

Julia Reference Card

v1.0 for Julia 0.5

(c) 2013 John Lynch modeled on M Goerz's Python card to help map Python to Julia
Information taken liberally from the Julia documentation and various other sources.
You may freely modify and distribute this document.

1 Variable Types

1.0 On All Objects or Collections (c)

is(a, b) or === object identity
isequal(x,y) or == value identity
isa(x, type) test if x is a type
isless(x,y) consistent x<y test
typeof(x) get x's concrete type
tuple(x's) tuple([]...) create tuple
ntuple(f::Function, n) tuple f(i) for i:n
object_id(x) ; hash(x) id equiv to === & ==
copy(x) ; deepcopy(x) shallow or recursive copies
eltype(d) type of elements
eval ; evalfile evaluate expression
collect(c) ; array of all items with
k,v tuples for dicts
Splat c, giving a tuple
Add to s vertically
Add horizontally
Concatenate r+c
empty(c) ; isempty(c)
empty or test c
x in s; in(s,x); !in(s,x)
length(c) ; endof(c)
is x a member of s
length and last index of c
size(c[,d]) size of c in dimension d
sum(c[,d]) ; prod(c[,d])
fn over dimensions
fill!(c,x) fill A with value x

(c...) minimum(c[,d]); maximum
s=[c...,x...] ; s=vcat(c,x)
findmin(c) ; findmax
s=[c x] ; s=hcat(c,x)
hvcate(a,r,c) ; [a b;c d;...]
any(c[,dims]) ; all
count(f(x),x)
first(c) ; last(c)
empty!(c) ; isempty(c)
x in s; in(s,x); !in(s,x)
length(c) ; endof(c)
is x a member of s
length and last index of c
size(c[,d]) size of c in dimension d
sum(c[,d]) ; prod(c[,d])
fn over dimensions
fill!(c,x) fill A with value x

filter(f(x),c) or f(k,v) return items where true
filter!(f(x),c) or update collection
map(f,c) map!(f,c) transformations
reduce(op,v0,c) ; reduce with operator
mapreduce(f,op,c) from init value v0

1.1 Numbers

42 0x2A 0o52 0b101010 42 (dec,hex,oct,bin,)
0.2 .8 4. 1e10 1e-7 3.2f0 floating point value
Inf NaN
z = 5 - 2im complex number
z = complex(real, imag) complex number
real(z); imag(z) real and imag part of z
2//3 - 1//2 rational numbers (gcd)

true; false boolean constants
abs(n) absolute value of n
divrem(x, y) (x/y, x%y)

42 0x2A 0o52 0b101010
cmp(x,y)
round(x,n)
Int() Int8() to Int128()
float("3.14") Float16()
Float32() Float64()
string(3.14)
hex(n) dec(n) oct(n) bin(n)
base(b,n)
int('x')

1.2 Sequences (arrays are mutable, tuples and strings are immutable). 1 dimensional arrays (column) replace vectors and arrays are indexed from 1 to end. Arrays use [] and can be heterogeneous if defined with type Any.

a=l=[1, 2, 3, 4] or [] create 1 dim Array
s=t=(1,"ba",[1+2j,1.4], 4) tuple creation
s=l=linpace(start, stop, n) n items for iteration
l=[t...] ; t=tuple(l...) list / tuple conversion
s=1:1000 range of integers
a=1:1000 1d array of ints
s[3][1] get element (1+2im)
l[end-1][end] get element (1.4)
s[i:j] slicing (i & j inclusive)
s[i:k:j] ; eg s[0:2:10] slice with stride k
s[j:-k:i] ; eg s[9:-1:1] reverse slice eg 9 to 1
s[2:2:end] ; s[1:3:end] every 2nd ; every 3rd
l[i:j]='a','b','c','d' replace equal slice

push!(a,x) x = pop!(a) Add/remove end of a
unshift!(a,x) x = shift!(a) Add/remove start of a
append!(l,l2) ; prepend add items in l2 at end of l
insert!(l,i,x) insert x at pos. i
splice!(a,i:j[,newarray]) remove i to j
reverse!(l,i,j) reverse l from i to j
sort!(l) sort (many options)
zip(s,t,...) [(s[0],t[0],...)]

1.3 Dictionaries

d=Dict{'x'=>42,'y'=>3.14} dict creation
d = Dict{[(i, f(i)) for i=1:n]} using comprehension
[]
d['x'] for inferred types
length(d) get entry for key 'x'
delete!(d,'x') number of keys
has_key(d, k) delete entry from dict
keys(d) does key exist?
values(d) iter of all keys
collect(d) iter of all values
get(d,k,x) array of keys / values
getkey(d,k,x) get value, default x
merge(dict, ...) get key, default to x
pop!(d,k,x) merge dicts
return & delete item

42 (dec,hex,oct,bin,)
x<y: -1, x==y: 0, x>y: 1
round x to n dec places
int from string or num
float from string or num
conversion
create hex, dec, oct,
base b string
code point of char

1.4 Sets

s=Set(s) create set
s=IntSet(i) create sorted int set
issubset(s,t); s<=t all s in t?
union!(s,t) array if t is array
intersect(s,t) elements in s and t
setdiff!(s,t) | (s,c) all s not in t
symdiff!(s,t) | (s,n) (s,c) all either s excl or t
complement!(s) set-complement intset

1.5 Strings and Regular Expressions

"bla"; 'hello "world"' string (of bytes)
\\ backslash
\N{id} \uhhhh \Uhhhhhhh unicode char
\xhh hex
'\u78' '\u2200' '\U10ffff' unicode string
@sprintf("%Fmt", args...) string formatting
%s %03d %.2f %+.0e %E string, int 3char + lead
zero, float 2 precision
var interpolation
t="eat" ; "\$t here"
"\${y+5} is the result"
s*s ; *(s,s1,s2) concatenate strings
s^n ; ^(s,n) repeat s n times
join((s,s,s),sep) join string with separator
collect(s) return an array from
String(s) s to utf-8 string
char(i) char from code point

Other String Methods:

search & replace: search(s,pat,i), rsearch(s,pat,i),
contains(s, pat) index(s,pat,i), rindex(s,pat,i),
beginswith(s,pat), endswith(s,pat),
replace(string, pat, r[, n])
formatting: lowercase, uppercase, ucfirst, lcfirst
splitting: split(s,m), rsplit(s,m), chop, chomp
padding: lpad(s,n,p), rpad(s,n,p), lstrip(s,c),
rstrip(s,c), strip(s,c)
checking: isalnum, isalpha, isascii, isblank, iscntrl,
isdigit, isgraph, islower, isprint, ispunct,
isspace, isupper, isxdigit

Regexes:

rm=match(r"regex",s,i) 1st. nothing if no match
rm.match substring matched
rm.captures tuple of matches
rm.offset offset to match
rm.offsets vector of offsets
matchall(r"",s) -> [s s ...] vector of matches
eachmatch(r"",s[,o]) -> iter iterator over matches
flags after the double quote
i case insensitive
m multiline string
s single line string
x ignore whitespace

1.6 Arrays (homogeneous & type may be specified)/

Array(T, dims) Uninitialized dense dim
 array of Type T

Initialize different arrays (sometimes with T else just dims):

zeros ones trues falses rand randn
eye eye(n) linspace(start, stop, n)

Vector = 1 dim column array is like a list in Python

Functions on arrays:

nnz (num non zero values) stride(A,n) strides(A)
ndims, transpose & ctranspose .' & '

Array(Any, 10, 1) uninitialized 1d (list)
Array{Int32,0} [] Empty array
reshape(A, dims) new shape, same data
similar reinterpret
[a...,x] [a x] add element
[f(x,y) for x=rx, y=cy] array comprehensions
 for x in rx for y in cy] and include filters

1.7 DataFrames (using DataFrames)

DataArray NA NAtype Array with missing values
DataFrame(A=1:4,B=[...]) Tabular hetero dataset
removeNA replaceNA(dv,val) remove or replace NAs
failNA
df[2,"A"] df[[rows],1:2] get & slice
df[1:2,["A","B"]]
df[df["A"] % 2 .== 0, :]
colnames!(df[,newnames]) read or insert col names
head, tail, describe
join(df..., jointype) join two dataframes
groupby(df,catvar) split df by categorical var
by(df,catvar, df->f(df[])) split and apply fn or
by(df,catvar,:(n=...; m=...)) expression to subset
stack(df,categorical var) reshape data
readtable(fname,header=false, defaults are true, comma
 separator='\t') also, writetable(f, fname)

2 Basic Syntax

if expr statements Conditional
elseif expr statements ; if on same line
else statements end terminate with end
z = cond ? x : y ternary version
z = ifelse(cond, x, y) as ? but all args evaluated
z = cond && x short circuit
sleep(0.1) zzzz
nothing empty statement

while expr statements end while loop
while true .. if cond break do .. while equivalent
for target in iter for loop
 statements ; end
for i=itr, j=itr ... end over multiple variables
for key in keys(d)... over dictionary
break, continue end loop / jump to next
s=start(I);while !done(I,s) iterator from sequence
 (i,s) = next(I,s)
print("hello world") print or println (new line)

[expr for x in seq lc if y] list comprehension
function f(params) ... end function definition
f(x, y=0) = return x+y optional parameter
f(a,b,c...) = varargs c = [] or ()
f(a,b; dir="nth") = named args
f(a,b; d=5,e...) = varargs as of k,v tuples
... |> (x,f) apply fn to preceding args
f(1,1), f(2), f(y=3, x=4) function calls
(x, y) -> x*y+a anonymous function

function make_adder_2(a) closure, alternatively,
 add(b) = return a+b function mkadr(a)
 return add b -> a+b
end end
let v=1,w=" " ... ; end scope block with vars
@time() report time elapsed
@profile fn Profile.print() profile, print & clear

global v bind to global variable
: ... ; quote ... end create an expression
eval(expr) evaluate expression

using name fn() load module namespace
import name name.fn() import gives named access
require(filepath) Load file once
reload(filepath) and reload it
include(filepath) set dir & load source
evalfile(file) execute file
cd("data") safely write file in a
 directory and close after.
open("outfile", "w") do f open file
 write(f,data) write to file
end

4 Exception Handling & Debugging

try ... Try-block
catch y catch exception as var

 print data
 error("...")

end exception handling
finally ... in any case
assert(expression) debug assertion
throw(e()) explicit exception
using Debug loads the debugger
@debug @bp before module, set breakpoint
 list lines, print vars
 step into, continue to @bp

5 System Interaction

run(`cmd`) or ;cmd system call
spawn(`cmd`) run asynchronously
success(`cmd`) bool for exit condition
process_running(process) determine if running
process_exited(process) determine if has exited
kill(process, signal) (its stdout, process)
readfrom(command) runs asynch & returns
writeto(command)

readandwrite(command) (its in, its out, process)
detach(command) run & outlive Julia process
setenv(command,env) set vars for running
ENV EnvHash->EnvHash Sys environment vars
getpid() get Julia's pid
clipboard(x) print x to clipboard
s = clipboard() or s from clipboard
@time() @elapsed()->secs time and expression
strftime([f,]time()) time as string
cd(f[,dir]) run f in temporary dir

Filesystem Operations

gethostname(), getipaddr(), pwd(), cd('dir'),
mkdir(p,mode), mkpath(p,mode), rmdir(p),
ignorestatus(cmd),
redirect in run commands: |> std output, |>> append
stdout, |.> stderr to process, file or DevNull

6 Input/Output

open(filename, mode) open file (a & w create,
mode = r r+ w w+ a a+ + is both r&w, w truncates)
open(f[,args]) f(result of open args)
close(stream) flush and close
write(stream, x) write binary x to stream
writedlm writescv array, dlm with csv delimiter
read(stream, type[,dims]) read value from stream
readbytes readdlm readcsv nb bytes, array, csv
readall readline[s] all as string, line or lines
position(s) get position of a stream
seek(s, pos) seek stream to position
seekstart(s), seekend(s) to start to end
skip(s, offset) seek relative
isopen eof isreadonly open? end of file? read only?
ltoh(x) htol(x) little endian conversions
[de]serialize(stream,val) [de]serialize stream
download(url[,localfile]) unix download
+ others

7 Areas Not Covered

Julia has a dynamic type system but with a rich language of types including parametric. Type declaration is optional so the casual user can usually ignore it.
Multiple dispatch permits methods to be called based on the types of all unnamed arguments.
Object orientated design can be achieved by combining type definition and multiple dispatch to associate methods with new classes of objects
Tasks or Coroutines permit computations to be flexibly suspended and resumed, effectively enhanced generators.
Parallel and distributed computing and metaprogramming are supported.

?name get help on object
apropos("search string") search docs for string