# R Cheat Sheet: Atomic Vectors (often just called "vectors" in R)

## Atomic vectors:

- An object with contiguous, indexed values
- Indexed from 1 to length(vector)
- All values of the same basic atomic type
- Vectors do not have a dimension attribute
- Has a fixed length once created

## Six basic atomic types:

| Class | Example |
|---|---|
| logical | TRUE, FALSE, NA |
| integer | 1:5, 2L, 4L, 6L |
| numeric | 2, 0.77 (double precision) |
| complex | 3.7+4.2i, 0+1i |
| character | "string", 'another string' |
| raw | (byte data from 0-255) |

## No scalars

In R, these basic types are always in a vector. Scalars are just length=1 vectors.

## Creation (length determined at creation)

Default value vectors of length=4

```
u <- vector(mode='logical', length=4)
print(u) # -> FALSE, FALSE, FALSE, FALSE
v <- vector(mode='integer', length=4)
Also: numeric(4); character(4); raw(4)
```

Using the sequence operator

```
i <- 1:5 # produces an integer sequence
j <- 1.4:6.4 # a numeric sequence
k <- seq(from=0, to=1, by=0.1) # numeric
```

Using the c() function

```
l <- c(TRUE, FALSE)    # logical vector
n <- c(1.3, 7, 7/20)   # numeric vector
z <- c(1+2i, 2, -3+4i) # complex vector
c <- c('pink', 'blue') # character vector
```

Other things

```
v1 <- c(a=1, b=2, c=3) # a named vector
v2 <- rep(NA, 3) # 3 repeated NAs
v3 <- c(v1, v2) # concatenate and flatten
v4 <- append(origV, insertV, position)
```

## Conversion

```
as.vector(v); as.logical(v); as.integer(v)
as.numeric(v); as.character(v) # etc. etc.
unlist(l) # convert list to atomic vector
```

**Trap**: unlist() wont unlist non-atomic items

```
unlist(list(as.name('fred')))      # FAILS
```

## Basic information about atomic vectors

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

```
mode(v); class(v); typeof(v); attributes(v)
is.numeric(v); is.character(v); # etc. etc.
```

**Trap**: lists are vectors (but not atomic)

**Trap**: array/matrix are atomic (not vectors)

**Tip**: use (is.vector(v) && is.atomic(v))

## The contents of a vector

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

## Indexing: [ and [[ (but not $)

- [x] selects a vector for the cell/range x
- [[x]] selects a length=1 vector for the single cell index x (*rarely used*)
- $ operator invalid for atomic vectors

Index by positive numbers: *these ones*

```
v[c(1,1,4)] # get 1st one twice then 4th
v[m:n] # get elements from indexes m to n
v[[7]] <- 6 # set seventh element to 6
v[which(v == 'M')] # which() yields nums
```

Index by negative numbers: *not these*

```
v[-1] # get all but the first element
v[-length(v)] # get all but the last one
v[-c(1,3,5,7,9)] # get all but ...
```

Index by logical atomic vector: *in/out*

```
v[c(TRUE, FALSE, TRUE)] # get 1st and 3rd
v[v > 2] # get all where v is g.t. two
v[v > 2 & v < 9] # get where v>2 and v<9
v[v == 'M'] # get where v equals char 'M'
v[v %in% c('me', 'andMe', 'meToo')] # get
```

Indexed by name (*only with named vectors*)

```
v[['alpha']] # get single by name
v[['beta']] <- 'b' # set single by name
v[c('alpha', 'beta')] # get multiple
v[!(names(v) %in% c('a', 'b'))] # exclude
names(v)['z'] <- 'omega' # change name
```

## Most functions/operators are vectorised

```
c(1,3,5) + c(5,3,1) # -> 6, 6, 6
c(1,3,5) * c(5,3,1) # -> 5, 9, 5
```

## Sorting

```
upSorted <- sort(v) # also: v[order(v)]
d <- sort(v, decreasing=TRUE) # rev(sort(v))
```

## Raw vectors (byte sequences)

```
s <- charToRaw('raw') # string input
r <- as.raw(c(114, 97, 119)) # decimal in
print(r) # -> 72 61 77 (hex output)
```

## Traps

Recycling vectors in math operations

```
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
```

Automatic type coercion (often hidden)

```
x <- c(5, 'a') # c() converts 5 to '5'
x <- 1:3; x[3] <- 'a' # x now '1' '2' 'a'
typeof(1:2) == typeof(c(1,2)) # -> FALSE
```

For-loops on empty vectors

```
for(i in 1:length(c())) print(i) # loopx2
for(i in seq_len(x)) # empty vector safe
```

Also: for(j in seq_along(x))

Some Boolean ops not vectorised

```
c(T,F,T) && c(T,F,F) # TRUE (!vectorised)
c(T,F,T) & c(T,F,F) # TRUE, FALSE, FALSE
```

Similarly: || is not vectorised; | is

Factor indexes are treated as integers

**Tip**: decode with v[as.character(f)].