang.org/`
- Julia channel on *youtube*
  `https://www.youtube.com/channel/UC9IuUwwE2xdjQUT_LMLONoA/featured`
- Questions and discussion on *discourse*
  `https://discourse.julialang.org/`
- Book online « Think Julia: How to Think Like a Computer Scientist »
  `https://benlauwens.github.io/ThinkJulia.jl/latest/book.html`

Planning, organisation

Planning, organisation

Planning:

- 2 days, notions and practices
- AM: 09h00 → 12h00
- PM: 13h30 → 16h30

Content of the course:

- Day 1: Julia
  Crash course in Julia, optimization algorithms with Julia
- Day 2: JuMP
  Optimization models and algorithms with JuMP

Profile of the public for this course

Level of fluency with

- a programming language..... ★★★☆☆
- data structures..... ★★☆☆☆
- LP/IP/MIP optimisation..... ★★★★☆
- an algebraic modeling language... ★★★★☆