R Cheat Sheet: Vectors and Lists

Vector: one-dimensional indexed array

```
# - indexed from 1 to length(vector)
# - all items of same basic atomic type
# - has a fixed length once created
v <- 2 # all scalars are length-1 vectors
v1 <- vector(mode='logical', length=5)
v2 <- 1:3 # an integer vector sequence
v3 <- c(1.3, 7, 7/20) # a numeric vector
v4 <- c(v2, v3) # concatenate and flatten
v5 <- c('red', 'white', 'blue') # char vec
v6 <- c(TRUE, FALSE, TRUE) # logical vector
v7 <- c(1+2i, 2, -3+4i) # a complex vector
v8 <- c(a=1, b=2, c=3) # a named vector
v9 <- rep(NA, 3) # vector of 3 repeated NAs
v10 <- as.vector(something) # conversion
```

Basic information about vectors

```
is.vector(v) # -> TRUE or FALSE (see traps)
print(v) # print the contents of the vector
len <- length(v) # get vector length
str(v) # print basic info on vector
head(v); tail(v) # first/last items in v
#mode(v) class(v) typeof(v) yield base info
```

Indexing: [

```
sex <- c('M', 'F', 'M', 'F', 'M', 'M')
sex[1] <- 'F' # change first item
which(sex == 'M') # -> 3, 5, 6
sex == 'M' # -> F, F, T, F, T, T
sex[which(sex == 'M')] <- '?' # F,F,?,F,?,?
sex[sex == '?'] <- 'M' #back to F,F,M,F,M,M
s <- sex[-c(1,2,4)] # exclude items 1, 2, 4
s <- sex[-length(sex)] # exclude last item
a <- c(a=1, b=2, c=3) # named vector len 3
x <- a['b'] # get: x is named vec of len 1
a['c'] <- 6 # set: a[3] now contains int 6
a[names(a) %in% c('a', 'b')] <- c(9, 8)
# [[ works but $ does not work on vectors
```

Most functions/operators are vectorised

```
# with element-by-element operation
c(1,3,5) + c(5,3,1) # -> 6, 6, 6
c(1,3,5) * c(5,3,1) # -> 5, 9, 5
sorted <- sort(v) # vector sorting
```

Traps

```
# 1) Recycling shorter vectors in math ops
c(1,2,3,4,5) + 1 # -> 2, 3, 4, 5, 6
c(1,2,3,4,5) * c(1,0) # -> 1, 0, 3, 0, 5
# 2) Automatic (hidden) type conversion
x <- c(5, 'a') # c() will convert 5 to '5'
x <- 1:3; x[3] <- 'a' # x now '1', '2', 'a'
typeof(1:2) == typeof(c(1,2)) # -> FALSE
# 3) Empty vectors
x <- c() # creates an empty vector
for(i in 1:length(x)) # will loop twice!
for(i in seq_along(x)) # empty vector safe
# 4) Some Boolean ops are not vectorised
c(T,F,T) && c(T,F,F) # -> TRUE !vectorised
c(T,F,T) & c(T,F,F) # -> TRUE, FALSE, FALSE
c(T,F,T) || c(T,F,F) # -> TRUE !vectorised
c(T,F,T) | c(T,F,F) # -> TRUE, FALSE, TRUE
# 5) is.vector() -> TRUE for atomic lists
```

List: a generic vector-like structure

```
# - at top: a 1-dimensional indexed object
# - indexed from 1 to length(list)
# - items can be of many different types
# - deeply nested lists of lists possible
# - can be arbitrarily extended (not fixed)
l1 <- list('cat', 5, 1:10, FALSE) # unnamed
l2 <- list(x='dog', y=5+2i, z=3:8) # named
l3 <- c(l1, l2) # one list partially named
l4 <- list(l1, l2) # a list of 2 lists
l5 <- as.list( c(1, 2, 3) ) # conversion
```

Basic information about lists

```
is.list(l1) # -> TRUE or FALSE
length(l1) # -> length of top level list
names(l1) # yields a char vector of names
# names: unnamed items are presented as ""
names(l2)[names(l2) %in% c('x', 'y')] <-
    c('m', 'n') # change index names
# mode(l2) class(l2) typeof(l2) → "list"
# Also: print(l) head(l) tail(l) str(l)
```

Indexing: [versus [[versus $

```
# use [ to get/set multiple items at once
# use [[ and $ to get/set specific items
# $ only works with named list items

# Let's start with simple set operations
l <- list(x='a', y='b', z='c', t='d')
# next: use [[ $ because specific selection
l[[6]] <- 'new' # also l[[5]] set to NULL
l$w <- 'new-w' # becomes l[[7]] named 'w'
l[['w']] <- 'dog' # l[[7]] now set to 'dog'
# next: use [ because multiple setting
l[l %in% c('b', 'c')] <- 0
# change named values: (note order ignored)
l[names(l) %in% c('t', 'x')] <- c(1, 2)
# in previous: l$x set to 1 and l$t to 2

# One-dimension get operations:
j <- list(a='cat', b=5, c=FALSE)
x <- j$a # puts 1-item char vec 'cat' in x
x <- j[['a']] # much the same as above
x <- j['a'] # puts 1-item list 'cat' in x
# j[1] & j[[1]] behave like named one above

# Multi-dimension get operations
i <- c('aa', 'bb', 'cc') #
k <- list(i, j) #list of things; i,j copied
k[[1]] -> x # puts the vector from i in x
k[1] -> x   # puts vec from i into a list
            # and puts that list into x
x <- k[[1]][[1]] #puts the 'aa' vec in x
x <- k[1][1] # same as k[1] - WRONG
x <- k[[1]][[2]] # puts the 'bb' vec in x
x <- k[1][2] # WRONG puts NULL in x
x <- k[1][[2]] # WRONG a subscript error
x <- k[[2]][1] #puts a list of 1 'cat' in x
x <- k[[2]][[1]] #put 1-item 'cat' vec in x
```

Traps

```
# 1) When using lists, most of the time
#    you want to use [[ or $ and avoid [
```