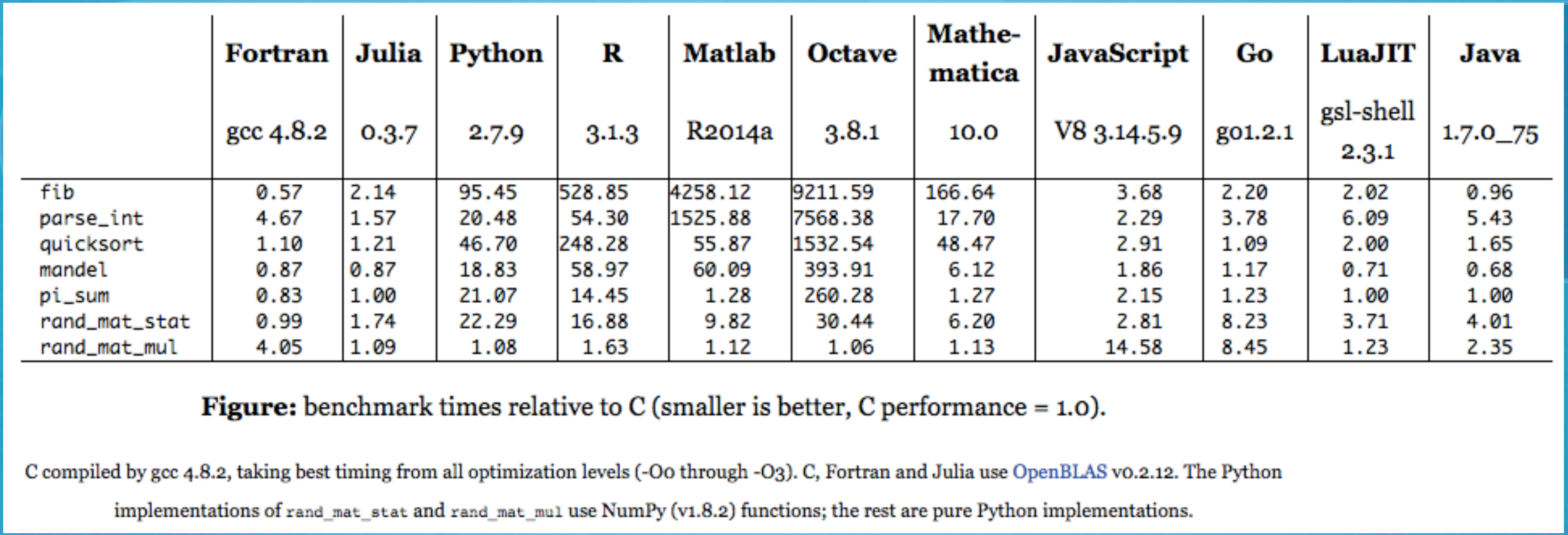
Why Julia ?

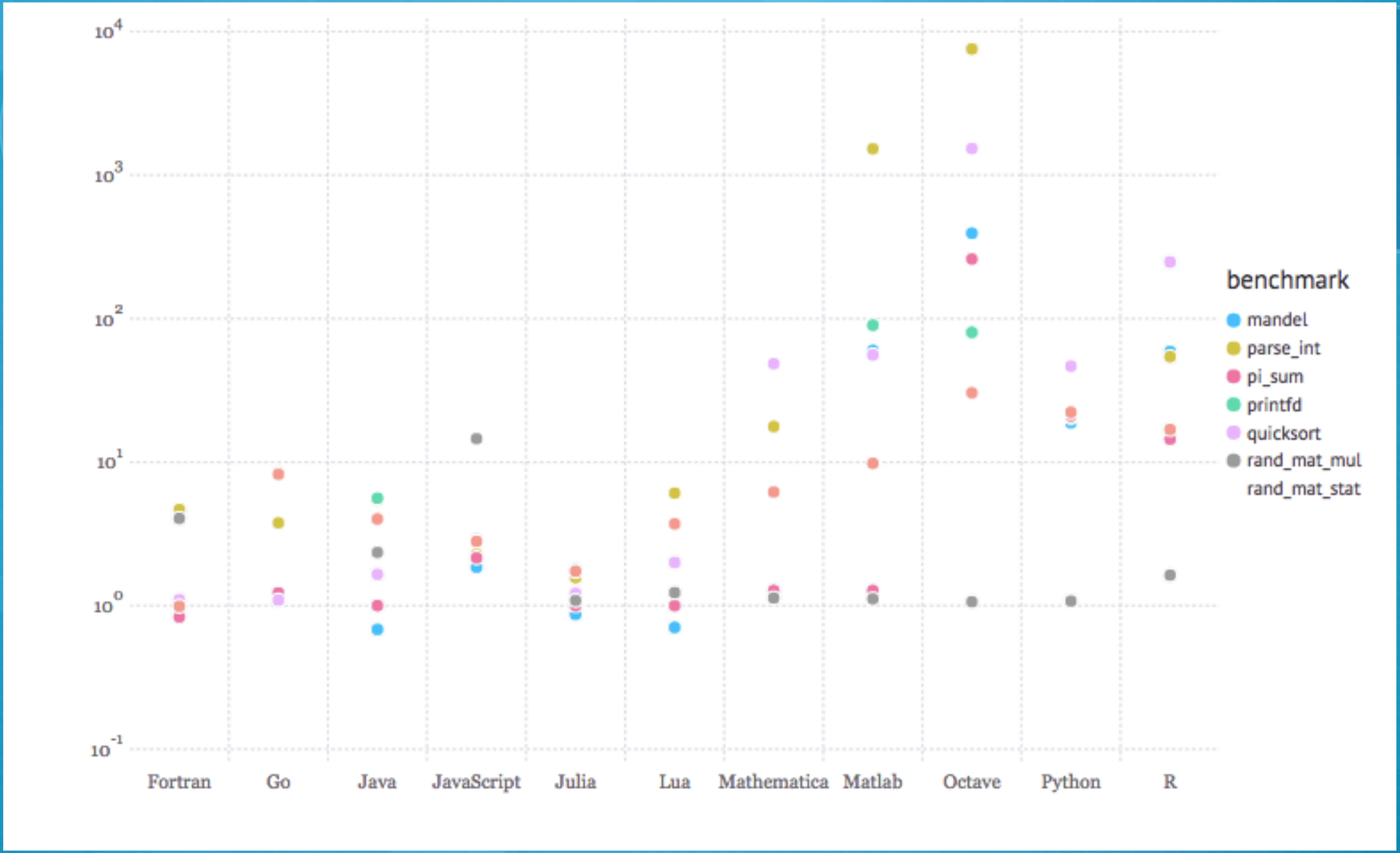
Alex Codoreanu, June 15th 2015

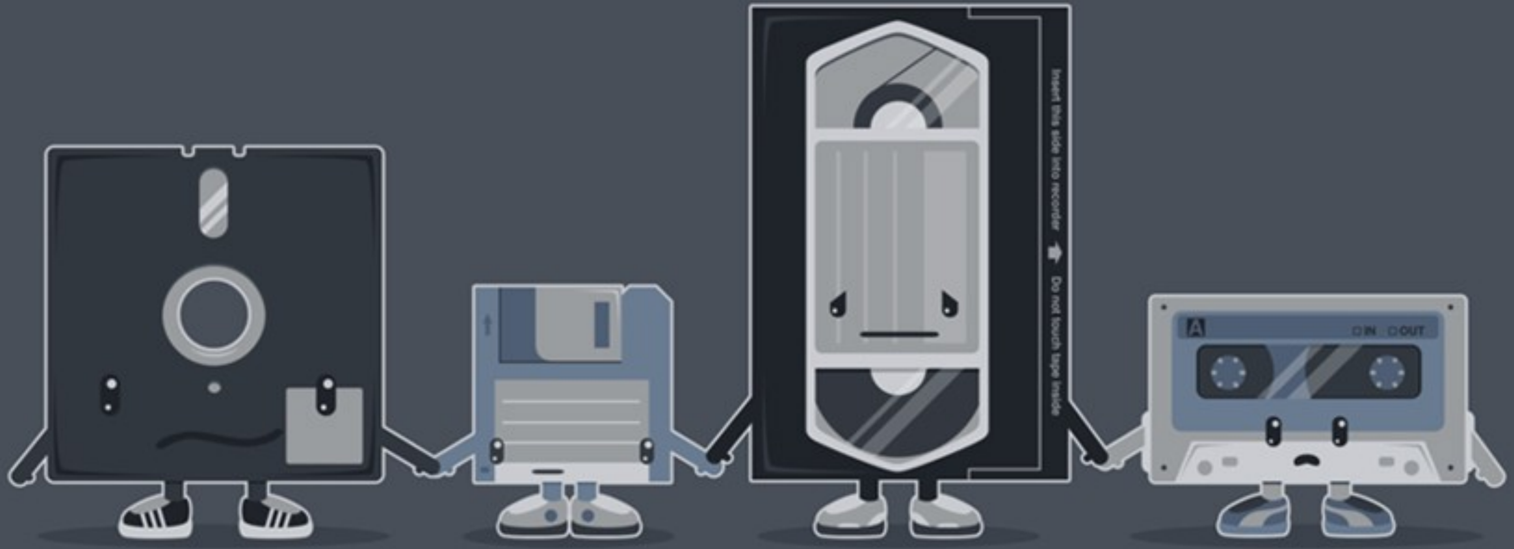


	Fortran gcc 4.8.2	Julia 0.3.7	Python 2.7.9	R 3.1.3	Matlab R2014a	Octave 3.8.1	Mathematica 10.0	JavaScript V8 3.14.5.9	Go go1.2.1	LuaJIT gsl-shell 2.3.1	Java 1.7.0_75
fib	0.57	2.14	95.45	528.85	4258.12	9211.59	166.64	3.68	2.20	2.02	0.96
parse_int	4.67	1.57	20.48	54.30	1525.88	7568.38	17.70	2.29	3.78	6.09	5.43
quicksort	1.10	1.21	46.70	248.28	55.87	1532.54	48.47	2.91	1.09	2.00	1.65
mandel	0.87	0.87	18.83	58.97	60.09	393.91	6.12	1.86	1.17	0.71	0.68
pi_sum	0.83	1.00	21.07	14.45	1.28	260.28	1.27	2.15	1.23	1.00	1.00
rand_mat_stat	0.99	1.74	22.29	16.88	9.82	30.44	6.20	2.81	8.23	3.71	4.01
rand_mat_mul	4.05	1.09	1.08	1.63	1.12	1.06	1.13	14.58	8.45	1.23	2.35

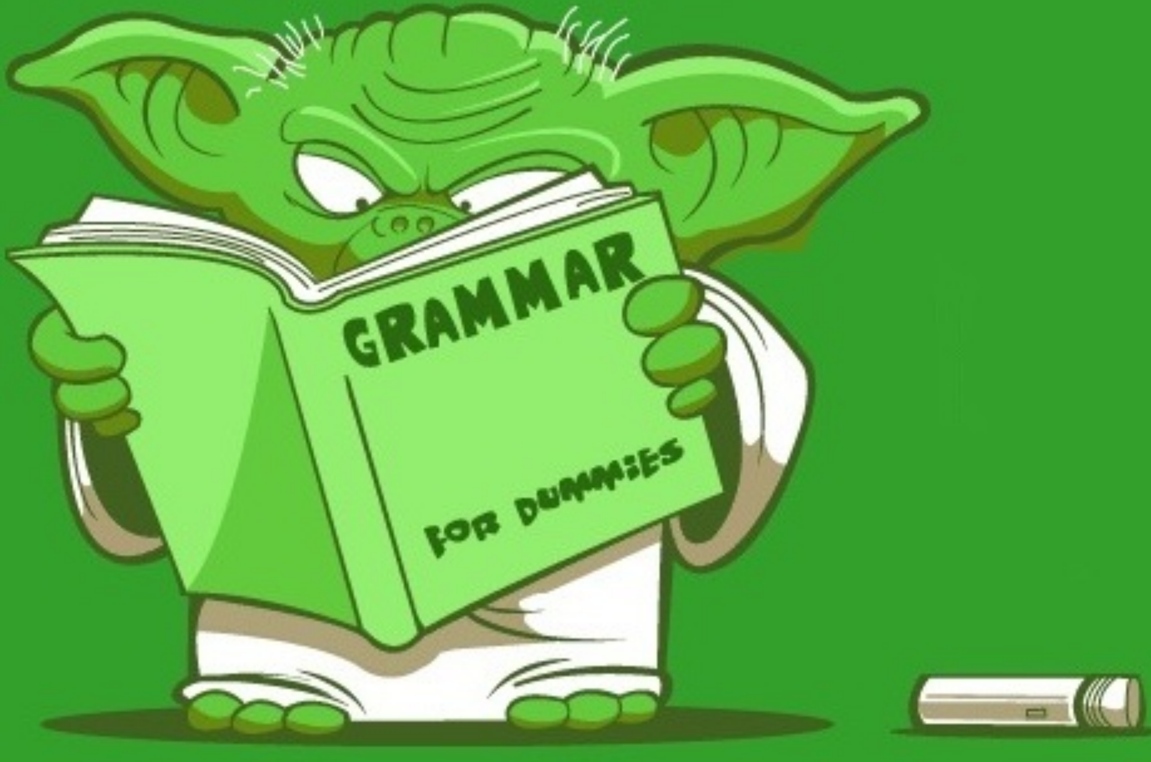
Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

C compiled by gcc 4.8.2, taking best timing from all optimization levels (-O0 through -O3). C, Fortran and Julia use OpenBLAS v0.2.12. The Python implementations of `rand_mat_stat` and `rand_mat_mul` use NumPy (v1.8.2) functions; the rest are pure Python implementations.

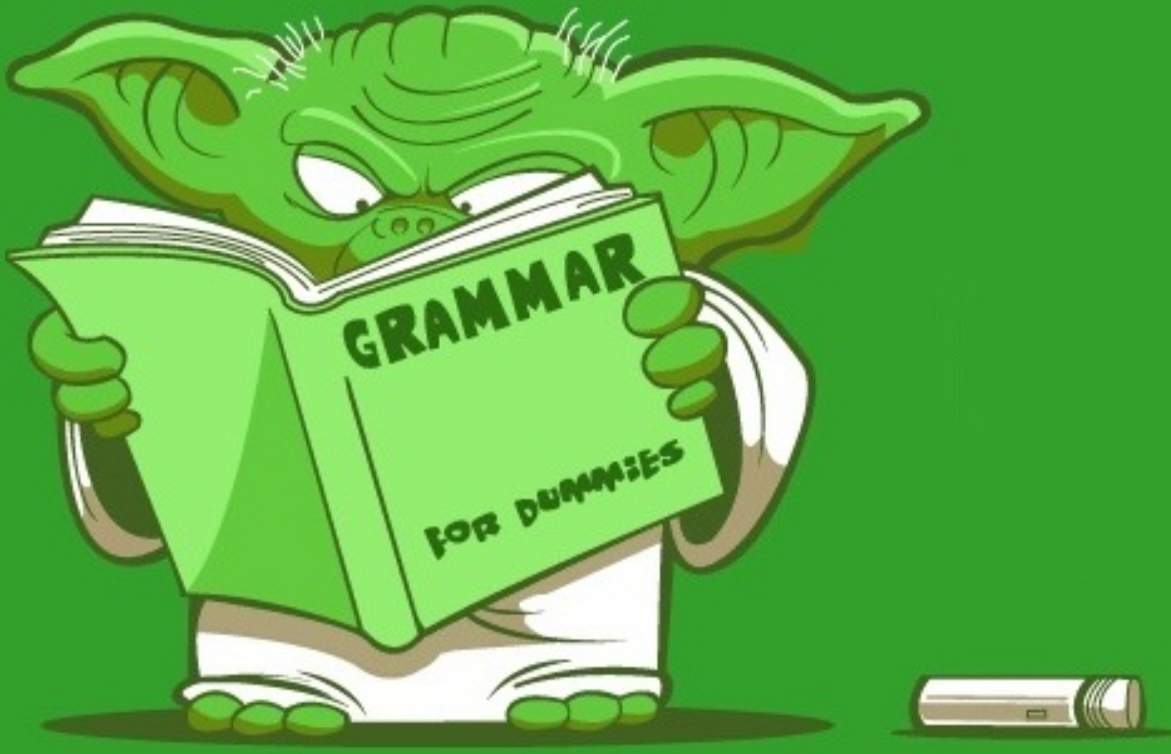

10^4
10^3
10^2
10^1
10^0
10^-1
Fortran
Go
Java
JavaScript
Julia
Lua
Mathematica
Matlab
Octave
Python
R
benchmark
mandel
parse_int
pi_sum
printfd
quicksort
rand_mat_mul
rand_mat_stat



NEVER FORGET

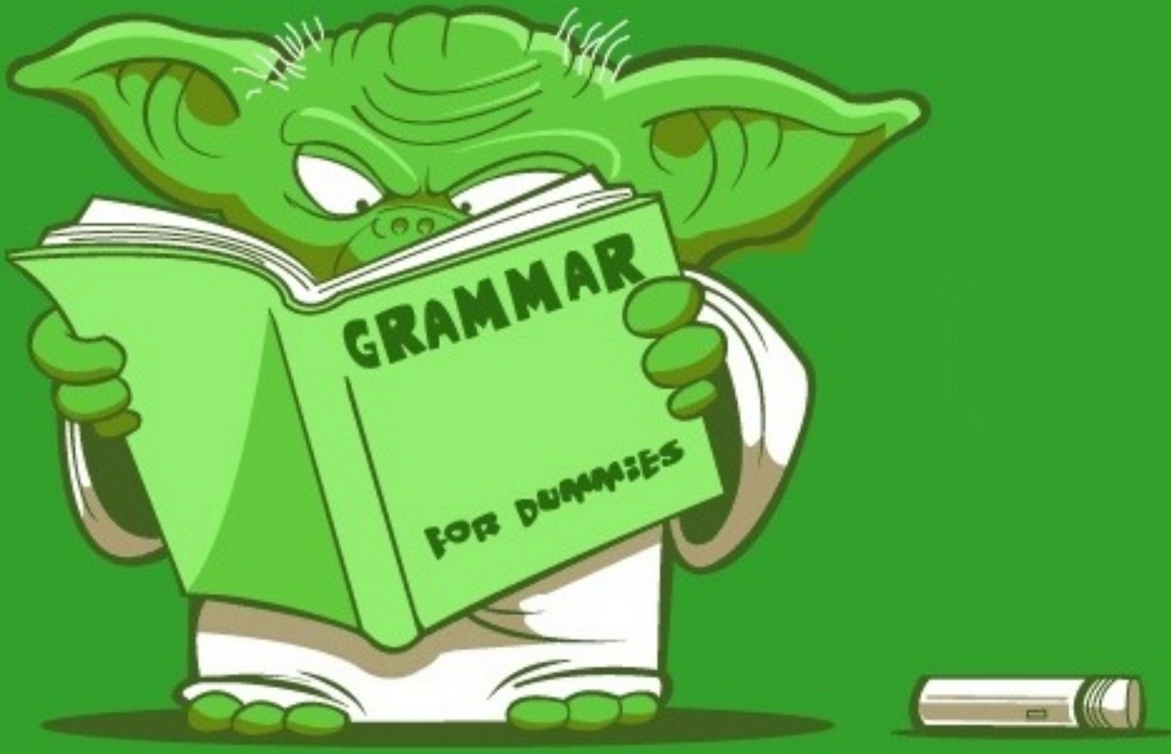

GRAMMAR
FOR DUMMIES

High Level Syntax



High Level Syntax

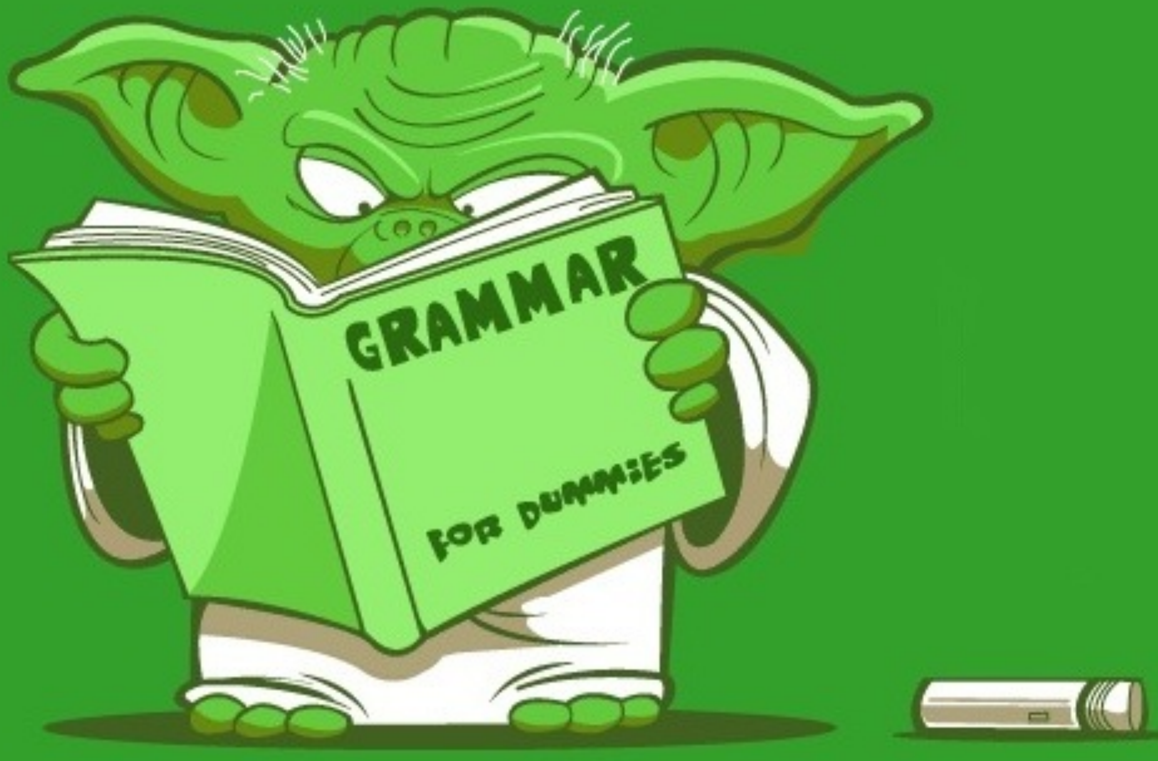
Just In Time Compiled



High Level Syntax

Just In Time Compiled

Ijulia Notebook

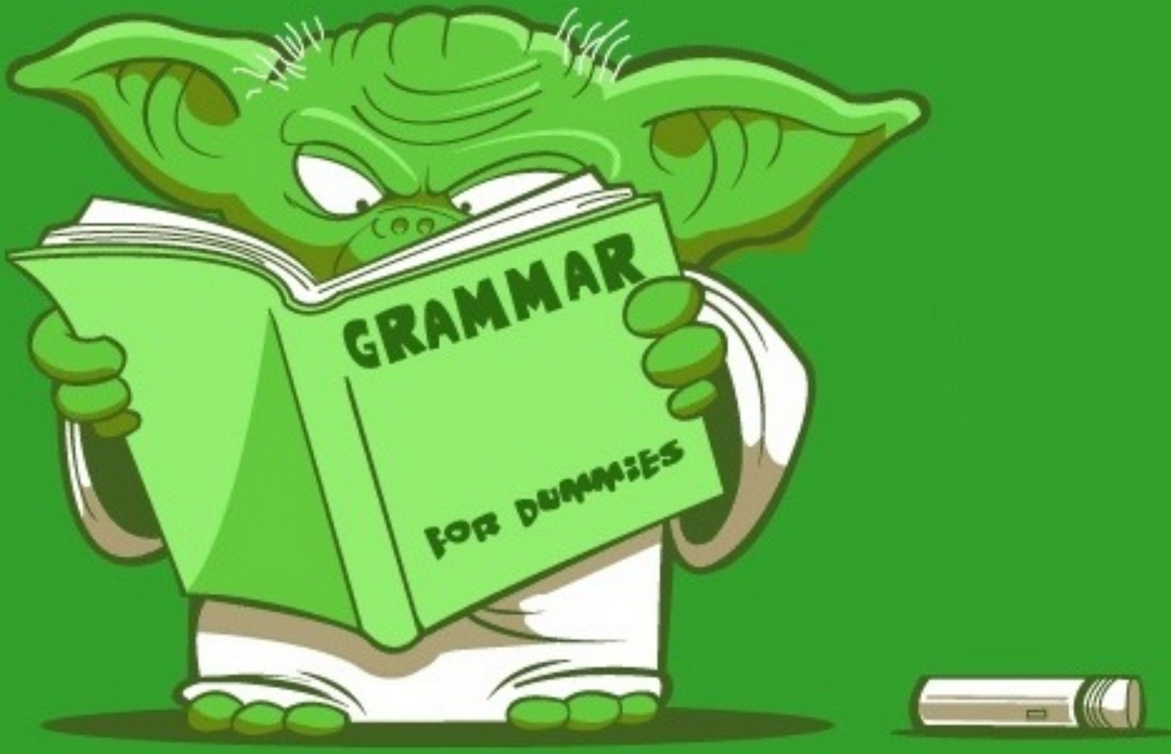


High Level Syntax

Just In Time Compiled

Ijulia Notebook

Dynamic Interpreter



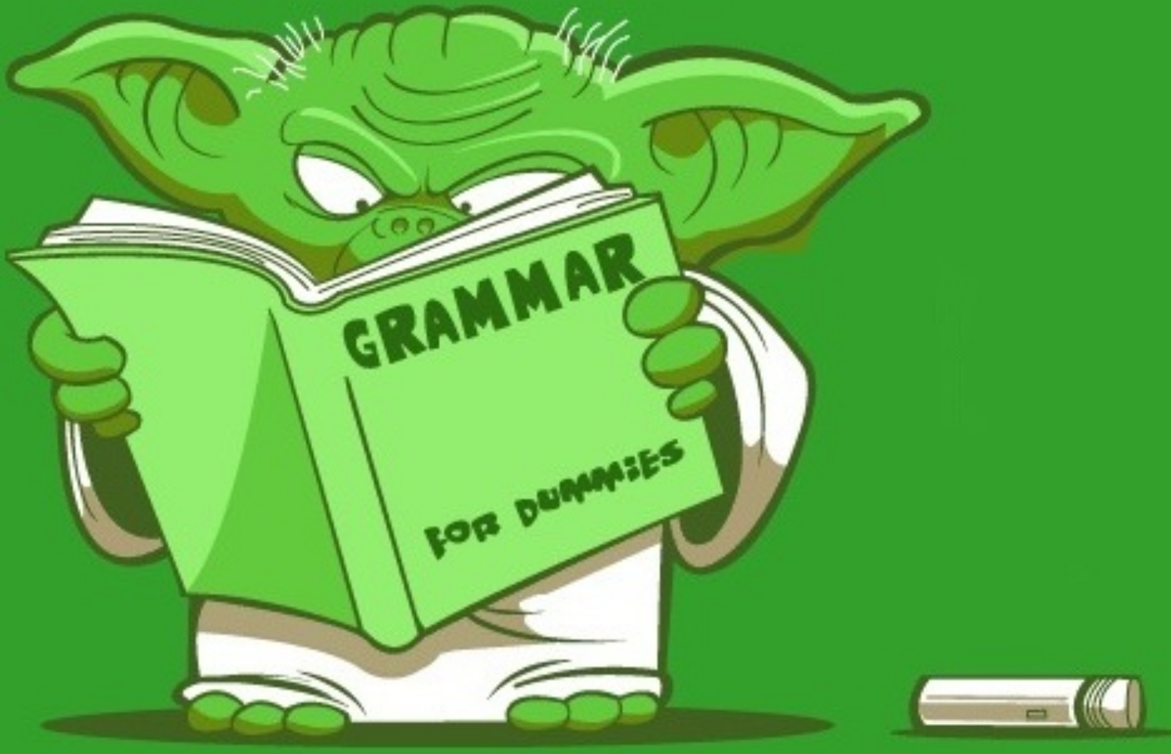
High Level Syntax

Just In Time Compiled

Ijulia Notebook

Dynamic Interpreter

LateX AutoComplete



High Level Syntax

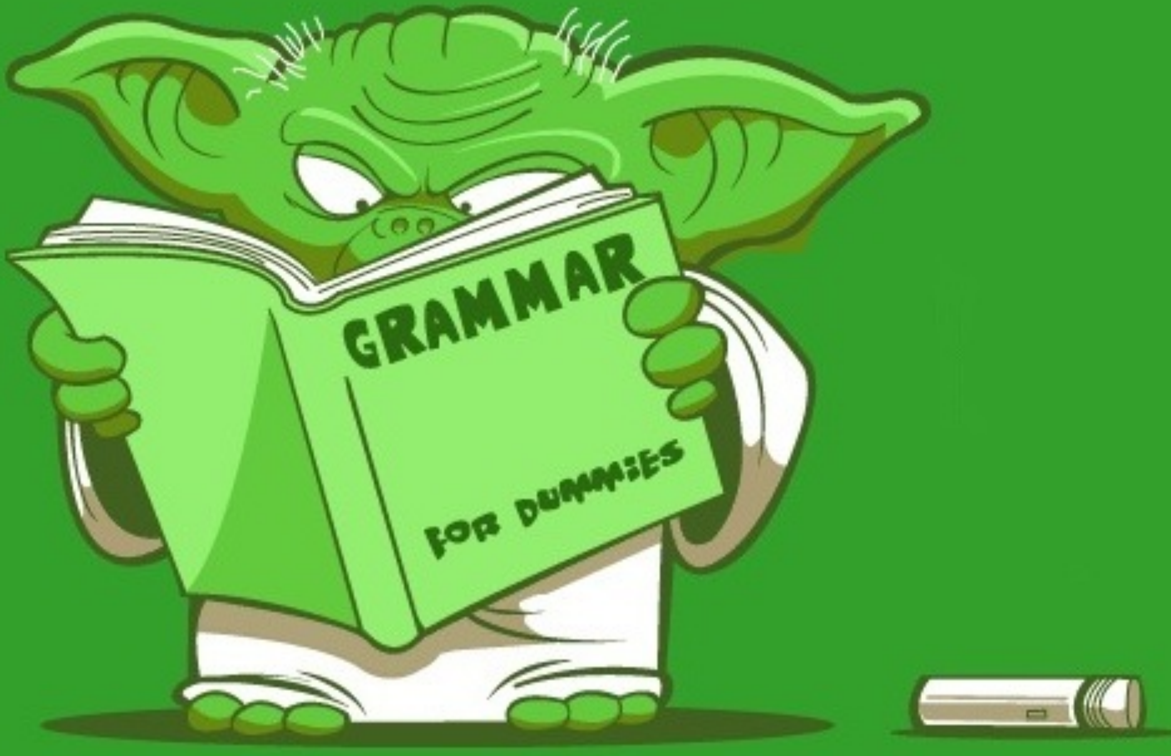
Just In Time Compiled

Ijulia Notebook

Dynamic Interpreter

LateX AutoComplete

Dynamic Type Promotion



High Level Syntax

Just In Time Compiled

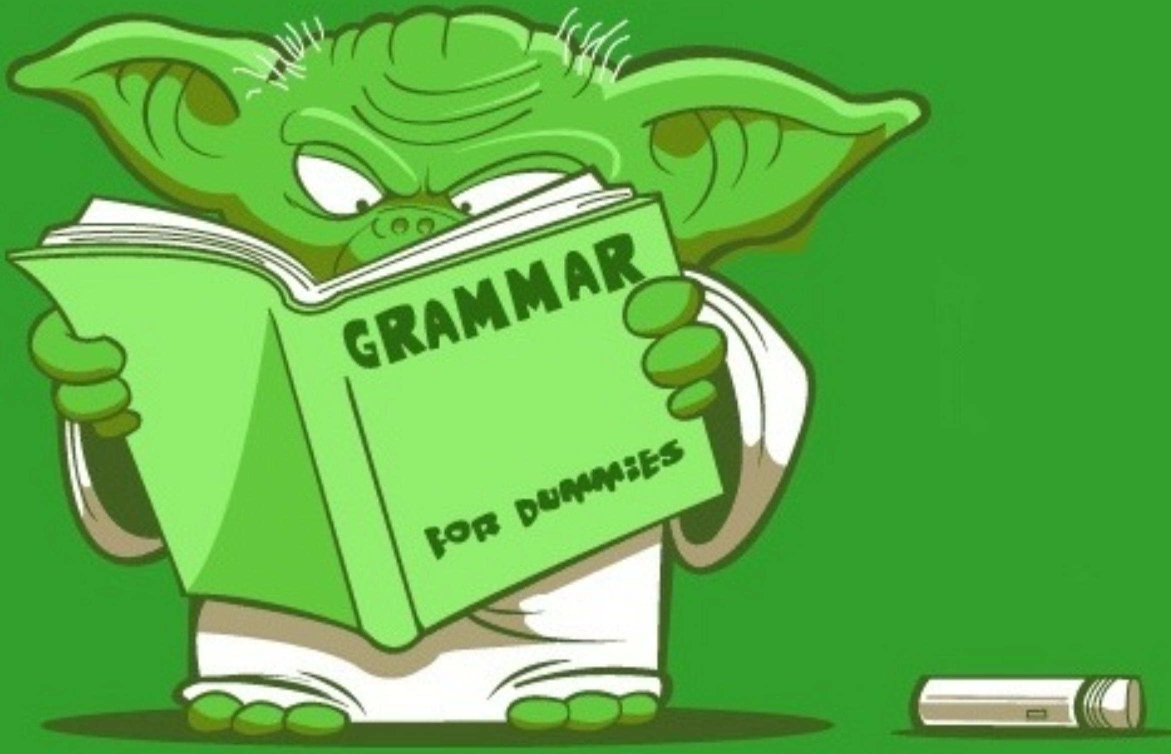
Ijulia Notebook

Dynamic Interpreter

LateX AutoComplete

Dynamic Type Promotion

Very very familiar



Julia can feel and look a bit as an impostor

Julia can feel and look a bit as an impostor

Designed to directly interact with and access Python modules

Using PyCall

```
@pyimport  numpy as np
```

Julia can feel and look a bit as an impostor

Designed to directly interact with and access Python modules

Using PyPlot

now you can use all of your matplotlib knowledge !!!

Julia can feel and look a bit as an impostor

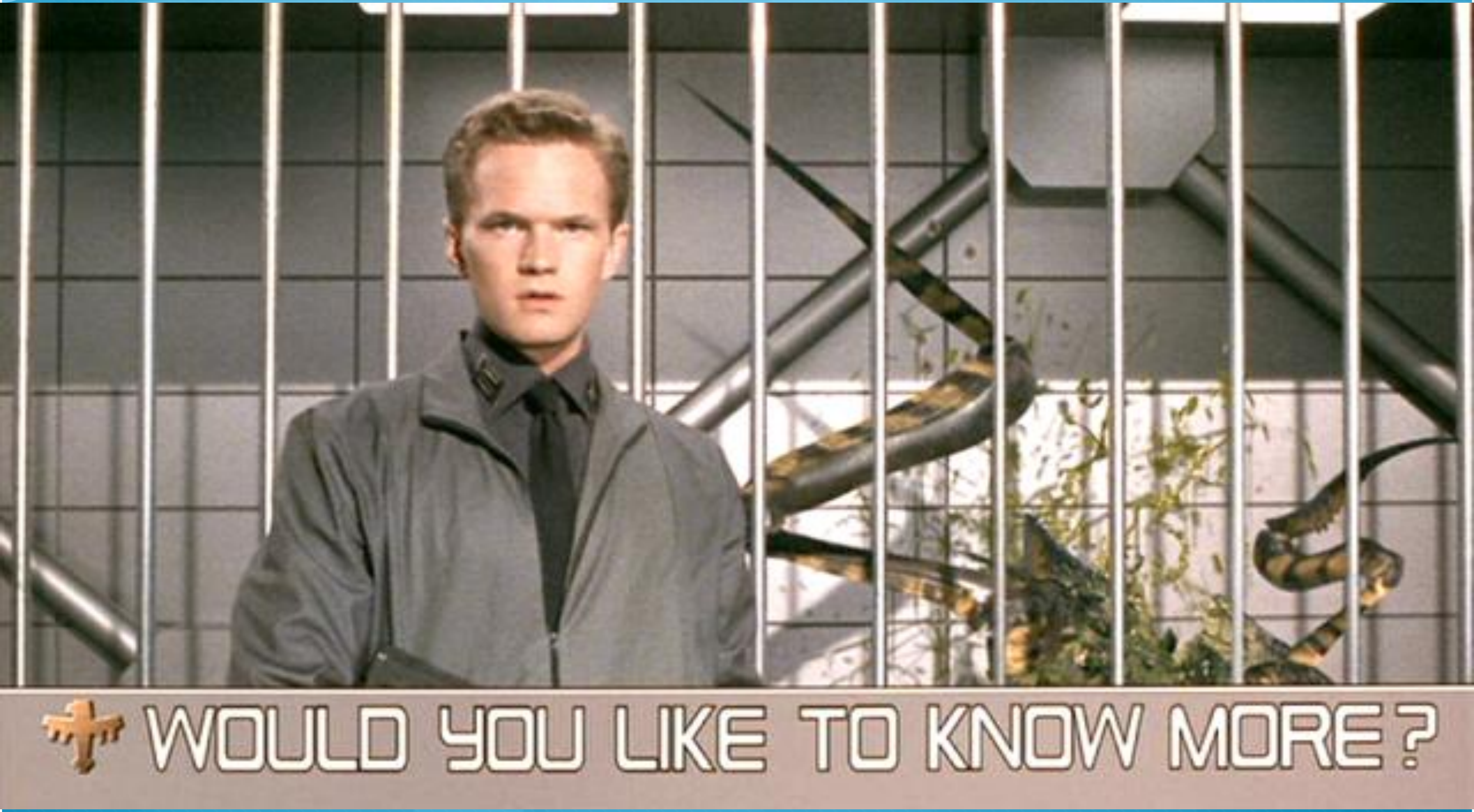
Designed to directly interact with C using the native function ccall() which has a dynamic return type. Whatever your C function returns becomes the return type.

Julia can feel and look a bit as an impostor

Designed to directly interact with R as well

using RCall

adds R functions to DataFrames which support NA type


WOULD YOU LIKE TO KNOW MORE?

Advanced Manufacturing and Design Centre - Room 206

Thursday 18th June from 3:30-5:30pm