



Gotta go fast

# Overview

- High level, interactive like MatLab, Python (with Jupyter)
- User friendliness we expect from Python
  - Dynamically typed, sometimes
- (Often) as fast as C/FORTRAN
- Free / Open source
- Over 5,000 community packages and growing (including familiar faces)

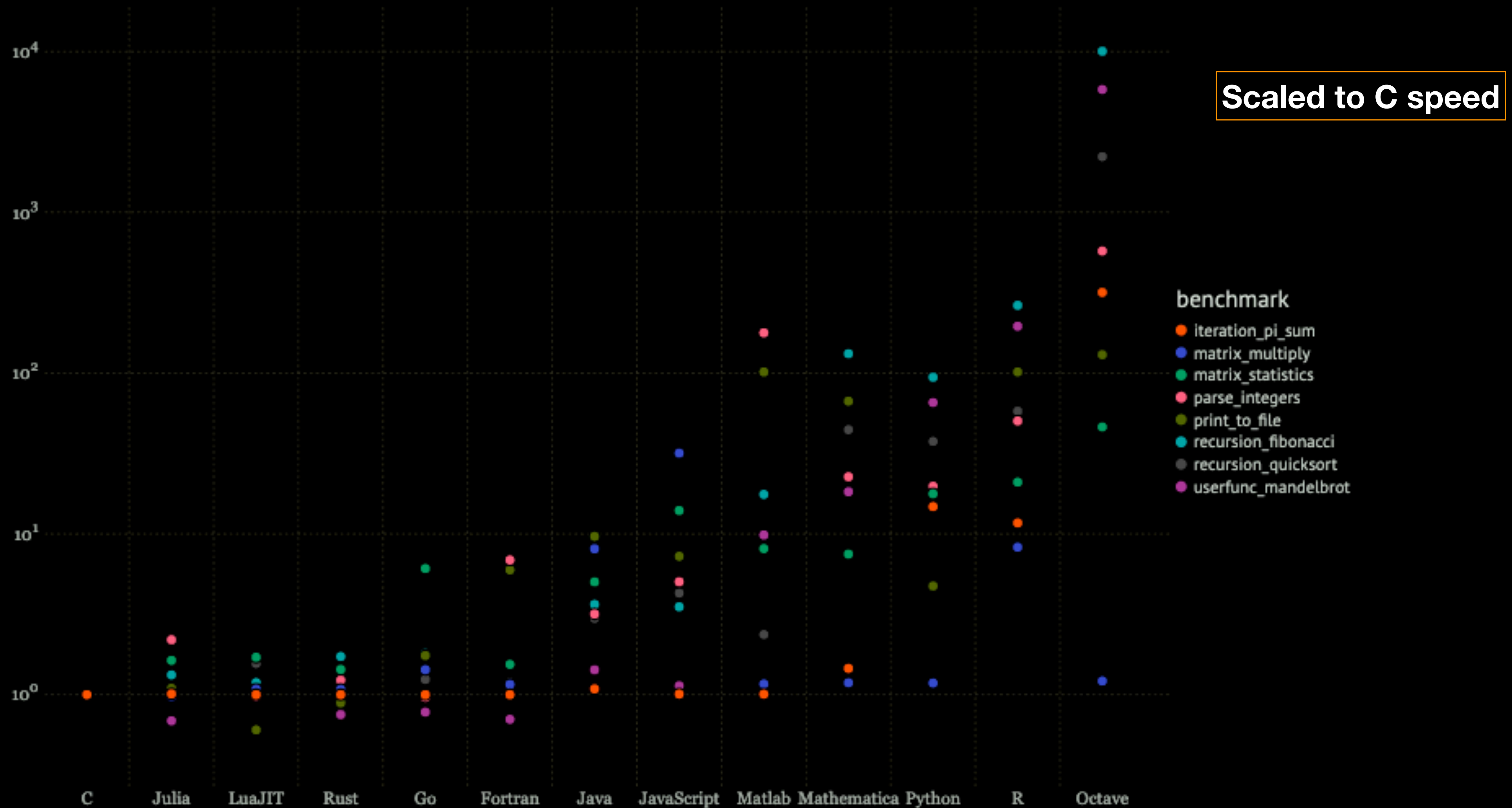
# Features

- Just-In-Time Compilation
  - Like Numba for python, but automatic and hassle-free
- Parallel support
  - Distributed memory (including existing MPI libraries), shared memory, SIMD, and GPU support
- Easily call C, FORTRAN, and Python code from within Julia
- NOT object oriented >:)

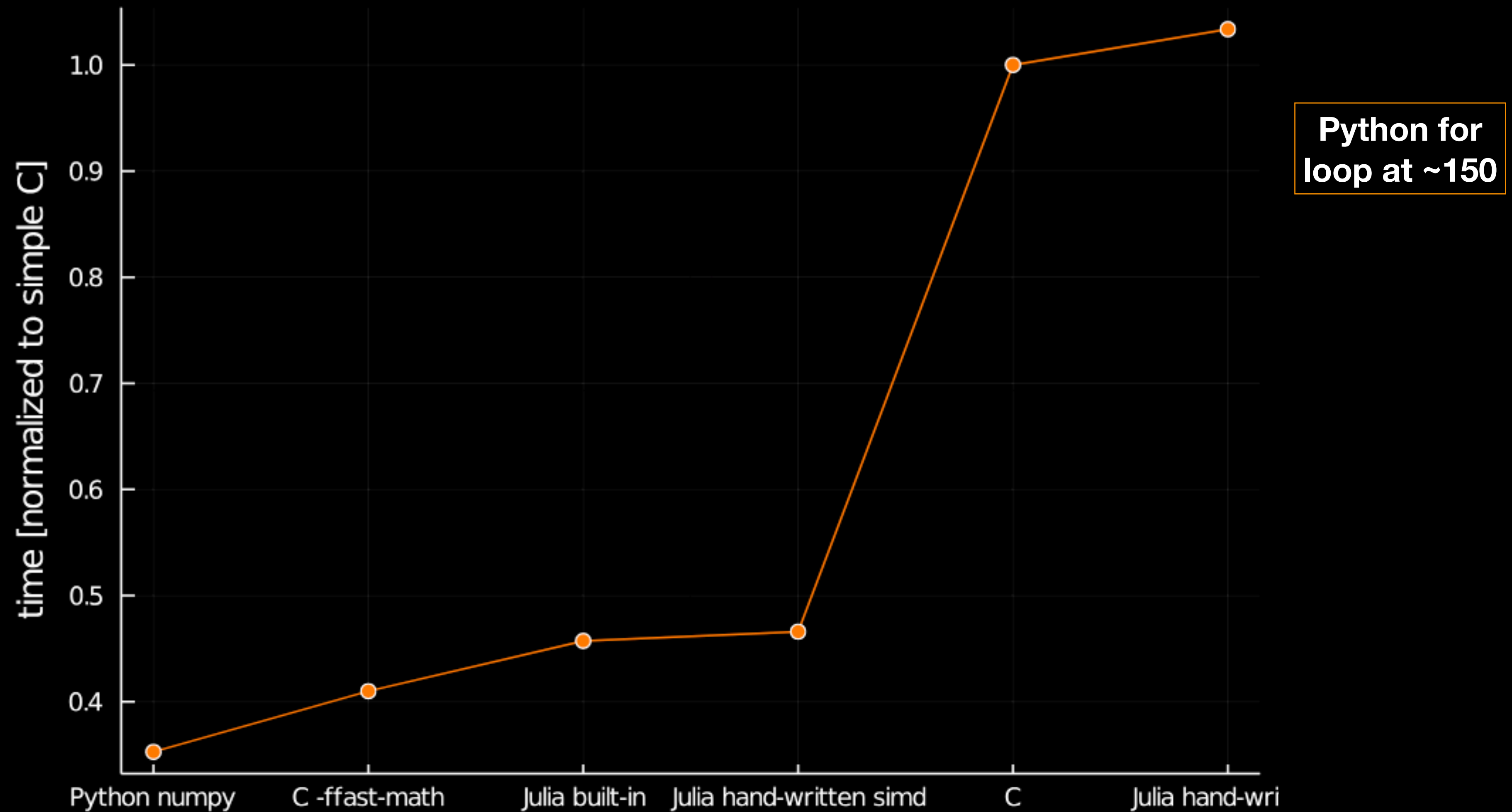
# Lots of familiar conveniences

- HDF5.jl - Essentially h5py. Just as user friendly
- Pandas.jl - Wrapper of Python Pandas library
  - DataFrames.jl - Essentially the same but pure Julia
- PyPlot.jl - Julia Matplotlib wrapper

# Some Benchmarks



# Sum 1e7 rand



# Examples

```
"""
This is okay!
Returns Hilbert matrix of size n
"""
function Hilbert_dynamic( n )
    matrix = zeros( n, n )

    for i in 1:n
        for j in 1:n
            matrix[i,j] = 1.0 / ( i + j - 1.0 )
        end
    end

    return matrix
end
```

**zeros(n,n) like np.zeros((n,n))**

**Indentation doesn't matter!**

**1 based indexing**

**Docstrings**

```
"""
This is better!
Returns Hilbert matrix of size n
"""
function Hilbert_static( n::Int64 )
    matrix::Array{Float64, 1} # or Vector{Float64}

    for i in 1:n
        for j in 1:n
            matrix[i,j] = 1.0 / (i + j - 1.0)
        end
    end

    return matrix
end
```

**FunctionName( var::Type )**

**declare similar to FORTRAN**

# Multiple Dispatch

- Function and operator overloading, but not

- 

```
[julia> methods(+)  
# 184 methods for generic function "+":  
[1] +(x::Dates.Date, y::Dates.Day) in Dates at /Applications/Julia-1.5.app/Contents/Resources/julia/share/julia/stdlib/v1.5/Dates/src/arithmetic.jl:74  
[2] +(x::Dates.Date, y::Dates.Week) in Dates at /Applications/Julia-1.5.app/Contents/Resources/julia/share/julia/stdlib/v1.5/Dates/src/arithmetic.jl:72  
[3] +(dt::Dates.Date, z::Dates.Month) in Dates at /Applications/Julia-1.5.app/Contents/Resources/julia/share/julia/stdlib/v1.5/Dates/src/arithmetic.jl:54  
[4] +(dt::Dates.Date, y::Dates.Year) in Dates at /Applications/Julia-1.5.app/Contents/Resources/julia/share/julia/stdlib/v1.5/Dates/src/arithmetic.jl:27  
[5] +(dt::Dates.Date, t::Dates.Time) in Dates at /Applications/Julia-1.5.app/Contents/Resources/julia/share/julia/stdlib/v1.5/Dates/src/arithmetic.jl:19  
[6] +(B::BitArray{2}, J::LinearAlgebra.UniformScaling) in LinearAlgebra at /Applications/Julia-1.5.app/Contents/Resources/julia/share/julia/stdlib/v1.5/LinearAlgebra/src/uniformscaling.jl:117  
[7] +(a::Float16, b::Float16) in Base at float.jl:398
```

**Free to define new methods for existing functions**



# Using and Running Julia

- Download Julia (might be in HomeBrew? I just get from website), alias executable, possibly add to path
- Like running “python” at command line, can run Julia to open REPL
- “julia code.jl” executes code
- Supported in Jupyter notebooks by default!
- Pluto.jl
  - Like Jupyter, but real-time updates cell dependencies

# Cons?

- Fewer mature libraries / ports
  - Exs: built in (not PyPlot) plotting still lacking some controls, no yt port
- Can be a little syntactically burdensome if workflow is python library heavy.
- Short / “simple” scripts may be slower than python eqv. from compile
  - “np.loadtxt and plot script” equivalent will just be slower
- Harder to share code — for now!
  - Used to be that people were hesitant to switch Perl -> Python!

# Pros

- It's \*\*\*\*\* fast
- Even writing “bad” Julia code, you can compete with “good” Python/C
  - No need to fuss with Numba
  - No need to spend time planning how to avoid Python's optimization issues
- Growing quickly
- Just as “easy” to write as Python, some syntactical obstacles
- Integrates into analysis workflow (Jupyter support)