Linear Collider Physics Analysis in Jupyter

Let's get started

- Log on to your site
  - For OSG: ssh -L 80xy:localhost:80xy login.snowmass21.io #replace the server name for your site
  - x=0 y=9 is mine.
  - Pick your own x and y.
- Download julia
  - wget https://julialang-s3.julialang.org/bin/linux/x64/1.5/julia-1.5.3-linux-x86_64.tar.gz
  - tar xzf julia-1.5.3-linux-x86_64.tar.gz
  - julia-1.5.3/bin/julia
- Start julia – this is the REPL (read-eval-print loop)
  - Powerful support for different modes: e.g., shell, package, julia, C++ modes
  -] add IJulia #] starts the package mode
  - build IJulia
  - Backspace to get back to julia mode
- Start the notebook
  - On your laptop
    - using IJulia
    - notebook()
  - On OSG/KEK/NAF
    - source /cvmfs/belle.cern.ch/tools/b2setup release-04-02-08
    - jupyter notebook --no-browser --port=80xy # use the same x and y from when you logged in.

Running the notebooks

- Run the notebooks from here: https://github.com/jstrube/LC_with_Julia_examples/
- You will see that it won't run!
- Not all necessary packages have been installed. The error message will tell you what to do. Click on the + symbol to add a new cell and copy and paste the code that the error message suggests.
  - This is how you add new packages.
- Add all packages that you see in the notebook (using xxx)
  - If you follow in the REPL instead of the notebook, replace "StatsPlots" with "UnicodePlots"
- Run again
  - You will see messages like "Precompiling...". This will take a while, but it's only necessary after installing or updating packages.
- In the meantime, let's move on with the slides.

Program for today

- https://github.com/jstrube/LC_with_Julia_examples/blob/main/FirstSteps.ipynb
  - Introduction for how to open files and read a few collections
  - If you have worked with LCIO before, the operations should be very familiar
- https://github.com/jstrube/LC_with_Julia_examples/blob/main/NTupleExample.ipynb
  - A simple example for how to read an LCIO file and write out a DataFrame for data analysis offline, including how to make cuts.
- https://github.com/jstrube/LC_with_Julia_examples/blob/main/DL_Calo.ipynb
  - An example for how to use the deep learning library Flux, using a simple calorimeter calibration
  - A Gaussian distribution can be fit to the calibrated distribution.

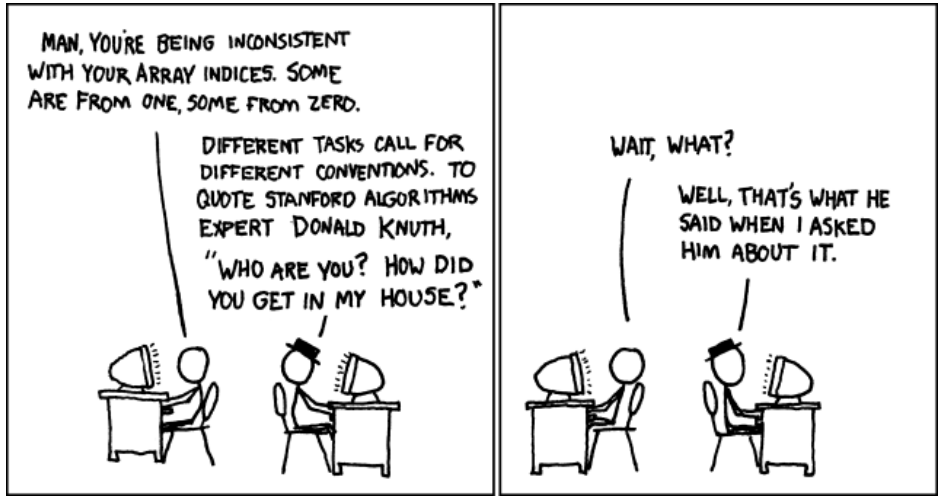
Julia – the "ju" in Jupyter

- Support for multithreaded, concurrent, and distributed processing
- Unicode support for variables
- Interactive programming
- Multi-dimensional arrays (like numpy, but built-in)
- Rich ecosystem for technical computing
  - Statistics: Distributions.jl, Turing.jl (probabilistic programming), …
  - Differential Equations: DifferentialEquations.jl, SciML.ai
  - Deep Learning: Flux.jl, Knet.jl
  - Plotting: Plots.jl (with different backends), PyPlot.jl (wrapper around matplotlib)
- Salespoint for me: Allows me to explore the data, and when I need a fast function for serious work (e.g. a new calorimeter clustering), I can write it in the same language I use for interactive exploration.

First steps in Julia

- Julia supports unicode: Enter \mu<TAB>
  - UTF-8 is fully supported, but not everything has a \-shortcut
- Full support for matrices
  - X = randn((20, 10)) # makes a 20x10 matrix
  - Y = X' # transposes the matrix
- Iterations and printing similar to python
  - for x in 0:10 println(x) end
  - Note: no ":", but "end" to delimit blocks
- Functions don't need type parameters (but you can use them)
  - F(x) = sin(x) is a function
  - function F(x::Int64) sin(1.5x) end is another function with the same name.
  - Return is optional. The value of the last statement in the function is returned. function F(x::Float64) return sin(0.5x) end is also fine.

Some noteworthy differences to languages you may be familiar with



- 1-based indexing by default
  - Or, random, if you want
- Structs, yes, but no member functions
  - Multiple dispatch instead
  - Use the object as the first parameter of the function instead.
    Example: C++: vec.size() Julia: length(vec)
- No semicolon required, no indentation or {} to delimit blocks
  - if … end; for … end, function … end

Further information about Julia

- Starting point: https://julialang.org
  - Documentation: https://docs.julialang.org/en/v1/
    - Note that things that run in v1.0 are guaranteed to run in any v1.x, but do choose the latest version to get more features.
  - Other learning resources: https://julialang.org/learning/
- The recent community conference online has a good mix of introductory and overview material https://www.youtube.com/playlist?list=PLP8iPy9hna6Tl2UHTrm4jnIYrLkIcAROR