Julia 0.2 Reference Card v0.3

1 Variable Types

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)
`hex(n)  dec(n)  oct(n)`	create hex, dec, oct,
`base(b,n)`	base b string
`int('x')`	code point of char
`round(x,n)`	round x to n dec places
`cmp(x,y)`	x<y: -1, x==y: 0, x>y: 1
`isa(x, type)`	test if x is a type
`convert(type, x)`	convert x to type
`int() int8() to int128()`	int from string or num
`float("3.14")  float16()`	float from string
`float32() float64()`	

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 [] but heterogeneous arrays, cells, use {} and can replace lists.

`a=l=[1, 2, 3, 4] or []`	create 1 dim array
`s=l={1,"ba",{1+2im,1.4}, 4}`	list creation
`s=t=(1,"ba",[1+2J,1.4], 4)`	tuple creation
`s=l=linspace(start, stop, n)`	
`fill!(A,x)`	fill A with value x
`l=[t...] ; t=tuple(l...)`	list/tuple conversion
`s=1:1000`	range of integers
`a=[1:1000]`	1d array of ints
`s=start(I);while !done(I,s)`	iterator from sequence
`(i,s) = next(I,s)`	
`s[3][1]`	get element (1+2im)
`l[end-1][end]`	get element (1.4)
`s[i:j]; s[i:]; s[:j]`	slicing (i & j incl.)
`s[i:k:j]`	slice with stride k
`S[2:2:]; s[end:-1:1]`	every 2nd ; reverses
`x in s; in(s,x);  !in(s,x)`	is x a member of s?
`size(s,1)`	number of elements/rows
`minimum(s); maximum(s)`	min/max
`l[i:j]=['a','b','c','d']`	replace equal slice
`in(x,l) ; x in l`	x is contained in l
`count(f(x),x)`	# where f(x) is true
`index(l,x)`	highest index of x, or 0
`append!(l,l2)`	append l2 at end of l
`d=[s1,s1]`	sequence concat
`s=[s,x] ; s=vcat(s,x)`	Add element to s
`push!(a,x)  x = pop!(a)`	Add/remove end of a
`unshift!(a,x)  x = shift!(a)`	Add/remove start of a
`insert!(l,i,x)`	insert x at pos. i
`splice(a,i:j[,newarray])`	remove i to j
`reverse!(1,i,j)`	reverse l from i to j
`sort(l)`	sort (many options)
`zip(s,t,...)`	[(s[0],t[0],...),..]

1.3 Dictionaries

`d={'x'=>42,'y'=>3.14,'z'=>7}`	dict creation
`d={d[i][1] => d[i][2]` `for i=1:length(d)}`	creation with dict comprehension
`d['x']`	get entry for 'x'
`length(d)`	number of keys
`delete(d,'x')`	delete entry from dict
`copy(d)`	create shallow copy
`has_key(d, k)`	does key exist?
`keys(d)`	iter of all keys
`values(d)`	iter of all values
`collect(d)`	array of keys / values
`get(d,k,x)`	get entry for k, or return default x
`getkey(d,k,x)`	get key
`merge(collection, ...)`	merge dicts
`pop!(d,k,x)`	return & delete item
`filter!(f(k,v),d)`	return when fn is true

1.4 Sets - functions that can act on data in place have an !

`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)`	all s not in t
`symdiff!(t)`	all either s excl or t
`copy()`	shallow copy of s
`complement!(s)`	set-complement intset

1.5 Strings and Regular Expressions

`"bla"; 'hello "world"'`	string (of bytes)
`\\`	backslash
`\N{id} \uhhhh \Uhhhhhhhh`	unicode char
`\xhh`	hex
`'\u78' '\u2200'  '\U10ffff'`	unicode string
`string(3.14)`	conversion
`@sprintf("%Fmt", args...)`	string formatting
`%s  %03d  %.2f  %+.0e  %E`	string, int 3char + lead zero, float 2 precision
`t="eat" ; "$t here"`	var interpolation
`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
`utf8(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)`	Uninitalized dense dim array of Type T

Initialize different arrays (sometimes with T else just dims):
`zeros ones trues falses rand randdf randn eye eye(n) linspace(start, stop, n)`
Functions on arrays:
`fill!(A,x) size(A,n) eltype length ndims, nnz (num non zero values) stride(A,n) strides(A) transpose & ctranspose .' & '`

`cell(dims ...)`	uninitialized heterogeneous array
`Array(Int32,0)  []  {}`	Empty array or cell

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

`[A,x]  {C,x}`	add element
`hcat(a,r) or [a b c ..]`	add row to an array
`vcat(a,c) or [a,b,c, ..]`	add column to an array
`hvcat(a,r,c) [a b; c d; ...]`	Concatenate r+c
`reshape(A, dims)`	new shape, same data
`copy(A) deepcopy(A)`	shallow and recursive cp
`similar reinterpret`	
`A = [F(x,y,...) for x=rx, y=ry, ...]`	array comprehensions
`A=map(f,A) map!((x)->f(x),A)`	transformations
`reduce(op,v0,A) ; mapreduce()`	reduce with operator

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
`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`	goo.gl/2FjAI7
`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
`if is(a, b) ...`	object identity
`if a == 1`	value identity
`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
`print("hello world")`	print or println (new line)
`[ expr for x in seq lc ]`	list comprehension
`nothing`	empty statement
`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
`f(1,1), f(2), f(y=3, x=4)`	function calls
`yield()`	switch back to scheduler
`global v`	bind to global variable
`function make_adder_2(a)`	closure, alternatively,
`  add(b) = return a+b`	function mkadr(a)
`  return add`	b -> a+b
`end`	end
`x -> x+a`	anonymous function
`: ... ; 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
`reload(filepath)`	and reload it
`include(filepath)`	set dir & load source
`evalfile(file)`	execute file
`cd("data") do`	safely write file in a
`  open("outfile", "w") do f`	directory and close after.
`  write(f,data)`	
`  end`	
`end`	
`let v=1,w=" "... ; end`	scope block with vars
`@time()  gc_disable()`	report time elapsed
`@profile  Profile.print`	profile, print & clear

3 Object Orientation and Modules

4 Exception Handling & Debugging

`try ...`	Try-block
`catch`	catch exception
`  print data`	
`  error("...")`	exception handling
`end`	
`finally ...`	in any case
`@assert expression`	debug assertion
`throw(e)`	explicit expection

5 System Interaction

`run(`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, signum)`	
`readsfrom(command)`	(its stdout, process)
`writesto(command)`	runs asynch & returns (its stdin, process)
`readsandwrite(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 Julias pid
`clipboard(x) or -> s`	x to clipboard or () from
`@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(`*cmnd*`),`

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, + is
`mode = r r+ w w+ a a+`	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 writecsv`	array, dlm with csv delimeter
`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)`	
`download(url[,localfile])`	unix download
`+ others`	

Should I cover:

tasks and coroutines
parametric types and functions
mapping object orientated to multiple dispatch