R Cheat Sheet: Lists

Context: R has two types of vector

Atomic vectors contain *values*

These values are all of the same type. They are arranged contiguously. Atomic vectors cannot contain objects. There are six types of atomic vector: raw, logical, integer, numeric, complex and character.

Recursive vectors contain *objects*

R has two types of recursive vector:

Class	Example
list	list(a=1, b=2, c=3:10)
expression	expression(a + b)

Lists are an oft-used workhorse in R.

Lists

- At top level: 1-dimension indexed object that contains objects (not values)
- Indexed from 1 to length(list)
- Contents can be of different types
- Lists can contain the NULL object
- Deeply nested lists of lists possible
- Can be arbitrarily extended (not fixed)

List creation: usually using list()

```
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
l6 <- append(origL, insertVorL, position)
```

Basic information about lists

Function	Returns
dim(l)	NULL
is.list(l)	TRUE
is.vector(l)	TRUE
is.recursive(l)	TRUE
is.atomic(l)	FALSE
is.factor(l)	FALSE
length(l)	Non-negative number
names(l)	NULL or char vector

mode(l); class(l); typeof(l); attributes(l)

The contents of a list

```
print(l) # print vector contents
str(l); dput(l); # print list structure
head(l); tail(l) # first/last items in l
```

<u>Trap</u>: cat(x) does not work with lists

Indexing: [versus [[versus $

- use [to get/set multiple items at once
  <u>Note</u>: [always returns a list
- use [[and $ to get/set a specific item
- $ only works with named list items
  all same: $name $"name" $'name' $`name`
- indexed by positive numbers: *these ones*
- indexed by negative numbers: *not these*
- indexed by logical atomic vector: *in/out*
- an empty index l[] returns the list

<u>Tip</u>: When using lists, most of the time you want to index with [[or $; and avoid [

Indexing examples: one-dimension get

```
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
x <- j[[1]] # 1-item char vec 'cat' in x
x <- j[1] # puts 1-item list 'cat' in x
```

Indexing examples: set operations

- start with example data
  ```
  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$w <- 'new-w' # becomes l[[7]] named 'w'
  l[['w']] <- 'dog' # l[[7]] set to 'dog'
  ```
- change named values: (note order ignored)
  ```
  l[names(l) %in% c('t', 'x')] <- c(1, 2)
  # in previous: l$x set to 1 and l$t to 2
  ```

Indexing example: multi-dimension get

- Indexing evaluated from left to right
- Let's start with some example data ...
  ```
  i <- c('aa', 'bb', 'cc') #
  j <- list(a='cat', b=5, c=FALSE)
  k <- list(i, j) #list of things
  ```
- Let's play with this data ...
  ```
  k[[1]] -> x # puts the vector from i in x
  k[[2]] -> y # puts the list from j in y
  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] - SILLY
  x <- k[1][1][1][1][1][1] # same as above
  x <- k[[1]][[2]] # puts the 'bb' vec in x
  x <- k[1][2] # WRONG: k[1] is 1-item list
  x <- k[1][[2]] # WRONG same as above
  x <- k[[2]][1] # put list of 'cat' in x
  x <- k[[2]][[1]] # put vector 'cat' in x
  ```

List manipulation

1. Arithmetic operators cannot be applied to lists (as content types can vary)
2. Use the apply() functions to apply a function to each element in a list:
   ```
   x <- list(a=1, b=month.abb, c=letters)
   lapply(x, FUN=length) # (list) 1 12 26
   sapply(x, FUN=length) #(vector) 1 12 26
   # Next eg: passing args to apply fn
   y <- list(a=1, b=2, c=3, d=4)
   sapply(y, FUN=function(x,p) x^p, p=2)
   # -> (vector) 1 4 9 16
   sapply(y, FUN=function(x,p) x^p, p=2:3)
   # -> (matrix 2x4) 1 4 9 16 / 1 8 27 64
   ```
3. Use unlist to convert list to vector
   ```
   unlist(x) # -> "1", "Jan", ... "z"
   ```
   <u>Trap</u>: unlist() wont unlist non-atomic
   ```
   unlist(list(expression(a + b))) # FAILS
   ```
4. Remove NULL objects from a list
   ```
   z <- (a=1:9, b=letters, c=NULL)
   zNoNull <- Filter(Negate(is.null), z)
   ```
5. Use named lists to return multiple values
6. <u>Trap</u>: factor indexes treated as integer
   <u>Tip</u>: decode with v[as.character(f)] etc.