12.010 Computational Methods of Scientific Programming

Lecture 19: Julia

Julia programming language

- Python, Matlab high-level, interpreted, can be fast but can be very slow for e.g. loops.
- R - great at statistics, but sloooow.
- C, C++, Fortran fast, compiled, but not very interactive, old style very pedantic syntax.
- Julia tries to fuse good features and leave out undesirable qualities

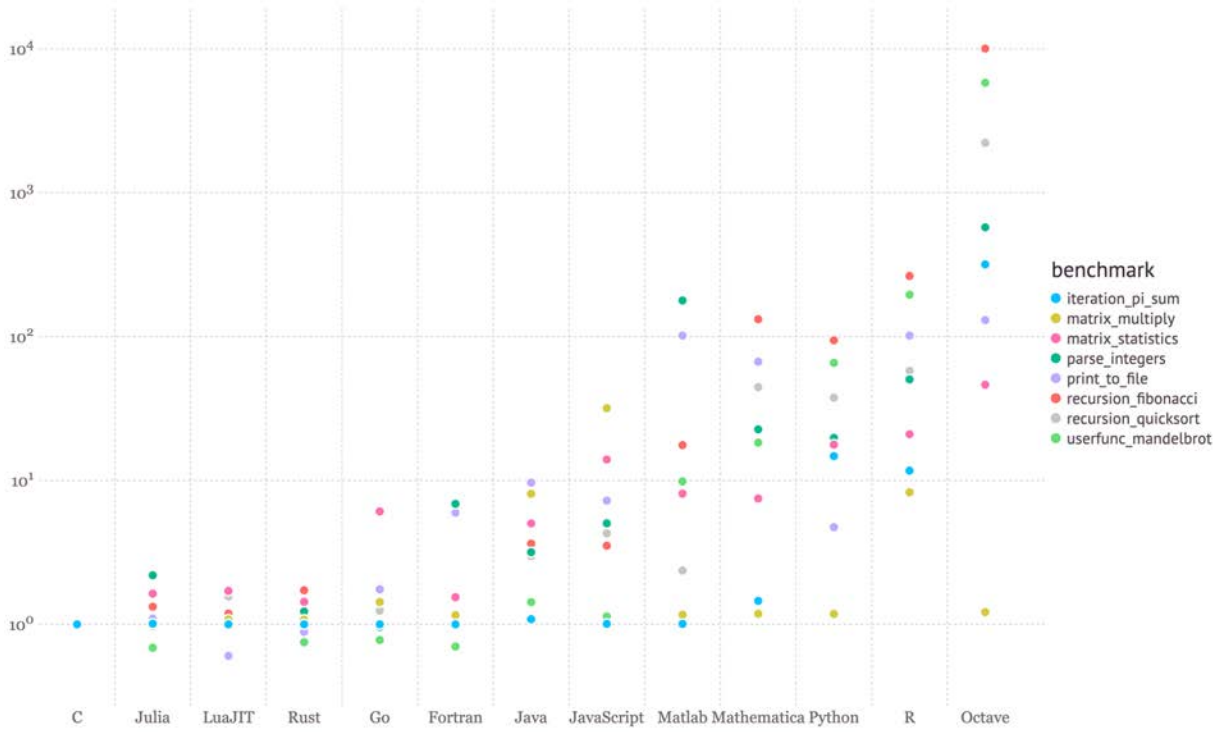
Julia relative performance

Plot shows relative performance of some computing micro-benchmarks in different languages.

Julia is an interpreted/JIT language but is competitive with compiled languages. (JIT – just in time compiled).

JIT means that we can work interactively, but code (in functions) will be compiled on the fly

Compiled code is cached so one time compile overhead can be amortized.



Julia uptake

Still much less widespread than Python, R, Matlab

But growing

Mostly interoperable with Python packages (via “PyCall” and/or PyPlot – Matplotlib optimized interface).

	Total Cumulative as of Jan 1, 2020	Total Cumulative as of Jan 1, 2021	Change
Number of Julia Downloads (JuliaLang + Docker + JuliaPro)	12,950,630	24,205,141	+87%
Number of Julia Packages	2,787	4,809	+73%
GitHub stars (Julia language repo + registered packages)	99,830	161,774	+62%
YouTube views (Julia language channel)	1,562,223	3,320,915	+113%
Published citations of Julia: A Fast Dynamic Language for Technical Computing (2012) + Julia: A Fresh Approach to Numerical Computing (2017)	1,680	2,531	+51%
Discourse posts	137,399	211,888	+54%
TIOBE Index Rank	#47	#23	+24

Julia history

- Named origin ~2009 MIT, Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral Shah.
- Some heritage of numerous “parallel matlab” projects dating bak to at least 1995.
- V1.0 release 2018.
- Some interesting things
  - Arrays are arrays (no numpy, lists etc..)
  - Types support different style of work
    - Interactive, but loops can be fast
    - Functions have some nice syntax
  - Strong support for mathematical ideas

Julia getting set up

- Try install in Python notebook
  Julia/account_configadd_julia_kernel.ipynb

```
# Setup Julia kernel in my account
# ================================
#
# Note - if you execute this cell more than once it will leave things unchanged after the first
# execution.
#
!julia -e 'using Pkg; Pkg.add("IJulia");Pkg.update();'
```

  - this should add

- or on laptop (download) and try “REPL”

Julia notebook

```
[1]: versioninfo(verbose=true)
```

```
Julia Version 1.6.2
Commit 1b93d53fc4 (2021-07-14 15:36 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
      Ubuntu 20.04.3 LTS
  uname: Linux 5.4.0-1045-aws #47-Ubuntu SMP Tue Apr 13 07:02:25 UTC 2021 x86_64 x86_64
  CPU: Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz:
              speed         user         nice          sys         idle          irq
       #1  2300 MHz     165257 s       4511 s      25622 s   65296668 s          0 s
       #2  2300 MHz     170724 s       4731 s      25684 s   65294682 s          0 s

  Memory: 7.773159027099609 GB (621.27734375 MB free)
  Uptime: 6.550467e6 sec
  Load Avg:  0.33  0.15  0.06
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-11.0.1 (ORCJIT, broadwell)
Environment:
  HOME = /home/jpy_class/mit/12.010/cnh@mit.edu
  PATH = /opt/tljh/user/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
```

First command in a notebook takes a few seconds. Julia compiles everything to cache on the fly, and a lot of Julia itself is in Julia e.g.

Julia notebook

[4]:
```
@which versioninfo()
```

[4]: versioninfo() in InteractiveUtils at /opt/julia-1.6.2/share/julia/stdlib/v1.6/InteractiveUtils/src/InteractiveUtils.jl:85

[9]:
```
;tail --lines=+85 '/opt/julia-1.6.2/share/julia/stdlib/v1.6/InteractiveUtils/src/InteractiveUtils.jl'
```

```
function versioninfo(io::IO=stdout; verbose::Bool=false)
    println(io, "Julia Version $VERSION")
    if !isempty(Base.GIT_VERSION_INFO.commit_short)
        println(io, "Commit $(Base.GIT_VERSION_INFO.commit_short) ($(Base.GIT_VERSION_INFO.date_string))")
    end
    if ccall(:jl_is_debugbuild, Cint, ())!=0
        println(io, "DEBUG build")
    end
    println(io, "Platform Info:")
    println(io, "  OS: ", Sys.iswindows() ? "Windows" : Sys.isapple() ?
        "macOS" : Sys.KERNEL, " (", Sys.MACHINE, ")")

    if verbose
        lsb = ""
        if Sys.islinux()
            try lsb = readchomp(pipeline(`lsb_release -ds`, stderr=devnull)); catch; end
        end
        if Sys.iswindows()
            try lsb = strip(read(`$(ENV["COMSPEC"]) /c ver`, String)); catch; end
        end
        if !isempty(lsb)
```

Julia REPL

Running "REPL" from a shell.

```
(base) chriss-MacBook-Pro:C chrishill$ /Applications/Julia-1.6.app/Contents/Resources/julia/bin/julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.6.2 (2021-07-14)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> versioninfo(verbose=true)
Julia Version 1.6.2
Commit 1b93d53fc4 (2021-07-14 15:36 UTC)
Platform Info:
  OS: macOS (x86_64-apple-darwin18.7.0)
  uname: Darwin 21.1.0 Darwin Kernel Version 21.1.0: Wed Oct 13 17:33:23 PDT 2021; root:xnu-8019.41.5~1/RELEAS
E_X86_64 x86_64 i386
  CPU: Intel(R) Core(TM) i9-9980HK CPU @ 2.40GHz:
                 speed         user         nice          sys         idle          irq
       #1-16  2400 MHz    2094765 s          0 s    1044112 s   62017672 s          0 s

  Memory: 64.0 GB (23.78125 MB free)
  Uptime: 435551.0 sec
  Load Avg:  1.978515625  1.97802734375  2.10546875
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-11.0.1 (ORCJIT, skylake)
Environment:
  TERM = xterm-256color
  PATH = /Users/chrishill/opt/anaconda3/bin:/Users/chrishill/opt/anaconda3/condabin:/usr/local/bin:/usr/bin:/b
in:/usr/sbin:/sbin:/Library/TeX/texbin:/usr/local/share/dotnet:/opt/X11/bin:~/.dotnet/tools:/Library/Framework
s/Mono.framework/Versions/Current/Commands
  XPC_FLAGS = 0x0
  HOME = /Users/chrishill

julia>
```

Julia REPL

Running
“REPL” from
a shell.

```
julia> @which versioninfo()
versioninfo() in InteractiveUtils at /Applications/Julia-1.6.app/Conten
1.6/InteractiveUtils/src/InteractiveUtils.jl:85

shell> less /Applications/Julia-1.6.app/Contents/Resources/julia/share/
InteractiveUtils.jl
```

```
Print information about the version of Julia in use. The output is
controlled with boolean keyword arguments:

- `verbose`: print all additional information
"""
function versioninfo(io::IO=stdout; verbose::Bool=false)
    println(io, "Julia Version $VERSION")
    if !isempty(Base.GIT_VERSION_INFO.commit_short)
        println(io, "Commit $(Base.GIT_VERSION_INFO.commit_short) ($(Base.GIT_VERSION_INFO.date_string))")
    end
    if ccall(:jl_is_debugbuild, Cint, ())!=0
        println(io, "DEBUG build")
    end
    println(io, "Platform Info:")
    println(io, "  OS: ", Sys.iswindows() ? "Windows" : Sys.isapple() ?
        "macOS" : Sys.KERNEL, " (", Sys.MACHINE, ")")

    if verbose
        lsb = ""
        if Sys.islinux()
            try lsb = readchomp(pipeline(`lsb_release -ds`, stderr=devnull)); catch; end
        end
        if Sys.iswindows()
            try lsb = strip(read(`$(ENV["COMSPEC"]) /c ver`, String)); catch; end
        end
        if !isempty(lsb)
            println(io, "      ", lsb)
        end
        if Sys.isunix()
            println(io, "  uname: ", readchomp(`uname -mprsv`))
        end
    end

    if verbose
        cpuio = IOBuffer() # print cpu_summary with correct alignment
        Sys.cpu_summary(cpuio)
        for (i, line) in enumerate(split(String(take!(cpuio)), "\n"))
            prefix = i == 1 ? "  CPU: " : "       "
            println(io, prefix, line)
        end
    else
        cpu = Sys.cpu_info()
```

Julia basic math is much like python, C, Fortran etc…

```
[10]: x=2
      y=1e23

[10]: 1.0e23

[13]: x+y-y

[13]: 0.0
```

Everything is typed, but types are inferred when they can be.

```
[14]: π

[14]: π = 3.1415926535897...

[15]: typeof(π)

[15]: Irrational{:π}

[16]: typeof(x)

[16]: Int64

[17]: typeof(y)

[17]: Float64
```

Julia types are very organized and central to language

```
[34]: supertypes(typeof(x))
```

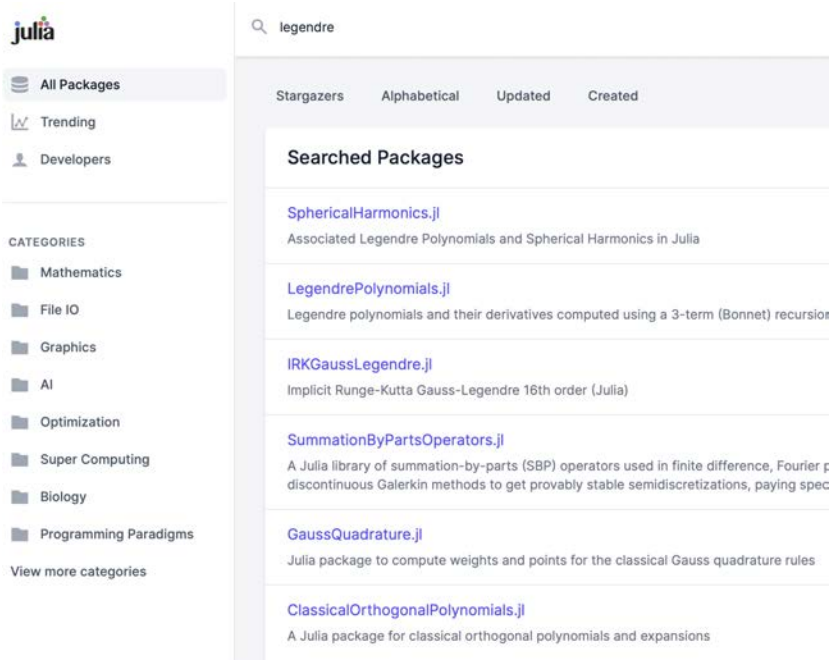
```
[34]: (Int64, Signed, Integer, Real, Number, Any)
```

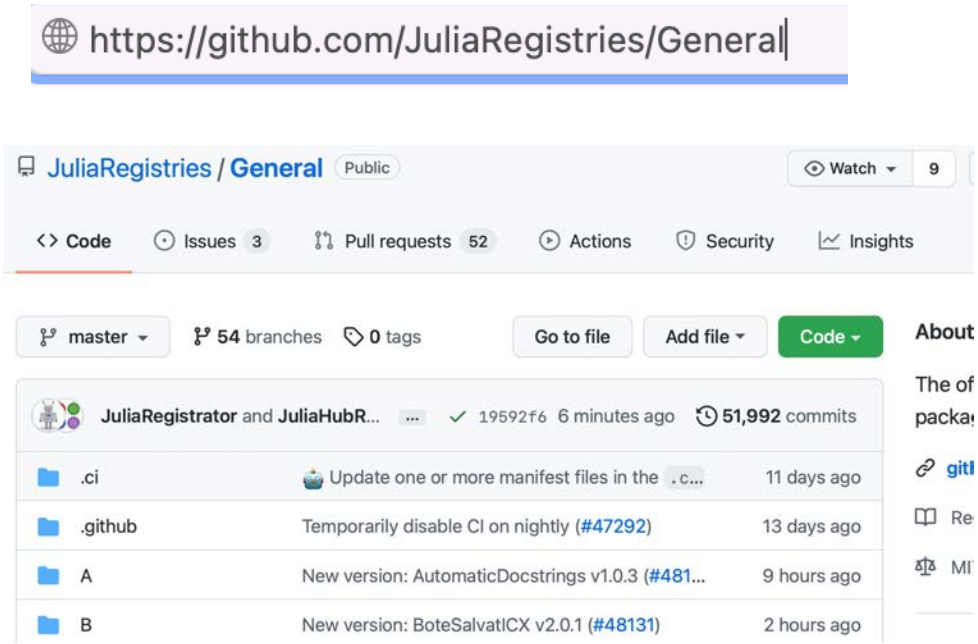
```
[35]: supertypes(typeof(π))
```

```
[35]: (Irrational{:π}, AbstractIrrational, Real, Number, Any)
```

Julia has a builtin package ecosystem that has > 4000 packages/modules

https://juliapackages.com





Julia package syntax is multi-stepped

Pkg.add is a bit like conda install in Python. Once a package is added no need to install again. Need to Pkg.add() before can import with “using”.

```
[4]: using Pkg
     Pkg.add("Plots")
```

```
  Resolving package versions...
 No Changes to `~/.julia/environments/v1.6/Project.toml`
 No Changes to `~/.julia/environments/v1.6/Manifest.toml`
```

using NAME is similar to “import” in Python

```
[5]: using Plots
```

```
[6]: x = 1:10; y = rand(10);
     plot(x,y)
```

[6]:

Julia basic syntax

- Some online resources
  - https://juliadocs.github.io/Julia-Cheat-Sheet/
  - https://juliabyexample.helpmanual.io

- Basics notebooks
  - account_configadd_julia_kernel.ipynb
  - Lec20_julia_basics.jl
  - julia-basics.ipynb

MIT OpenCourseWare

https://ocw.mit.edu

12.010 Computational Methods of Scientific Programming, Fall 2024