1 Julia: first contact

- Introductory material partially inspired by Hua Zhou and the Julia Cheat Sheet

1.0.1 Resources

- Homepage
- Documentation
- Cheat Sheet
- WikiBook

1.0.2 Hint for starting

Use the Cheat Sheet to see a compact and rather comprehensive list of basic things Julia. This notebook tries to discuss some concepts behind Julia.

1.1 Open Source

- Julia is an Open Source project started at MIT
- Julia itself is distributed under an MIT license, several packages have different licenses
- Development takes place on github
- As of today there 882 contributors to the code
- The Open Source paradigm corresponds well to the fundamental requirement that scientific research should be transparent and reproducible

1.2 How to install and run Julia

- Install and run as system executable
  - Download from julialang.org (recommended by Julia creators)
  - Installation via system package manager (yast, apt-get, homebrew)
  - Access in command line mode - edit source code in any editor
  - Access via JUNO plugin of Atom code editor
- Access via Jupyter server
  - https://www-pool.math.tu-berlin.de/jupyter/ (TU Berlin Unix pool account required)
  - Your local Jupyter installation
1.3 REPL: Read-Evaluation-Print-Loop

Start REPL by calling `julia` in terminal

1.3.1 REPL modes:

- **Default mode**: `julia>` prompt. Type backspace in other modes to enter default mode.
- **Help mode**: `help?>` prompt. Type `?` to enter help mode. Search via `?search_term`
- **Shell mode**: `shell>` prompt. Type `;` to enter shell mode.
- **Package mode**: `Pkg>` prompt. Type `]` to enter package mode.

1.3.2 Helpful commands in REPL

- `quit()` or `Ctrl+D`: exit Julia.
- `Ctrl+C`: interrupt execution.
- `Ctrl+L`: clear screen.
- Append `;` to suppress displaying output from a command
- `include("filename.jl")`: source a Julia code file.

1.4 Jupyter

- Browser based interface to Julia, python, R
- Jupyter notebooks: JSON (Javascript serialization format) files which contain script snippets and results
- The user interface consists of code cells and text cells between, both can be edited. Code cells can be executed
- Any intermediate state of the notebook can be saved

1.5 Package management

- Julia has an evolving package ecosystem developed by a large community
- Packages provide functionality which is not part of the core Julia installation
- Each package is a git repository
  - Mostly on github as `Package.jl`, e.g. `AbstractTrees`
  - Package registry: `git repository` with metadata of registered packages
- Packages can be added to and removed from Julia installation
- Registered packages are added by name
- Any packages can be installed from URL
- Additional package dependencies are resolved automatically

Use package manager: itself a package installed by default

[1]: `using Pkg`

Add package `AbstractTrees`:

[2]: `Pkg.add("AbstractTrees")`

```
Updating registry at `~/.julia/registries/General`
Updating git-repo
`https://github.com/JuliaRegistries/General.git`
```
Resolving package versions...

Updating `~/.julia/environments/v1.2/Project.toml`
[no changes]

Updating `~/.julia/environments/v1.2/Manifest.toml`
[no changes]

List installed packages

[3]: Pkg.status()

Status `~/.julia/environments/v1.2/Project.toml`
[1520ce14] AbstractTrees v0.2.1
[7e460c6] ArgParse v0.6.2
[6e4b80f9] BenchmarkTools v0.4.3
[a9c87d775] CPUTime v1.0.0
[288d3ca9] GR v0.42.0
[7073ff75] IJulia v1.20.0
[98b081ad] Literate v2.1.0
[91a5bccc] Plots v0.27.0
[1fd47b50] QuadGK v2.1.1
[295af30f] Revise v2.2.2
[56f361f5] Triangle v0.2.0
[82b139dc] VoronoIFVM v0.5.0

[~/Wias/work/julia/dev/VoronoiFVM]
[44d3d7a6] Weave v0.9.1

Test package AbstractTrees:

[4]: Pkg.test("AbstractTrees")

Testing AbstractTrees
Status `/tmp/jl_En7JNn/Manifest.toml`
[1520ce14] AbstractTrees v0.2.1
[2a0f44e3] Base64 ["@stdlib/Base64""]
[8ba89e20] Distributed ["@stdlib/Distributed"]
[b77e0a4c] InteractiveUtils ["@stdlib/InteractiveUtils"]
[56dd0b16] Logging ["@stdlib/Logging"]
[d6f4376e] Markdown ["@stdlib/Markdown"]
[9a3f2824] Random ["@stdlib/Random"]
[9e88b42a] Serialization ["@stdlib/Serialization"]
[6462fe0b] Sockets ["@stdlib/Sockets"]
[8dfed614] Test ["@stdlib/Test"]

/usr/bin/julia -Cnative -J/usr/lib64/julia/sys.so -g1 --code-coverage=none --color=yes --compiled-modules=yes --check-bounds=yes --inline=yes --startup-file=yes --track-allocation=none --eval append!(empty!(Base.DEPOT_PATH), ["/home/fuhrmann/.julia", "/usr/local/share/julia", "/usr/share/julia"]) append!(empty!(Base.DL_LOAD_PATH), String[])

cd("/home/fuhrmann/.julia/packages/AbstractTrees/z1wBY/test")
include("/home/fuhrmann/.julia/packages/AbstractTrees/z1wBY/test/runtests.jl")

Array{Any,1}
  1
    Array{Any,1}
    2
      3
    2
      3
        4
    2
      3
        4

Testing AbstractTrees tests passed

Remove package AbstractTrees:

[5]: Pkg.rm("AbstractTrees")

  Updating `~/.julia/environments/v1.2/Project.toml`
  [1520ce14] - AbstractTrees v0.2.1
  Updating `~/.julia/environments/v1.2/Manifest.toml`
  [1520ce14] - AbstractTrees v0.2.1

See Cheat Sheet for more

1.5.1 Local copies of packages

- Upon installation, local copies of the package source code is downloaded from git repository
- By default located in .julia/packages subdirectory of your home folder
- You can remove .julia on inconsistencies and re-install packages (aka “reinstall windows”…)

1.6 Standard number types

- Julia is a strongly typed language, so any variable has a type.
- Standard number types allow fast execution because they are supported in the instruction set of the processors

Integers

[6]: i=1
typeof(i)

[6]: Int64

Floating point numbers
```plaintext
[7]:  y = 1.0
    typeof(y)
[7]:  Float64

Rational numbers

[8]:  r = 3 // 7
[8]:  3 // 7

Unicode variable names: type \\pi<tab>

[9]: @show pi
    println(1.0*pi)
    typeof(pi)

pi =
3.141592653589793
[9]:  Irrational{:}

1.7 Vectors

- Elements of a given type stored contiguously in memory
- Vectors and 1-dimensional arrays are the same
- Vectors can be created for any element type

Vector creation by explicit list of elements

[10]:  v1 = [1, 2, 3, 4]
[10]:  4-element Array{Int64,1}:
    1
    2
    3
    4

Type of a vector element

[11]:  @show eltype(v1);

eltype(v1) = Int64

If any element in the initializer is float, the vector becomes float

[12]:  v2 = [1.0, 2, 3, 4]
[12]:  4-element Array{Float64,1}:
    1.0
```
2.0
3.0
4.0

[13]: `@show v2[2]`

\[
\begin{align*}
v2[2] &= 2.0
\end{align*}
\]

[13]: 2.0

Create integer vector of zeros

[14]: `v3=zeros(Int,4)`

[14]: 4-element Array{Int64,1}:
  0
  0
  0
  0

Create float vector of zeros

[15]: `v3=zeros(Float64,4)`

[15]: 4-element Array{Float64,1}:
  0.0
  0.0
  0.0
  0.0

Fill vector with constant data

[16]: `fill!(v3,10)`

[16]: 4-element Array{Float64,1}:
  10.0
  10.0
  10.0
  10.0

See Cheat Sheet for more

1.7.1 Ranges

Ranges describe sequences of numbers and can be used in loops, array constructors etc.

[17]: `r1=1:10`

```
@show r1
@show typeof(r1)
```
r1 = 1:10
typeof(r1) = UnitRange{Int64}

[17]: UnitRange{Int64}

[18]: r2 = 1:2:10
@show r2
@show typeof(r2)

r2 = 1:2:9
typeof(r2) = StepRange{Int64,Int64}

[18]: StepRange{Int64,Int64}

Create vector from range

[19]: collect(r1)

[19]: 10-element Array{Int64,1}:
   1
   2
   3
   4
   5
   6
   7
   8
   9
 10

[20]: collect(1:2:5)

[20]: 3-element Array{Int64,1}:
   1
   3
  5

[21]: collect(1:0.1:2)

[21]: 11-element Array{Float64,1}:
   1.0
   1.1
   1.2
   1.3
   1.4
   1.5
   1.6
   1.7
Create vector from list comprehension containing range

```
[22]: v=[sin(i) for i=1:5]
```

```
5-element Array{Float64,1}:
 0.8414709848078965
 0.9092974268256817
 0.1411200080598672
-0.7568024953079282
-0.9589242746631385
```

### 1.7.2 Vector dimensions

```
[23]: v=collect(1:2:100);
```

Size of a vector is a tuple with one element

```
[24]: @show size(v);
```

```
size(v) = (50,)
```

Length is the overall number of elements in a vector

```
[25]: @show length(v);
```

```
length(v) = 50
```

### 1.7.3 Subarrays

Subarrays are copies of parts of arrays

```
[26]: v=collect(1:2:10)
    @show v;
```

```
v = [1, 3, 5, 7, 9]
```

Subvector for indices 2 to 4 contains a copy of data of v

```
[27]: vsub=v[2:4]
    @show vsub;
```

```
vsub = [3, 5, 7]
```

Changing elements in vsub does not affect v
1.7.4 Array views

Array views allow to access of a part of an array.

```plaintext
[28]: vsub[1]=100
   @show vsub
   @show v;

vsub = [100, 5, 7]
v = [1, 3, 5, 7, 9]

[29]: v=collect(1:2:10)
   @show v;

v = [1, 3, 5, 7, 9]
Subvector for indices 2 to 4 contains a copy of data of v

[30]: vview=view(v,2:4)
   @show vview;

vview = [3, 5, 7]
Changing elements in vview also changes v

[31]: vview[1]=100
   @show vview
   @show v;

vview = [100, 5, 7]
v = [1, 100, 5, 7, 9]

@views macro

[32]: v=collect(1:2:10)
   @show v;

v = [1, 3, 5, 7, 9]
Subvector for indices 2 to 4 contains a copy of data of v

[33]: @views vview=v[2:4]
   @show vview;

vview = [3, 5, 7]
Changing elements in vview also changes v

[34]: vview[1]=1000
   @show vview;
   @show v;
```
vview = [1000, 5, 7]
v = [1, 1000, 5, 7, 9]

1.7.5 Dot operations

Element-wise operations on arrays

```julia
v = collect(0:0.1:1)
@show sin.(v)
@show 2 .* v;
```

sin.(v) = [0.0, 0.09983341664682815, 0.19866933079506122, 0.29552020666133955, 0.3894183423086505, 0.479425538604203, 0.5646424733950354, 0.644217687237691, 0.7173560908995228, 0.7833269096274834, 0.8414709848078965]

2 .* v = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

1.8 Matrices

- Elements of a given type stored contiguously in memory, with two-dimensional access
- Matrices and 2-dimensional arrays are the same

```
[35]: m1=zeros(4,3)
[36]: 4×3 Array{Float64,2}:
  0.0  0.0  0.0
  0.0  0.0  0.0
  0.0  0.0  0.0
  0.0  0.0  0.0

[37]: m2=Matrix{Float64}(undef, 5, 3)
[38]: fill!(m2,17)
```

Matrix from list comprehension
```plaintext
m3 = [cos(x)*exp(y) for x=0:0.1:10, y=-1:0.1:1]

101×21 Array{Float64,2}:

0.367879  0.40657  0.449329       ...       2.22554  2.4596  2.71828
0.366042  0.404539  0.447084       2.21442  2.44732  2.7047
0.360546  0.398465  0.440372       2.18118  2.41057  2.6641
0.351449  0.388411  0.42926        2.12614  2.34975  2.59687
0.338839  0.374475  0.413859       2.04986  2.26544  2.5037
0.322845  0.356798  0.394323       1.9531   2.1585  2.38552
0.303624  0.335556  0.370847       1.83682  2.03    2.24349
0.28137   0.310962  0.343666       1.70219  1.88121  2.07906
0.256304  0.28326   0.313051       1.55055  1.71362  1.89385
0.228678  0.252728  0.279307       1.38342  1.52891  1.68971
0.198766  0.219671  0.242773       1.20246  1.32893  1.46869
0.166869  0.184418  0.203814       1.0095   1.11567  1.233
0.133004  0.147324  0.162818       0.806442 0.891256 0.98499

-0.318376 -0.35186  -0.388865      -1.92606 -2.12863 -2.3525
-0.335186 -0.370438  -0.409397      -2.02776 -2.24102 -2.47671
-0.348647 -0.385315  -0.425839      -2.10919 -2.33102 -2.57617
-0.358625 -0.393645  -0.438025      -2.16955 -2.39773 -2.6499
-0.365019 -0.405496  -0.445836      -2.20824 -2.44048 -2.69715
-0.367767 -0.406445  -0.449191      -2.22486 -2.45885 -2.71745
-0.368639 -0.40542   -0.448058      -2.21925 -2.45265 -2.71059
-0.362246 -0.400344  -0.442449      -2.19146 -2.42194 -2.67666
-0.354034 -0.391268  -0.432418      -2.14178 -2.36704 -2.61598
-0.342285 -0.378283  -0.418067      -2.0707  -2.28848 -2.52916
-0.327115 -0.361518  -0.399539      -1.97893 -2.18706 -2.41707
-0.308677 -0.341141  -0.377019      -1.86739 -2.06378 -2.28083

size: tuple of dimensions

show size(m3);

size(m3) = (101, 21)

length: number of elements

show length(m3);

length(m3) = 2121

1.8.1 Basic linear algebra

Random vector (normal distribution)

u=randn(5)
v=randn(5);
```
Mean square norm

```plaintext
[43]: using LinearAlgebra
    @show norm(u)
    @show norm(v);

norm(u) = 1.3570160739755728
norm(v) = 3.182905228027219
```

Dot product

```plaintext
[44]: @show dot(u,v);

dot(u, v) = 1.1909146159507276
```

Random matrix (normal distribution)

```plaintext
[45]: m=randn(5,5)
    @show m;

m = [-1.18235628848292 1.3935615113960769 -0.4448929520414901 0.2167564414632219
    -0.4105629595273473; 0.5674080983750002 -1.1787547156382132 0.5639522447990897
    0.6428498807902325 1.5396970344236849; -0.8357860098330051 1.4326364242721084
    -0.6897807210950904 -1.8328245558745698 0.887271577247426; 1.0933232123964831
    0.6249919905304335 0.21892011370402462 0.24385741568274222 1.5170488292113755;
    0.2830487613151486 0.48249542209030194 -1.7557125895509422 -0.35112392161714817
    -1.1261431832859958]
```

Matrix vector multiplication

```plaintext
[46]: @show m*u;

m * u = [1.1708454541930942, -0.14229745117179937, 2.165616862078145,
        1.9457107093390187, 0.05809463771953183]
```

Trace

```plaintext
[47]: @show tr(m);

tr(m) = -3.9331774928194765
```

Determinant

```plaintext
[48]: @show det(m);

det(m) = 14.81487336610951
```

See Cheat Sheet for more

1.9 Control structures

Conditional execution
```python
condition1=false
condition2=true
if condition1
    println("cond1")
elseif condition2
    println("cond2")
else
    println("nothing")
end

for loop
```
Same as

```julia
[52]: for i in 1:10, j in 1:5
    println(i * j)
end
```

1
2
3
Preliminary exit of loop

```plaintext
[53]: for i in 1:10
    println(i)
    if i==3
        break # skip remaining loop
    end
end
```

Preliminary exit of iteration

```plaintext
[54]: for i in 1:10
    if i==5
        continue # skip to next iteration
    end
    println(i)
end
```

1.10 Functions

- All arguments to functions are passed by reference
- Function name ending with ! indicates that the function mutates at least one argument, typically the first
- Function objects can be assigned to variables

Structure of function definition

```plaintext
function func(req1, req2; key1=dflt1, key2=dflt2) # do stuff return out1, out2, out3 end - Required arguments are separated with a comma and use the positional notation - Optional arguments need a default value in the signature - Return statement is optional, by default, the result of the last statement is returned - Multiple outputs can be returned as a tuple, e.g., return out1, out2, out3.
```

Function definition

```plaintext
[55]: function func0(x; y=0)
    println(x+2*y)
end
```
\begin{verbatim}
func0(1)
func0(1,y=1000);
1
2001
Assignment
[56]: f=func0
    f(1);
1
One line function definition
[57]: sin2(x)=sin(2*x)
   @show sin(5)
   @show sin2(5);
sin(5) = -0.9589242746631385
sin2(5) = -0.5440211108893698
Nested function definition
[58]: function outerfunction(n)
    function innerfunction(i)
        println(i)
    end
    for i=1:n
        innerfunction(i)
    end
end
outerfunction(13);
\end{verbatim}
function ff(f0)
    f0(5)
end
fx(x)=sin(x)
@show ff(fx);

ff(fx) = -0.9589242746631385

Anonymous functions

@show ff(x->sin(x));

ff((x->begin
    # = In[60]:1 =#
    sin(x)
    end)) = -0.9589242746631385

1.10.1 Functions on vectors

Dot syntax can be used to make any function work on vectors

x=collect(0:0.1:1)
myf(x)=sin(x)*exp(-x)
@show myf.(x);

myf.(x) = [0.0, 0.09033301095242417, 0.16265669081533915, 0.2189267536743471, 0.261034921143457, 0.29078628821269187, 0.3098823596321072, 0.319909035924728, 0.322328869227062, 0.318476955112901, 0.3095598756531122]

map: apply function to each element of a collection (e.g. matrix, vector)

map(x->sin(x^2), collect(0:0.1:1))

11-element Array{Float64,1}:
  0.0
  0.00999983333416666
  0.03998933418663417
  0.0898754919801104
  0.159318206614246
  0.247403959255452294
  0.35227423327508994
  0.4706258881715797
  0.5971954413623921
  0.7242871743701426
  0.8414709848078965

Equivalent:
map(collect(0:0.1:1)) do x
    return sin(x^2)
end

11-element Array{Float64,1}:
    0.0
    0.00999983334166666
    0.03998933418663417
    0.08987854919801104
    0.159318206614246
    0.24740395925452294
    0.3522742332750899
    0.47062588817115797
    0.5971954413623921
    0.7242871743701426
    0.8414709848078965

mapreduce: apply function to each element of a collection and apply reduction

println(mapreduce(x->sin(x^2), *, collect(0:0.1:1)))
println(mapreduce(x->sin(x^2), +, collect(0:0.1:1)))

0.0
3.5324436045742598

sum: apply function to each element of a collection and add up

sum(x->sin(x^2), collect(0:0.1:1))

3.5324436045742598

This notebook was generated using Literate.jl.