R Cheat Sheet: Lists

Context: R has two types of vector Indexing examples: one-dimension get

Atomic vectors contain *values* j <- list(a='cat', b=5, c=FALSE)

These values are all of the same type. x <- j$a # puts 1-item char vec 'cat' in x

They are arranged contiguously. Atomic x <- j[['a']] # much the same as above

vectors cannot contain objects. There are x <- j['a'] # puts 1-item list 'cat' in x

six types of atomic vector: raw, logical, x <- j[[1]] # 1-item char vec 'cat' in x

integer, numeric, complex and character. x <- j[1] # puts 1-item list 'cat' in x

Recursive vectors contain *objects*

R has two types of recursive vector: Indexing examples: set operations

- start with example data

Class Example

l <- list(x='a', y='b', z='c', t='d')

list list(a=1, b=2, c=3:10)

- next: use [[$ because specific selection

expression expression(a + b)

l[[6]] <- 'new' # also l[[5]] set to NULL

Lists are an oft-used workhorse in R.

l$w <- 'new-w' # becomes l[[7]] named 'w'

l[['w']] <- 'dog' # l[[7]] set to 'dog'

Lists

- change named values: (note order ignored)

- At top level: 1-dimension indexed object

l[names(l) %in% c('t', 'x')] <- c(1, 2)

that contains objects (not values)

in previous: l$x set to 1 and l$t to 2

- Indexed from 1 to length(list)

- Contents can be of different types

Indexing example: multi-dimension get

- Lists can contain the NULL object

- Indexing evaluated from left to right

- Deeply nested lists of lists possible

- Let's start with some example data ...

- Can be arbitrarily extended (not fixed)

i <- c('aa', 'bb', 'cc') #

j <- list(a='cat', b=5, c=FALSE)

List creation: usually using list()

k <- list(i, j) #list of things

l1 <- list('cat', 5, 1:10, FALSE) # unnamed

- Let's play with this data ...

l2 <- list(x='dog', y=5+2i, z=3:8) # named

k[[1]] -> x # puts the vector from i in x

l3 <- c(l1, l2) # one list partially named

k[[2]] -> y # puts the list from j in y

l4 <- list(l1, l2) # a list of 2 lists

k[1] -> x # puts vec from i into a list

l5 <- as.list(c(1, 2, 3)) # conversion

and puts that list into x

l6 <- append(origL, insertVorL, position)

x <- k[[1]][[1]] #puts the 'aa' vec in x

x <- k[1][1] # same as k[1] - SILLY

Basic information about lists

x <- k[1][1][1][1][1][1] # same as above

Function Returns x <- k[[1]][[2]] # puts the 'bb' vec in x

dim(l) NULL x <- k[1][2] # WRONG: k[1] is 1-item list

is.list(l) TRUE x <- k[1][[2]] # WRONG same as above

is.vector(l) TRUE x <- k[[2]][1] # put list of 'cat' in x

is.recursive(l) TRUE x <- k[[2]][[1]] # put vector 'cat' in x

is.atomic(l) FALSE

is.factor(l) FALSE List manipulation

length(l) Non-negative number 1 Arithmetic operators cannot be applied

names(l) NULL or char vector to lists (as content types can vary)

mode(l); class(l); typeof(l); attributes(l) 2 Use the apply() functions to apply a function to each element in a list:

The contents of a list x <- list(a=1, b=month.abb, c=letters)

print(l) # print vector contents lapply(x, FUN=length) # (list) 1 12 26

str(l); dput(l); # print list structure sapply(x, FUN=length) #(vector) 1 12 26

head(l); tail(l) # first/last items in l # Next eg: passing args to apply fn

Trap: cat(x) does not work with lists y <- list(a=1, b=2, c=3, d=4)

sapply(y, FUN=function(x,p) x^p, p=2)

Indexing: [versus [[versus $ # -> (vector) 1 4 9 16

- use [to get/set multiple items at once sapply(y, FUN=function(x,p) x^p, p=2:3)

Note: [always returns a list # -> (matrix 2x4) 1 4 9 16 / 1 8 27 64

- use [[and $ to get/set a specific item 3 Use unlist to convert list to vector

- $ only works with named list items unlist(x) # -> "1", "Jan", ... "z"

all same: $name $"name" $'name' $`name` Trap: unlist() wont unlist non-atomic

- indexed by positive numbers: *these ones* unlist(list(expression(a + b))) # FAILS

- indexed by negative numbers: *not these* 4 Remove NULL objects from a list

- indexed by logical atomic vector: *in/out* z <- (a=1:9, b=letters, c=NULL)

- an empty index [] returns the list zNoNull <- Filter(Negate(is.null), z)