Advanced R - Data structure

Anh Le

August 12, 2014

1 Vector

`is.vector()` does not test if an object is a vector. Instead it returns `TRUE` only if the object is a vector with no attributes apart from names.

```
real_vector <- c(1, 2, 3)
is.vector(real_vector)

## [1] TRUE

attr(real_vector, "someattr") <- "attrvalue"
is.vector(real_vector)

## [1] FALSE

is.atomic(real_vector)

## [1] TRUE
```

1.1 Atomic vector

`NA` is a logical vector of length one. `NA` is coerced to the appropriate type to blend in with the rest of the vector.

```
typeof(NA)

## [1] "logical"

v <- c(1, 2, NA, 3); typeof(v[3])

## [1] "double"

v <- c(1L, 2L, NA, 3L); typeof(v[3])

## [1] "integer"
```

2 List

Lists are sometimes called **recursive** vectors, in contrast to regular **atomic** vectors.

```
x <- list(list(list()))
str(x)

## List of 1
##  $ :List of 1
##   ..$ : list()

is.recursive(x)

## [1] TRUE
```

`c()` combines several list into one. If we try to combine vector and list, the vector will be coerced to list.

```
x <- list(list(1, 2), c(3, 4))
y <- c(list(1, 2), c(3, 4))
str(x)

## List of 2
##  $ :List of 2
##   ..$ : num 1
##   ..$ : num 2
##  $ : num [1:2] 3 4

str(y)

## List of 4
##  $ : num 1
##  $ : num 2
##  $ : num 3
##  $ : num 4
```

3 Exercises

1. What are the six types of atomic vector? How does a list differ from an atomic vector?

   Six types: logical, double, integer, character, and complex, raw. List is recursive, can hold multiple typles

2. What makes `is.vector()` and `is.numeric()` fundamentally different to `is.list()` and `is.character()`?

3. Test knowledge of vector coercion rule

```
c(1, FALSE) # Should be c(1, 0)

## [1] 1 0

c("a", 1) # Should be c("a", "1")

## [1] "a" "1"

c(list(1), "a") # Should be list(1, "a"), cuz "a" is coerced to list first

## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"

str(c(TRUE, 1L)) # Should be c(1L, 1L)

##  int [1:2] 1 1
```

4. Why do you need to use `unlist()` to convert a list to an atomic vector? Why doesn't `as.vector()` work?

   Probably because `list` is already a (recursive, non-atomic) vector. Indeed,

```
l <- list(1, 2)
is.vector(l)

## [1] TRUE

is.vector(l, mode="logical")

## [1] FALSE

is.vector(l, mode="list")

## [1] TRUE

is.vector(l, mode="expression")

## [1] FALSE
```

5. Why is `1 == "1"` true? (Because `1 (double)` is coerced to `"1 (character)"`). Why is `-1 < FALSE` true? (Because `FALSE (logical)` is coerced to `0 (double)`). Why is `"one" < 2` false? (Because `2` is coerced to `"2"`, and strings are compared alphabetically)

6. Why is the default missing value, `NA`, a logical vector? What's special about logical vectors?

   Probably because it's the most flexible, so whenever it's put together with other values in the vectors, `NA` will be coerced, not the other values.

4 Attributes

5 Factors

Factors are built on top of integer vectors using two attributes: the `class()`, "factor", which makes them behave differently from regular integer vectors, and the `levels()`, which defines the set of allowed values.

6 Exercises

1. An early draft used this code to illustrate `structure()`:

```
structure(1:5, comment = "my attribute")

## [1] 1 2 3 4 5
```

   But when you print that object you don't see the comment attribute. Why?

```
x <- structure(1:5, comment="my attribute")
comment(x)

## [1] "my attribute"
```

   Turns out `comment` is a special attribute that does not get printed. See `help(comment)`.

2. What happens to a factor when you modify its levels?

```
f1 <- factor(letters)
levels(f1) <- rev(levels(f1))
f1

##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

Both the observations and the level labels are switched to the new levels.

3. What does this code do? How do `f2` and `f3` differ from `f1`

```
f2 <- rev(factor(letters))
f3 <- factor(letters, levels=rev(letters))
f2
```

```
##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: a b c d e f g h i j k l m n o p q r s t u v w x y z
```

```
f3
```

```
##  [1] a b c d e f g h i j k l m n o p q r s t u v w x y z
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

`f2` only switches the order of the observations, the levels is the same as `f1`. `f3` has the same observations as `f1`, but the levels is reversed.

7 Matrices and arrays

8 Exercises

1. What does `dim()` return when applied to a vector?

   It returns `NULL`

2. If `is.matrix(x)` is `TRUE`, what will 'is.array(x)' return?

   Must also be `TRUE`.

3. How would you describe the following three objects? What makes them different to `1:5`?

```
x1 <- array(1:5, c(1, 1, 5))
x2 <- array(1:5, c(1, 5, 1))
x3 <- array(1:5, c(5, 1, 1))
x1 # 5 1x1 matrices
```

```
## , , 1
##
##      [,1]
## [1,]    1
##
## , , 2
##
```

```
##      [,1]
## [1,]    2
##
## , , 3
##
##      [,1]
## [1,]    3
##
## , , 4
##
##      [,1]
## [1,]    4
##
## , , 5
##
##      [,1]
## [1,]    5
```

```
x2 # 1 1x5 matrix
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
```

```
x3 # 1 5x1 matrix
```

```
## , , 1
##
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
## [5,]    5
```

9 Data frames

Under the hood, a data frame is a list of equal-length vectors.

A data frame share properties with both the matrix and the list. For example, `length()` of a data frame is the length of the underlying list, i.e equivalent to `ncol()`. `names()` is equivalent to `colnames()`.

Same idea with subsetting – we can subset a dataframe both in list-way

(`df$col`) and matrix-way (`df[x, y]`).

Data frame is S3 class, thus its type reflects the underlying vector to build it, which is a list. Use `class()` and `is.data.frame()` to test.

```
df <- data.frame(x=c(1,2), y=c(3,4))
typeof(df)

## [1] "list"

class(df)

## [1] "data.frame"

is.data.frame(df)

## [1] TRUE
```

10 Exercises

1. What attributes does a data frame possess?

```
df <- data.frame(x = c(1,2))
attributes(df)

## $names
## [1] "x"
##
## $row.names
## [1] 1 2
##
## $class
## [1] "data.frame"
```

2. What does `as.matrix()` do when applied to a data frame with columns of different types?

   Probably coerced to the least flexible?

```
df <- data.frame(x = c(1,2), y=c("a", "b"))
as.matrix(df)

##      x   y
## [1,] "1" "a"
## [2,] "2" "b"
```

3. Can you have a data frame with 0 rows? What about 0 columns?

```
df_norows <- data.frame(x=numeric(0), y=numeric(0))
df_norows

## [1] x y
## <0 rows> (or 0-length row.names)

# Cannot have no columns by itsef?
# data.frame() returns a df with 0 col and 0 row
```

11 Subsetting and assignment

You can't combine integer indices with NA but you can combine logical indices with NA (where it is treated as FALSE)

```
x <- c(1, 2, 3, 4)
x[c(1, NA)] <- 10
x(c(T, F, NA, NA, T)) <- 10

## Error:  could not find function "x<-"

x

## [1] 10  2  3  4
```