

The Julia programming language

Bolyai Kollégium Informatika Szeminárium

2018. 11. 06.

Features

- Website: <https://julialang.org/>.
- Open-source: <https://github.com/JuliaLang/julia>
- High-level general-purpose dynamic programming language
- High-performance
- Interpreted script language (JIT)
- Cross-platform (even FreeBSD)



History

- **Work began in 2009**
- **v1.0 in August 2018**
- **Tries to solve the “Fortran-Matlab problem”**



Languages for computing

C, Fortran, Assembly (?)

- **Low-level**
- **Compiled**
- **Fast**
- **Procedural, no modern programming concepts**

Python, Matlab, Mathematica

- **High-level**
- **Interpreted**
- **Much slower**
- **Built-in support for many math operations (M*)**
- **Modern programming language concepts (Python)**

Speed

- **Less than 2 times slower than C**
 - while "about 10 times faster" than Matlab (according to the FED [\[1\]](#))

Speed



Usability

- **Multi-paradigm (procedural, functional, and object-oriented)**
- **Built-in support for many things**
 - Standard library is written in Julia
 - Package manager integrated with Github (1900+ packages)
- **Usable interactive shell**
- **IDE as Atom/VS code plugin**

REPL

```
siposagoston@siposagoston-Aspire-F5-573G:~$ julia
```

Documentation: <https://docs.julialang.org>

```
Type "?" for help, "]?" for Pkg help.
```

Version 1.0.1 (2018-09-29)

Official <https://julialang.org/> release

```
julia> 
```

REPL

```
> factorial(BigInt(3000))
```

```
julia> factorial(BigInt(3000))
414935960347354085556867938066121710951119194931809917689467657697558565123531950860007652178003420075184635383617184957508711140459077945534021606833961162103790419917752206266339017968280516471969749596884
24577287660971030037262111095340241127711883315773881532843892973761382110631293037440148537872544607961029042949104979388812076251162513291700446166862117590203575175488980653577868915285093782469994674699190832
895311068363824287063522668544339213715715048858104368188090992929124971419008508938994404715351473154531587441509660471246787508746036714711707238674727714398892068369116185036081984597180937844535239580553776110
06511623631459208861085574508745139453054362137118981508471209442637420327456299963378494041477567141460882412074499914714875396697206389546705899601785694980263877611287106800459082740077112481947638640136919
54435412031278660143479254959514335012065103406625503231020738351502195103148673612323873939509655146215934901578994949407231100442692483814014145548787273808545602356158320431794595305830693351246890721246151
46848530872031267907103892733475375756899365173696424781733462510879015746373398929062767098317634390210717634398332444564047656540014414469947984355495779938670283428521341318891316569531084853135250
940061477740470073314065417944280044366919036854692708572717016480115120574524486079687737848036606530091098156390912941106337156215400938001350586716242623339024316662871652122859027456883350489726869369792
73768948414365738864369507473964882256222183380014600761196859217603234808467455216330411738004331144225926234690588782914907738857587485873982839539032383872570588242765430634715775897215045607136180173005
6284247476294272465755627827634987671952813689135891882449928474159168313033402199946752082914885763458638321334540520574959120620677232969513861229945608607527317854452449865348164169238484889061458509343734
42889814884427321817132172538915345065811438233812058753798086065088089761738828665259363375045454916860026772295912255288458448268665524313013375481224956123768607800770076793954184890714946737785440752830
7872988103912945121292986479370345125743645581459757140827270598362516535296584571123585270211934352981105568398089840949803461850780252738387638404216947273980464304250045030866637032760016314921442805708802
4308505678921086460747551395391198386361671930802781463801369324823327715951805961930659423783608260750887209297297942940456787733811987744468554429480032174105668942371054502887041961191507727390000316420144
742133232938716180295561400460286740042288539854653828018428515122296028759274180162182323698326977441047012533671489615232678349845539496043970535247766213959145192704221223416269986286460352098068622
43548133761493951319428681134865315622871732149764817053818461553265961875302964786011608722634044392257601926496109168851510131439457439830319255715462151424691223705191490978618494361509631099333639594561
796593396851958605338631176324147066842257192394742537126479559749993283247279807896470753054014194909200609712674753186365524032125775785393069730056595208274574994718981444537722482078884433351185456015688
5370818289289521830013965437694728641877665672815389733491594105438614354734613424469206700827824236455574508825667015724275281031714164063141068138433902402728131896088481304066522616955282563718386246494
4295688593938467267236941947557132054600182634257130291153535372880818727302159678708843729341211708451158062969769726660166363527695996902150212210495425956727859318516268447100374434620420033591203733893
095426959021486207390651990180213443342514978962842361985716477388126097434350563250866354726738971298084069719653772779893160200566275058007512407494448163922143981184927482819786551784785477491987141384850
4229038454099570842038137277135667703565084181708520689583136233521692740531015439921761834078817735647464749071616600653230439902639786065590085309872435445689315601329942407112295081545377152105194244551279536
497121487222219372989159833001742397977592530501318837883494884232225073188163994389356278171028754325887945588574278039071166
```

Functions

```
function hello()  
    println(" 你好, 世界 ")  
end
```

Functions

```
function  $\Sigma$ (x...)  
    if (length(x) == 1)  
        return x[1]  
    end  
    x[1] +  $\Sigma$ (x[2:length(x)]...)  
end
```

Functions

```
funcs = [sin, cos, tan]
vals = [1, 2, 3]
apply(f, x) = map((f, x) -> f(x), f, x)

apply(funcs, vals)
```

Numbers

```
# Division
7 / 5
7 ÷ 5
7 % 5
[1, 2, 3, 4, 5, 6] ./ 3
# Complex
(1+2im) * (4-3im)
cos(5+3im)
sqrt(-1) # error
sqrt(-1+0im)
# Rationals
r = 6//5 * 7//10
float(r)
# BigInt
2^BigInt(1200)
# BigFloat
BigFloat(2.0^66) / 3
```

Numbers

```
A = [1 2 3; 4 1 6; 7 8 1]
B = [1.5 2 -4; 3 -1 -6; -10 2.3 4]
```

```
using LinearAlgebra
```

```
function linalgtest(M)
    println(LinearAlgebra.det(M))
    vals, vecs = LinearAlgebra.eigen(M)
    x = M\[1;1;1]
    l, u = factorize(M)
    for y in (l, u, vals, vecs, x)
        Base.show(stdout, "text/plain", y)
        println();
    end
end
```

Types

```
primitive type UInt128 <: Unsigned 128 end
```

```
struct A  
    x :: Int64  
    y  
end
```

```
b = A( 2, "alma" )  
c = A( 3, 6.2 )  
d = A( 2.1, 6 ) # error
```

```
typeof(b.y) # String  
typeof(c.y) # Float64  
typeof(b.y) <: Number # false  
typeof(c.y) <: Number # true
```

Types

```
struct Polynomial{R}
    coeffs::Vector{R}
end

function (p::Polynomial)(x)
    v = p.coeffs[end]
    for i = (length(p.coeffs)-1):-1:1
        v = v*x + p.coeffs[i]
    end
    return v
end
```

Function dispatch

```
f(x, y) = x + y  
f(x :: Number, y :: Number) = x + y  
f(x :: AbstractFloat, y :: Number) = x * y  
f(x :: Number, y :: Integer) = x - y
```

```
f(1, 1) # 0  
f(1.0, 1) # error  
f(1, 1.0) # 2.0  
f(1.0, 1.0) # 1.0  
f("a", "b") # "ab"
```

Functions

```
function syntax(str)
  expr = Meta.parse(str)
  dump(expr)
  function (x)
    eval(Base.Cartesian.lreplace(expr, :x, x))
  end
end

syntax("(x + 4) / 2")
```

Calling C

```
function libctime()  
    t = ccall(:time, Int64, ())  
end
```

```
function sdltime()  
    t = Int64(ccall(:SDL_GetTicks, :libSDL2, UInt32, ()))  
end
```

Calling C

```
function mycompare(a, b)::Cint
    return (a < b) ? -1 : ((a > b) ? +1 : 0)
end

function csort()
    mycompare_c = @cfunction(mycompare, Cint,
        (Ref{Cdouble}, Ref{Cdouble}))
    A = [1.3, -2.7, 4.4, 3.1]
    ccall(:qsort, Cvoid, (Ptr{Cdouble}, Csize_t,
        Csize_t, Ptr{Cvoid}), A, length(A),
        sizeof(eltype(A)), mycompare_c)
    A
end
```

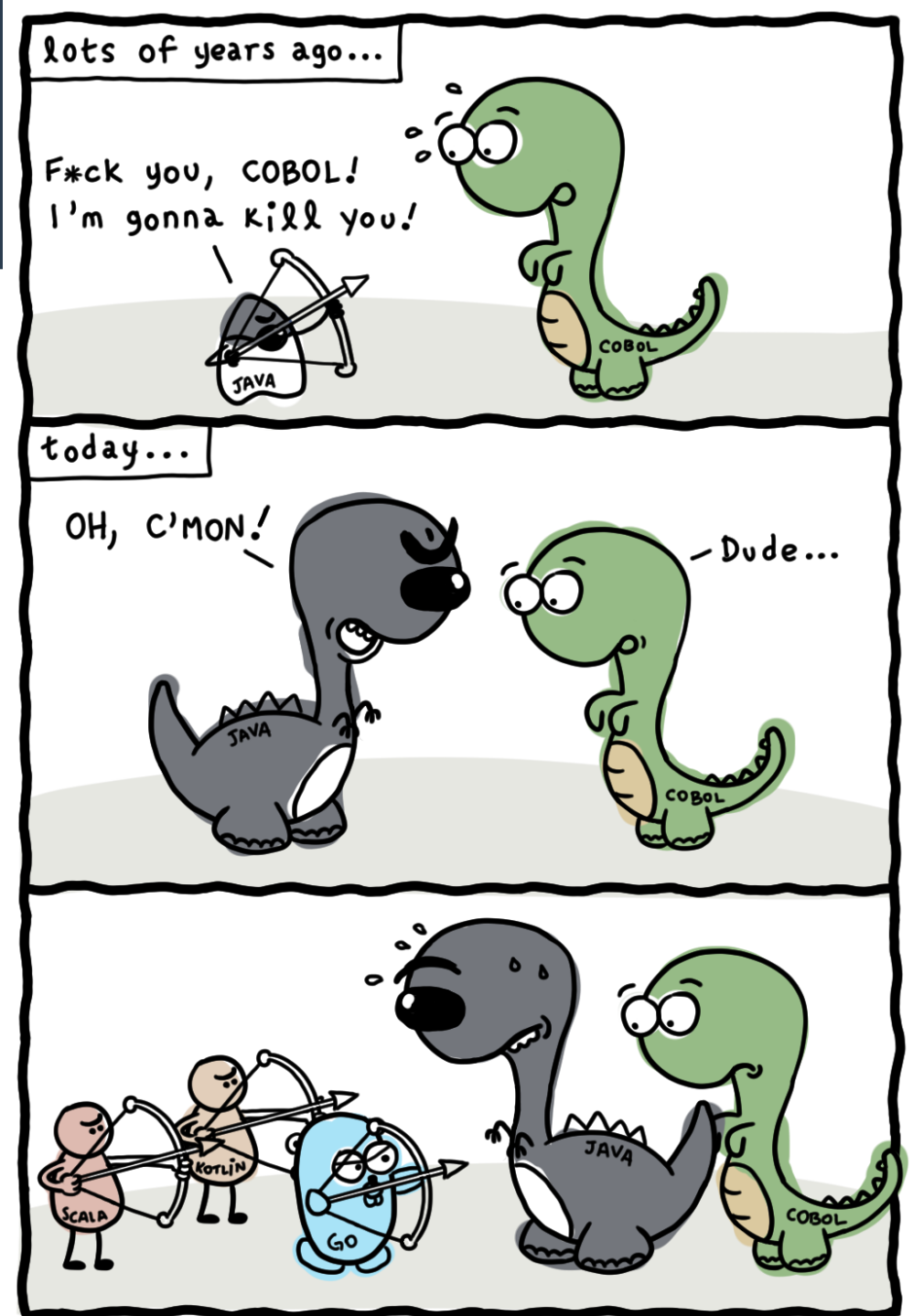
Syntax things

```
x, y = "q", "w"  
"∀ $x ∃ $y"
```

```
alma[1](x, y)[3]
```

Questions?

- **Example codes available at**
<http://cg.elte.hu/~agostons/presentations/julia/examples.jl>



Daniel Stori {turnoff.us}