Introduction to Julia Programming

Jake W. Liu

Outline

- Overview
- Syntax
- Packages
- Advance Topics
- Exercises

Overview

Overview

- Julia is a high-level and performative language.
- Julia is designed for technical computing, though its usage is general.
- Julia is open-source.
- Can call methods defined in C and Python easily.

Overview - Installation

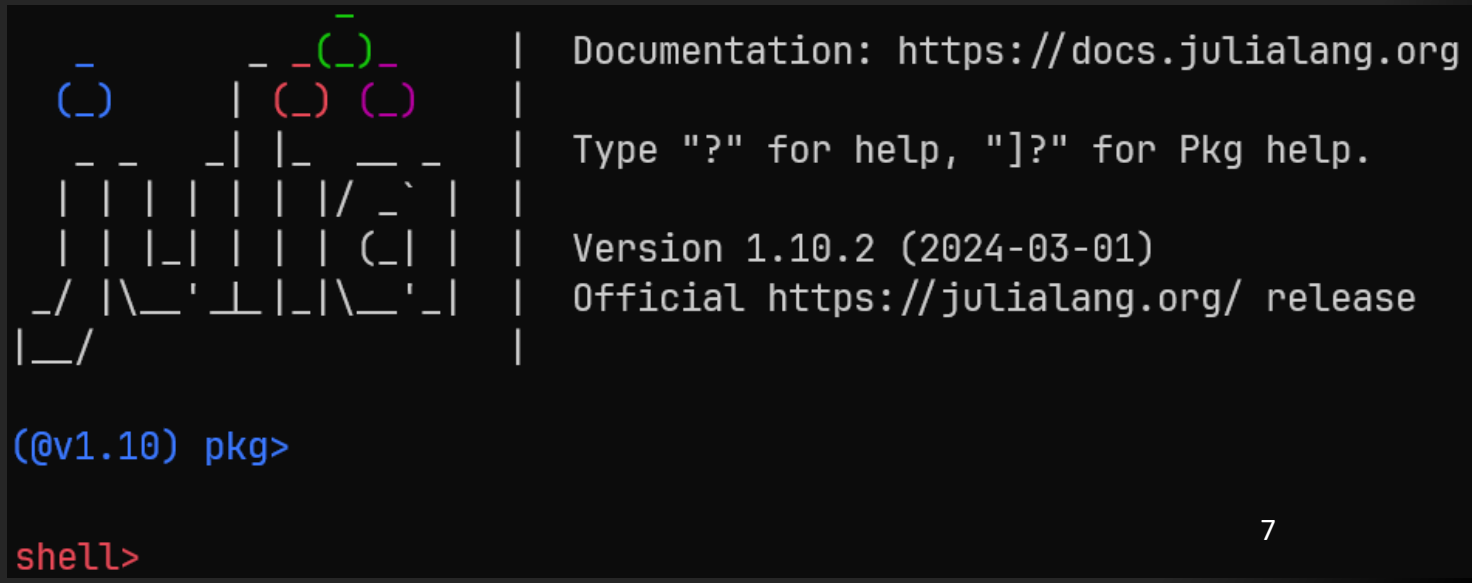
- https://julialang.org/downloads/

Overview - Run a Julia Program

- In Terminal
  - julia xxx.jl
- In REPL
  - include(“xxx.jl”)

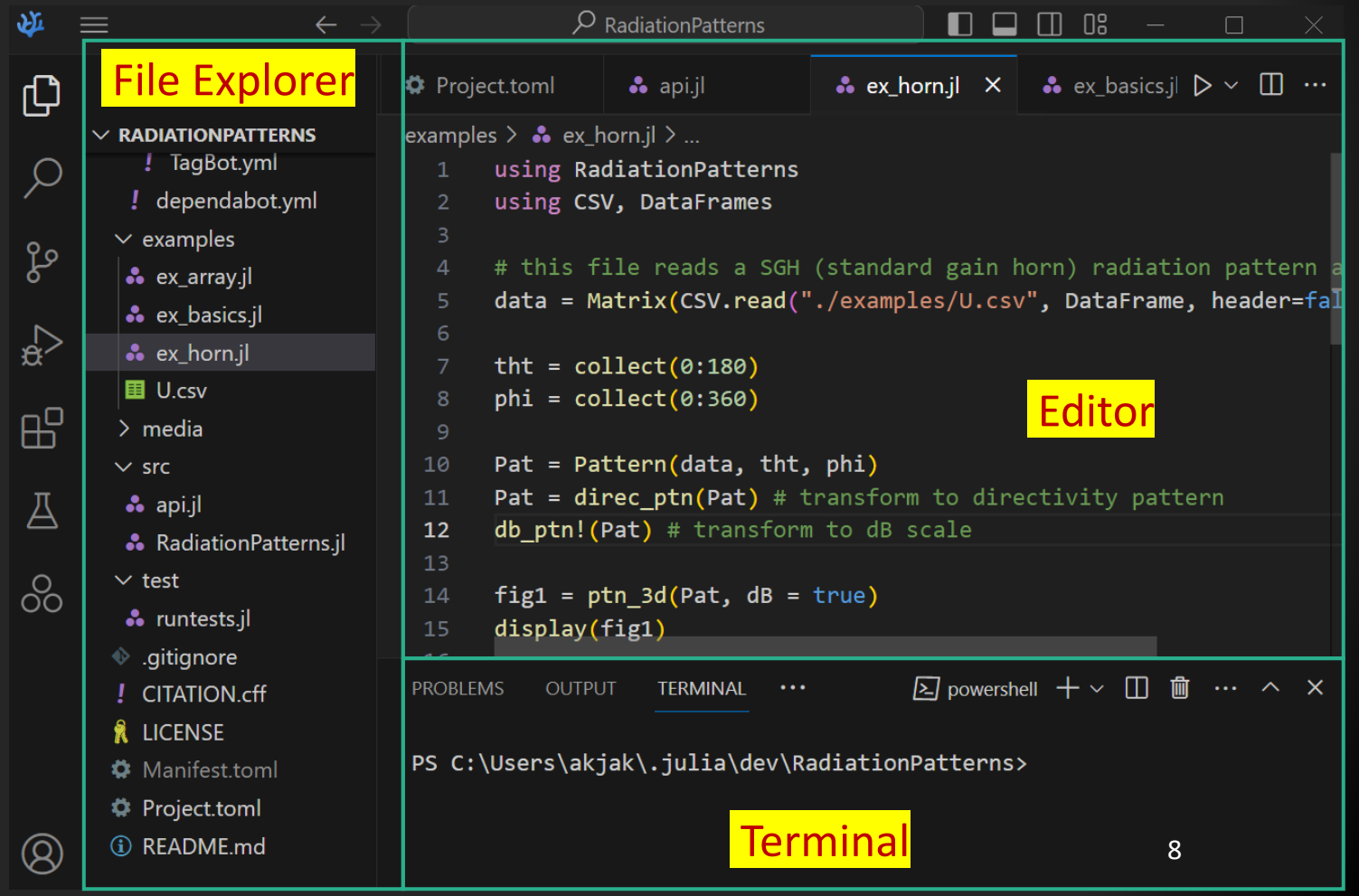
Overview - REPL

- Enter “**]**” in REPL => package mode (equivalent to using Pkg)
  - add PkgName
- Enter “**;**” in REPL => shell mode
- Enter “**back**” to exit REPL



Overview – Text Editor

- IJulia + Jupyter Notebook
- Juno
- Pluto
- VS Code (VS Codium)
- …

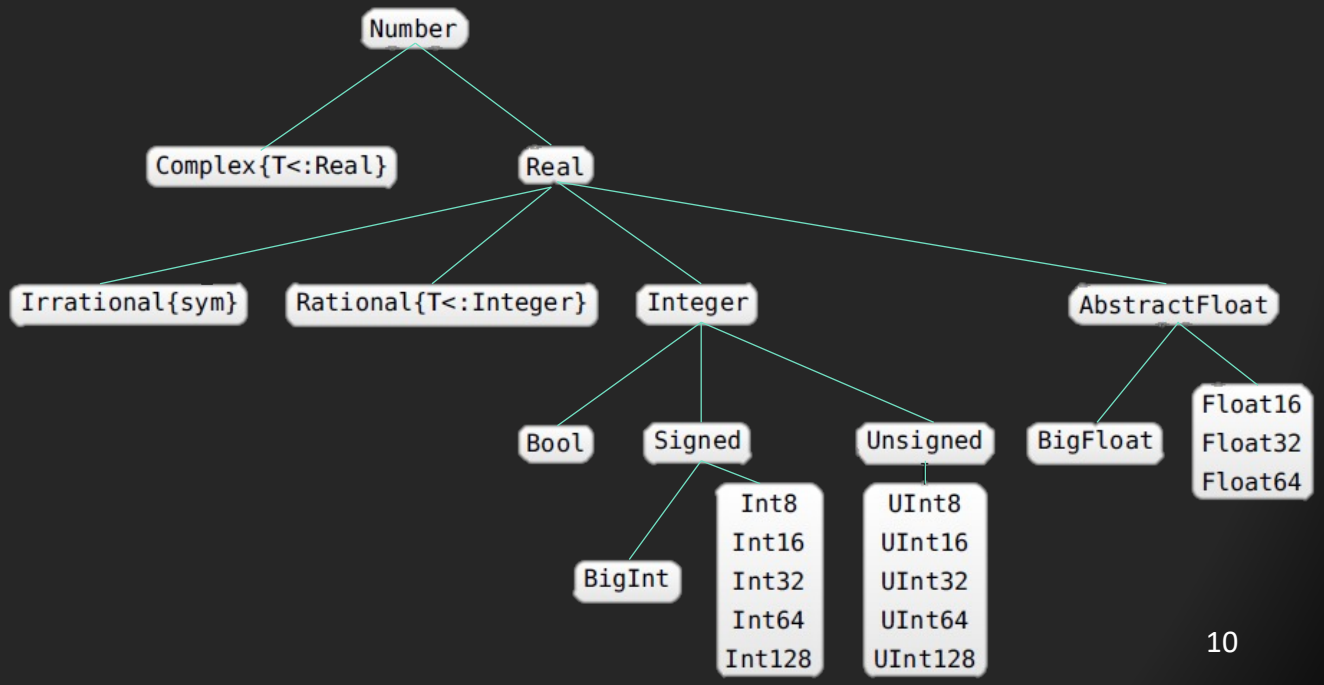


Syntax

Syntax – Basic Data Types

- Integers: Int64, Int32
- Real Numbers: Float64, Float32
- Boolean: Bool
- Strings: String
- Self-defined: struct

There are also abstract data types in Julia. Types are fundamental to Julia and enable an important feature called multiple dispatch



Syntax – Basics

- Flow controls
  - if / elseif / else
  - for loop
  - while loop
- Most syntaxes are similar with MATLAB!
  - ✓ similar mathematical functions
  - ✓array indexing also starts from 1

Syntax – Notable Differences with MATLAB

- Arrays are indexed with square brackets, A[i, j].
- Arrays are assigned by reference.
  - after A = B, changing elements of B will modify A as well!
- Does not automatically grow arrays in an assignment statement.
  - use push!() or append!()
- Literal numbers without a decimal point create integers instead of floating-point numbers.

There are comparisons with Python out there as well.

see: https://web.mit.edu/julia_v0.6.2/julia/share/doc/julia/html/en/manual/noteworthy-differences.html
cheat sheet: https://cheatsheets.quantecon.org/

Syntax – Macros

- Macros are special “functions” that transform and generate code at compile time, allowing for powerful metaprogramming capabilities.
- Starts with @

Code Snippet

```
macro HelloWorld()
    return :( println("Hello World!"))
end

julia> @HelloWorld
Hello World!
```

Syntax – Variable Scope

- The scope of a variable is the region of code within which a variable is accessible.

Construct	Scope type	Allowed within
module	global	global
struct	local (soft)	global
for, while, try	local (soft)	global, local
macro	local (hard)	global
functions, do blocks, let blocks, comprehensions, generators	local (hard)	global, local

begin blocks and if blocks do *not* introduce new scopes

https://docs.julialang.org/en/v1/manual/variables-and-scoping/

Syntax – Variable Scope Example

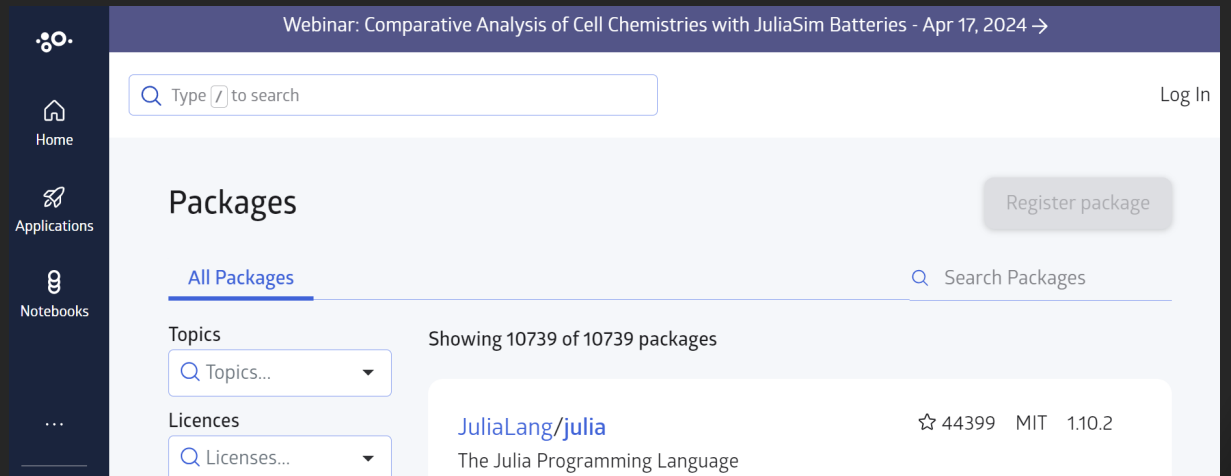
Code Snippet 1	Code Snippet 2

s = 0 # global
for i in 1:n
  s = s + i # ambiguous!!
end
 	
# Solve: add the keyword global before s in the for loop 	
	function sum_to(n)
	  s = 0 # new local
	  for i in 1:n
	  s = s + i # assign existing local
	  end
	  return s # same local => OK!
	end

Packages

Package Ecosystems

- Packages: Collections of reusable Julia code that extend the language's functionality.
- Open source => you can choose your preferred packages! https://juliahub.com/ui/Packages



Package Basics

- Package Manager (Pkg): Julia's built-in tool for managing packages, including installation, updating, and dependency resolution.
- Continuous Integration (CI): Many packages use CI services to ensure code quality by running automated tests on different platforms and Julia versions. => Avoids dependency hell!
- using PkgName

Package Ecosystems

- **LinearAlgebra.jl**
- **FFTW.jl**
- **Plots.jl**
- **Infiltrator.jl**
- **Flux.jl**
- …

You should explore the ecosystem based on your specific needs.

Linear Algebra: **LinearAlgebra.jl**

- http://web.mit.edu/julia_v0.6.2/julia/share/doc/julia/html/en/stdlib/linalg.html
  - Basic matrix-vector operations: *, \...
  - Basic LinAlg functions: inv( ), dot( ), svd( )...

Debugging: **Infiltrator.jl**

- https://github.com/JuliaDebug/Infiltrator.jl
  - Add @infiltrate in between the codes to act as breakpoints
  - @locals : Print local variables. @locals x y only prints x and y.
  - @continue : Continue to the next infiltration point or exit (shortcut: Ctrl-D).
  - @exit : Stop infiltrating for the remainder of this session and exit.

Debug tool in VS code is not so efficient in Julia currently. It is recommended to use this package for debugging.

Plottings: **Plots.jl**

- https://docs.juliaplots.org/stable/
  - Concise and flexible (always your first data visualization package).
  - Provides a unified API to various plotting backends.

I personally prefer to use **PlotlyJS.jl**. For a more *julian* implementation, one can check out **Makie.jl**

Advance Topics

Some Concepts in Parallel Computing

- Asynchronous: interactions with the outside world
- Multithreaded: parallel on multiple CPU cores / single process
- Distributed: parallel on multiple CPU cores / multiple processes

Warning: Writing memory-efficient code is more important than relying on parallel programming.

Asynchronous Programming

- @task and @async macro
- @async is equivalent to schedule(@task x)

Code Snippet

```
julia> t = @task begin; sleep(5); println("done"); end
Task (runnable) @0x00007f13a40c0eb0

julia> schedule(t); wait(t)
done
```

Multi-threaded Programming

- Set thread number at start: julia –t 4 => use 4 threads
- Use the @threads macro
- Be aware of data race issues!

Code Snippet 1	Code Snippet 2
@threads for i = 1:10 a[i] = Threads.threadid() end	function sum_multi_bad(a) s = 0 **@threads** for i in a s += i #data race occurs! end s end

Distributed Programming

- Set process number at start: julia –p 4 => use 4 processes
- Use the @spawnat macro and fetch()
- Data transfer: **MPI.jl** and **SharedArrays.jl**

Code Snippet

```
julia> r = @spawnat :any rand(2,2)
Future(2, 1, 4, nothing)

julia> fetch(r)
2×2 Matrix{Float64}:
 0.374379  0.468878
 0.564313  0.888577
```

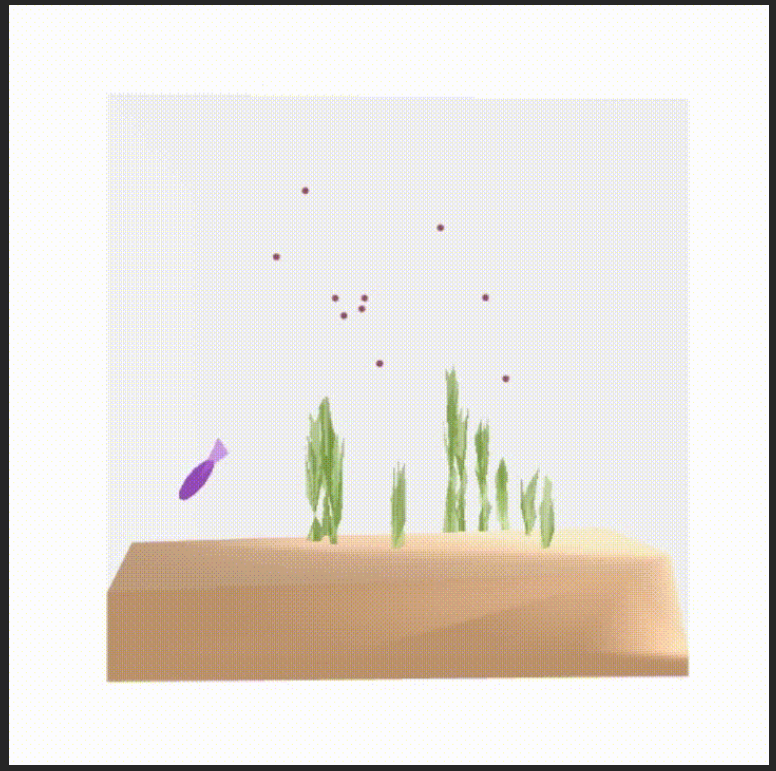
Excercises

Start Coding! Some Advices

- Translate current project into Julia (from MATLAB, Python, etc.)
- Create some entertaining small projects
- Ask questions on Julia Discourse

Electronic Pets: **FishTank.jl**

- FishTank.jl a fish tank app created with PlotlyJS
  - Modeling fish motion with simple linear algebra!



Thank You!