Julia Cheatsheet

Brian Jackson

January 15, 2021

1 Notation

`a,b,c` are scalars, `x,y,z` are vectors, and `A,B,C` are matrices. `S` is a square matrix, `s` is a string. `elw` means element-wise.

2 Assignment

```
a = 1   # scalar assignment
A = B   # alias assignment
A .= B  # element-wise copy
A = copy(B)       # copy
A = deepcopy(B)  # deep copy
a = 1 + 2im  # imaginary
b = 1 // 2   # rational
```

3 Scalar Arithmetic

```
a+2 # addition
a-1 # subtraction
2*a # multiplication
2a  # multiplication
a/3 # float division
a÷3 # int division (\div)
a^2 # exponential
a%3 # modulus
```

4 Arrays

4.1 Initialization

```
[1,2,3]        # vector
[1 2 3]        # row vector
[1; 2; 3]      # col vector
[1 2; 3 4]     # 2x2 matrix
zeros(3)       # all 0s (vec)
zeros(3,2)     # all 0s (mat)
ones(3,2)      # all 1s
ones(Int,3,2) # Integer 1s
rand(3,2)  # uniform from 0-1
randn(3,2) # Std. Gaussian
fill(10,3,2)  # all 10s
0:10     # integer range 0-10
0:2:10   # range 0,2,4,...,10
range(0,10,step=2) # same
0:0.1:10 # step of 0.1
range(0,10,length=101) # same
```

4.2 Arithmetic

```
2 .+ x  # scalar add
2 .- x  # scalar sub
x + y   # elw add
x - y   # elw sub
x .* y  # elw mult
x ./ y  # elw div
x’ * y  # dot product
x’y     # dot product
x * y’  # outer product
x * y   # undefined
A * B   # matrix mult
A .* B  # elw mult
A .^ 2  # elw square
S^2     # matrix mult
```

4.3 Indexing

Assume `size(A) == (3,2)`.

```
x[1]   # linear index
A[4]   # linear index
A[1,2] # row,col (same)
x[2:end]   # 2nd to last
x[1:end-2] # 1st to 3rd last
A[:,1]     # 1st column
A[1:2,:]   # first 2 rows
A[1] = 2   # assign element
A[:,1] .= 2  # assign range
A[:,1] = x   # assign range
```

5 Other Types

5.1 Strings

```
’c’           # char
"my string"  # string
:abc         # symbol (fast)
"my num: $a" # interpolation
string("a","b") # concat
"a" * "b1"      # concat
s[1]            # get char
s[1:2]          # sub-string
```

5.2 Dictionaries

```
d1 = Dict(:a=>1, :b=>2)
d2 = Dict("d"=>x, "e"=>y)
d1[:a]         # indexing
d2["g"] = x+y  # new entry
pop!(d2, "g)   # remove entry
keys(d1)       # get keys
values(d1)     # get values
for (k,v) in pairs(d1)
    # key k, value v
end
```

5.3 Lists

```
[1,2,3]         # good
["a","b","c"]  # good
[[1,2],[2]]    # good
[1,"b",[1,2]]  # avoid
maximum(x) # maximum element
minimum(x) # minimum element
argmax(x)  # index of max
findmax(x) # (val,idx) of max
push!(x,1) # add to end
insert!(x,1,5) # add to start
append!(x,y) # concat
[x; y]        # vert cat
[x y]         # horz cat
vcat(x,y)     # vert cat
hcat(x,y)     # horz cat
a in x        # exists in?
sort(x)       # sort
sort!(x)      # sort in-place
sortperm(x)   # sort indices
```

5.4 Other

```
(1,2,3)       # tuple
Set((1,2,3))  # set
```

6 Control Flow

6.1 Logic

```
a == b  # are equal
A == B  # all elm are equal
isapprox(a, b) # \approx
A === B # same memory loc
a != b  # not equal
a < b   # less than
a <= b  # less than or equal
a && b  # short-circuit and
a || b  # short-circuit or
a < b < c # b between a,c
```

6.2 Conditionals

```
if a < b
    # code
elseif b > a
    # code
else
    # code
end
a < b ? 1 : 0  # inline
(a < b) && 1   # short-circuit
```

6.3 Loops

```
# For loops
for x = 1:10
    # loop body
end
for a in x  # or \in
    # loop body
end
for i = 1:10, j = 1:10
    # nested loop
end

# While Loop
while (a < b)
    a += 1
end

# List comprehension
x = [sin(i) for i = 1:10]
A = [i+j for i in x, j in y]
```

6.4 Functions

```
function myfun(x,y,a=1;b=2)
    # function body
    return <expression>
end
# valid calls
myfun(1,2)
myfun(1,2,3)
myfun(1,2,3,b=3)
myfun(1,2,b=3)

# anonymous functions
mysum(x,y) = x+y
mysub = (x,y) -> x-y
```

7 Linear Algebra

```
using LinearAlgebra
norm(x)     # 2 norm
norm(x,Inf) # Inf norm
norm(x,p)   # p-norm
diag(A)     # get diagonal
inv(S)      # inverse
eigvals(S)  # eigenvalues
rank(S)     # rank
cond(S)     # condition num
isposdef(S) # x’S*x > 0?
Diagonal(x)  # diag mat
Symmetric(S) # symm mat
y = A\x      # solve Ax = y
eigen(S)     # Eigen decomp
qr(S)        # QR fact
svd(S)       # SVD fact
cholesky(S)  # Cholesky
```

8 Useful Macros

8.1 Benchmarking

```
@time f(x)      # print time
@elapsed f(x)   # get time
@allocated f(x) # get allocs

# to run many times
using BenchmarkTools
@btime f(x)     # print time
@benchmark f(x) # get details
```

8.2 Other

```
# get which method is called
@which f(x)

# type stability info
@code_warntype f(x)
```

9 Packages

```
# load package to use
using MyPackage

# shorten name
const MP = MyPackage

# load specific methods
using MyPackage: foo, bar

# load methods to redefine
import MyPackage.foo
```

9.1 Adding/Removing

In REPL, type `]` to open package manager. Here `Pack` can be any package name.

```
add Pack        # add
add Pack@1      # add version
add Pack#master # add branch
rm Pack         # remove
activate dir  # use env at dir
st  # list installed packages
```

10 Type System

10.1 Basic Ops

```
typeof(a) # Float64
typeof(x) # Array{Float64,1}
x isa Vector{Float64} # true
Vector <: Array # true
Int <: Number   # true
```

10.2 Custom Types

```
abstract type Phasors end
struct P1 <: Phasors
    a::Float64
    b::Float64
    isnorm::Bool
end # fields can’t be changed
mutable struct P2 <: Phasors
    a::Float64
    b::Float64
    isnorm::Bool
end # fields can be changed
function foo(x::Phasors)
    # define foo on both
    x.a + x.b  # return this
end
```