# RYANAIR LABS Data & Analytics

## A not so standard introduction to


David García Heredia
garciaherediad@ryanair.com

Madrid, 16/Sept/2021

# SO, WHAT'S THE PLAN?

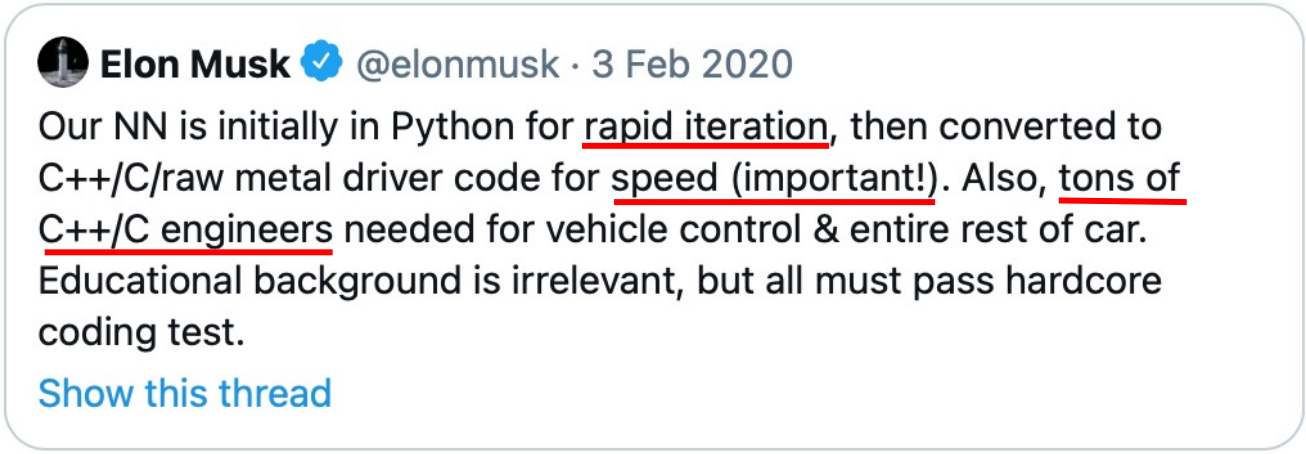
RYANAIR LABS
Data&Analytics

1. Today we will just discuss about the essence of the Julia programming Language. We will try to answer questions such as:
    1. Why Julia?
    2. What are the features of the language?
    3. What makes Julia different from C++, Python, R...?
    4. What resources are available to learn Julia?
    5. What are the tips to get the most from Julia?

2. Plan for the (potential) next session: Just coding! We will illustrate some of the most important features to start writing good code in Julia.

# JULIA, WHAT FOR?





# JULIA, WHAT FOR?



# FEATURES OF THE LANGUAGE



1. Multi-dispatch and more:
    1. The function to be used is selected based on all its arguments, not just the 1st one.
    2. Functions are 1st class citizens (everything is about functions).
    3. Inheritance, composition and design patterns are still available as in OOP languages.

2. Dynamic type system:
    1. It feels like other scripting languages.
    2. But types can be specified as in static type systems. This is one of the keys for high-efficient code.

3. REPL (read-eval-print loop) with different modes:
    1. Julian mode (prompt): To execute code in an interactive way.
    2. Pkg mode: To install Pkgs and handle environments.
    3. Help mode: To quickly access the documentation of the methods.
    4. Shell mode: To execute system commands.

4. Reproducible environments that make the language cross-platform.

5. Personal opinion: As Julia is posterior to R, Python, Matlab… You will see that in a lot of occasions it:
    1. Takes the best features of each language (LISP macros, Python list comprehension, Matlab matrix approach…).
    2. Improves what could be improved.

# FEATURES OF THE LANGUAGE



6. Just In Time (JIT) compilation process:

    1. The first time you call a function, this is compiled. As consequence, the first call to a function is always way much slower (by several orders of magnitude) than posterior calls. A lot of effort is being put to improve this.

    2. When you cannot afford the compilation time in the 1st execution, check: https://github.com/JuliaLang/PackageCompiler.jl

    3. As a compiled language, the more info you give to the compiler, the faster will run your code. (Actually not, the magic of Julia is that it deduces most of the info required by the compiler for you. That way you achieve a Python-like code with a C performance. However, for beginners I recommend being conservative and give “a lot” of information to the compiler if you are interested in getting a really efficient code.)

# FEATURES OF THE LANGUAGE



7. Speed[1]:

    1. Opposite to other popular languages, Julia was born for scientific computing and with performance in mind.

    2. It was also born to develop numerical algorithms in the language (e.g., Matlab performs really well with matrices, but its routines are written in C).

    3. As we will see, with Julia you can achieve code that:
        1. It is way less verbose than C/C++, but runs on the same order of magnitude.
        2. However, the latter is only true if coded with care. If you use bad practices usually employed in scripting languages (e.g., type instability, bad memory management, etc) Julia does not do magic.

[1] It is 1 of the 5 programming languages which has passes the 1.5 PetaFLOP/s mark: https://arxiv.org/pdf/1801.10277.pdf

# BENCHMARK

RYANAIR LABS
Data&Analytics


Full description in: https://julialang.org/benchmarks/

# LANGUAGE COMPARISON

RYANAIR LABS
Data&Analytics

## C++

Pros:

1. High performance.
2. Benefits of OOP.

Cons:

1. Number of Lines Of Code.
2. Compilation in different OS.
3. Linking Libraries.

## Python, R, Matlab…

Pros:

1. Ease of usage.
2. Easier to write clean code.
3. Cross-platform.
4. Easy to link libraries/Pkgs.

Cons:

1. Code written in these langs usually performs very bad (e.g., nested for-loops).

## Julia

Pros:

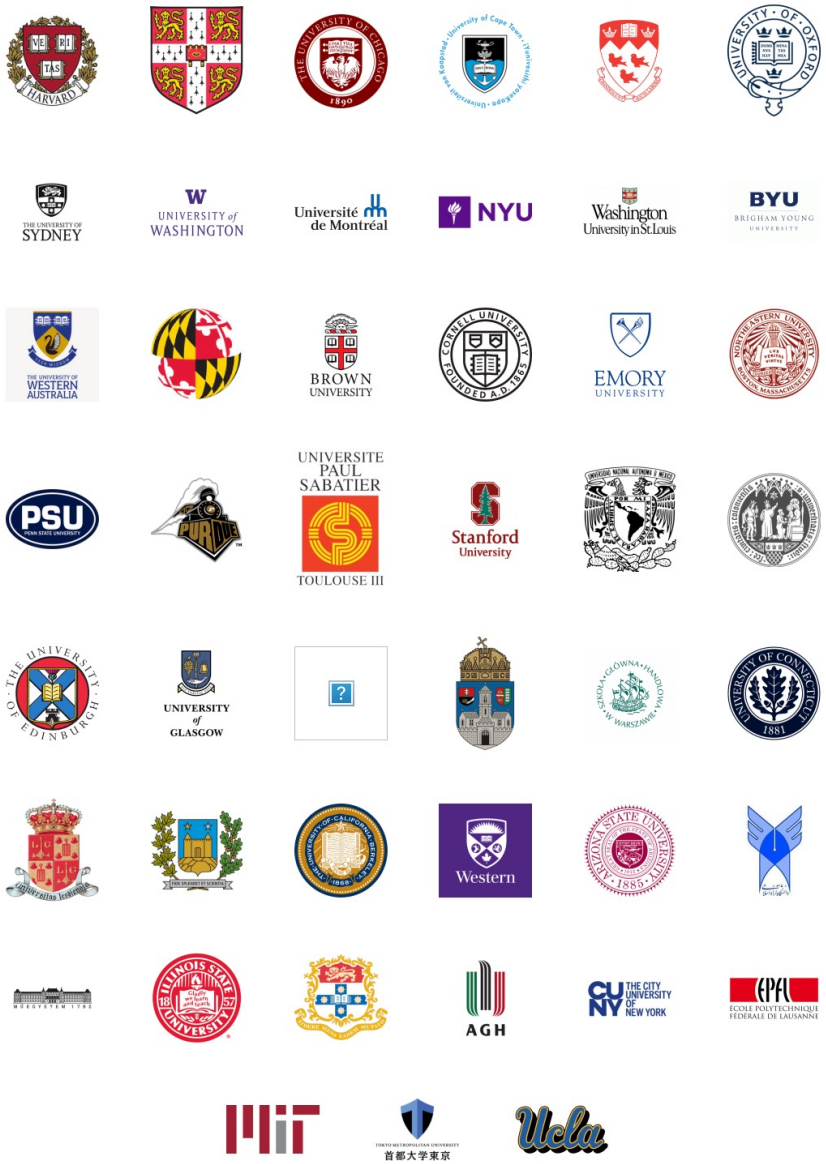
1. High performance if code with some care.
2. Benefits of Python, R…
3. Benefits of Multi-dispatch.

Cons:

1. Compilation time (time-to-first-plot).
2. Documentation of some Pkgs is not always good.
3. Compilation errors are very frustrating when learning Julia.

# WHO IS USING JULIA?

RYANAIR LABS
Data&Analytics

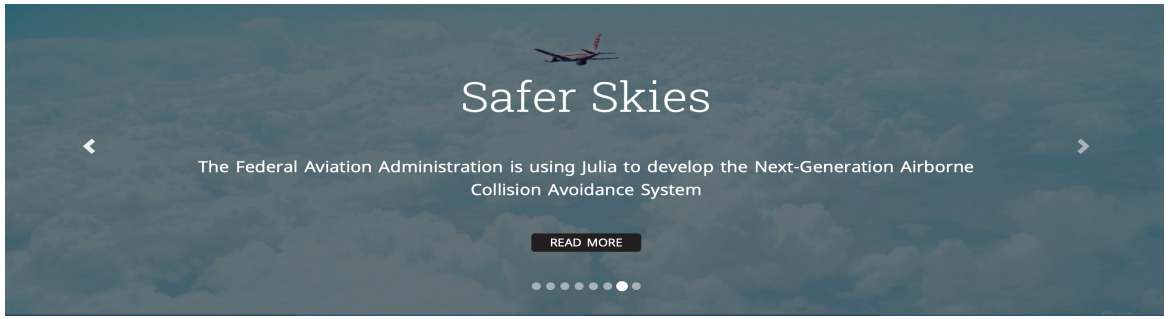




See more in: https://juliacomputing.com/case-studies/thomas-sargent/

# WHO IS USING JULIA?

RYANAIR LABS
Data&Analytics




Julia computing

PRODUCTS INDUSTRIES CASE STUDIES EVENTS TRAINING MEDIA RESOURCES BLOG

JULIA USERS AND JULIA COMPUTING CUSTOMERS

intel, Disney, BNDES (O BANCO DO DESENVOLVIMENTO DE TODOS OS BRASILEIROS), amazon, Capital One, KPMG, Google

Lincoln Laboratory, Microsoft, IBM, BlackRock, NASA

Ford, Oak Ridge National Laboratory, AstraZeneca, Federal Aviation Administration, Aviva, Western Asset, abbvie

Chevron, ExxonMobil, Federal Reserve Bank of New York, gsk, JPMorgan Chase & Co., Pfizer, moderna

# RESOURCES FOR A BETTER UNDERSTANDING OF JULIA

RYANAIR LABS
Data&Analytics

1. Multi-dispatch and how it permits more code reutilization than OOP: https://www.youtube.com/watch?v=kc9HwsxE1OY

2. Design patterns in Julia: https://www.youtube.com/watch?v=nkSuEkmsB28

3. What is bad about Julia (by Jeff Bezanson, one of the creators of Julia): https://www.youtube.com/watch?v=TPuJsgyu87U

4. JIT compilation and benefits for developers (by Christopher Rackauckas): http://www.stochasticlifestyle.com/like-julia-scales-productive-insights-julia-developer/

5. Julia vs Cython and Numba: http://www.stochasticlifestyle.com/why-numba-and-cython-are-not-substitutes-for-julia/

6. Your code = The pseudo-code of your paper: https://www.youtube.com/watch?v=ofWy5kaZU3g

# RESOURCES TO LEARN JULIA

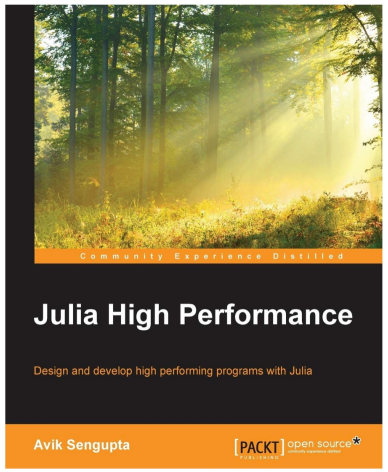
RYANAIR LABS
Data&Analytics

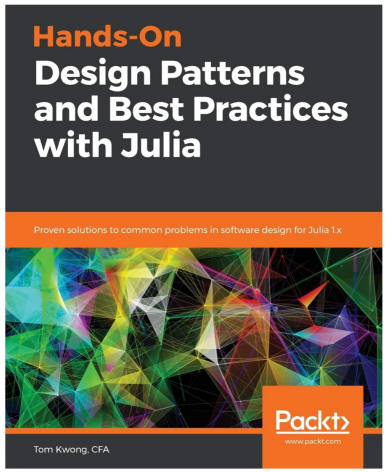
1. I personally learned it reading the official documentation (it is very well written): https://docs.julialang.org/en/v1/

2. You also have tutorials in:
   1. The official YouTube Channel: https://www.youtube.com/user/JuliaLanguage
   2. The YouTube Channel of Julia Computing: https://www.youtube.com/c/JuliaComputing/videos
   3. Julia academy (I don't really like the videos here, but maybe you do) https://juliaacademy.com

3. How to create Pkgs: https://www.youtube.com/watch?v=QVmU29rCjaA

4. Books (in the order I recommend to read them):
   1. Julia High Performance, by Avik Sengupta.
   2. Hands-on Design Patterns and Best Practices with Julia, by Tom Kwong.




This book is extremely good.

# INTERESTING PKGS AND COMMUNITIES

RYANAIR LABS
Data&Analytics

First of all, some words of wisdom: Be careful with versions 0.X of the Pkgs. If they go to version 1.0, some major changes may be introduced that break your code! It does not have to happen, but it can. This is why it is so important to work with environments! So this does not occur to you.

Pkgs for documentation/Reports
- Documenter.jl
- Weave.jl

Pkgs to create/handle Pkgs:
- PkgTemplates.jl
- Revise.jl
- DrWatson.jl

Others:
- JuliaGPU
- BenchmarkTools.jl
- LightGraphs.jl
- Pluto.jl
- Extensions for IDEs: VS Code is the one with more support.

Pkgs for Math Optimization:
- JuMP.jl
- HiGHS.jl
- COSMO.jl
- Coluna.jl

Pkgs for Plots:
- Plots.jl
- Gadfly.jl
- VegaLite.jl
- PlotlyJS.jl

Pkgs for Data Science & Machine Learning:
- DataFrames.jl
- Queryverse.jl
- Flux.jl
- JuliaStats
- AutoMLPipeline.jl
- ModelingToolkit.jl

Don’t simply check the GitHub repository in the links, but the documentation and official sites!

Find Pkgs by organizations: https://julialang.org/community/organizations/

Find Pkgs in official web: https://juliapackages.com/

# ABOUT GETTING HIGH PERFORMANCE



There is no free lunch. If you do not code following some rules, Julia will not make your code fast. Some basic points to remember that will give you a lot are:

1. Use functions all the time:
   i. The REPL is good for quick checks, but it is a global environment, so it kills performance.
   ii. Do not write scripts as in R or Python, but encapsulate all that code into functions.
   iii. Avoid global variables. If you can't, use keyword "const" to promise the compiler that the type (not the value) will not change.

2. Specify types:
   i. For the arguments and output of a function.
   ii. Inside a function, if you do not want to specify types, at least promise to the compiler that the type of the variable will not change (keyword "local").

3. Type stability:
   i. Obvious case: If a variable holds a string, then do not assign a matrix to it.
   ii. Not so obvious: If a variable holds an integer, do not transform it into a Float (e.g., with a division).

4. Memory management: In HPC, computation is free, memory management is crucial!
   i. Instead of creating an array over and over again, create it once and rewrite it.
   ii. Instead of pushing values to an array, try to reserve memory beforehand. Size hints for Sets and Dictionaries are also very valuable!
   iii. Prefer references over copies.

# ABOUT GETTING HIGH PERFORMANCE



1. The rules of the previous slide can be summarized in: The more info you give to the compiler, the happier this is and the faster runs your code. In other words, although Julia is a dynamic-type language, performance is gained when approaching it through the lens of a static one.

2. With experience you will learn that you can skip giving to the compiler some information (it will deduce it for you, this is part of the magic of Julia!!!). However, at the beginning, I do not recommend that.

3. If you follow the 4 previous rules, you will be writing pretty fast code. If you want more:
    1. Check the tips of the manual: https://docs.julialang.org/en/v1/manual/performance-tips/
    2. Prefer loops over vectorized code.
    3. Useful macros to have in mind: @inbounds, @code_warntype, @simd.
    4. Read the following (very recommended) posts:
        1. Tips to get high-performing code (the 1st one is extremely good):
            1. http://www.stochasticlifestyle.com/7-julia-gotchas-handle/
            2. https://julialang.org/blog/2013/09/fast-numeric/
        2. Type stability: https://www.johnmyleswhite.com/notebook/2013/12/06/writing-type-stable-code-in-julia/
        3. Vectorized vs Devectorized code, Julia vs R vs C: https://www.johnmyleswhite.com/notebook/2013/12/22/the-relationship-between-vectorized-and-devectorized-code/
        4. Videos from MIT course: https://www.youtube.com/channel/UCDtsHjkOEMHYPGgpKX8VOPg/videos

# ABOUT GETTING HIGH PERFORMANCE



Once you are done with all the previous recommendations. The next step is to enter the world of parallel computing. Julia supports parallelism at different levels (see in the official documentation):

1. Distributed memory environments (MIP).

2. Shared memory environments (multi-threading and SIMD instructions).
    1. This is yet to be improved, i.e., it is not as good as OpenMP or TBB. E.g, there is not reduce operation so we can have problems of False Sharing.
    2. They are developing this part in collaboration with the people of Intel (specifically, I think that with the people of TBB).

3. GPU programming. CUDA Support, not quite for other architectures.

# CONCLUSIONS

RYANAIR LABS
Data&Analytics

1. Julia is a language for scientific computing that can give you performances close to C if you code with care.

2. Learning the language is not difficult. What may be difficult at the beginning is to get the very best performance of the language if you have not previously worked in compiled languages (e.g., C++, Java…)

3. If you are looking for performance when working in Julia, 4 basic rules that can give you a lot:
    1. Code using functions at all time.
    2. Specify types.
    3. Keep type stability.
    4. Be careful about how you manage memory.

4. If you just use Pkgs for your work (i.e., you do not need to develop code at all), you may prefer stay with R, Python or whatever language you use. Why? Because Probably those Pkgs that you use are already using C/C++ behind the scenes and you already have a very good performance.

# A FINAL THOUGH

RYANAIR LABS
Data&Analytics

> “Learning Julia nowadays is like having bought Bitcoins in 2012… An investment that is worth making.”
>
> *Jorge López, Data Scientist at Ryanair labs.*

# QUESTIONS?

David García Heredia
garciaherediad@ryanair.com

Madrid, 16/Sept/2021