Advanced R - Chapter 1 - Data Structures

Matthew Strimas-Mackey

Data structures

Quiz

1. What are the three properties of a vector, other than its contents?

- length
- type

1. What are the four common types of atomic vectors? What are the two rare types?

- character
- boolean
- numeric
- dimension

2. What are attributes? How do you get them and set them?

3. How is a list different from an atomic vector? How is a matrix different from a data frame?

- Only one datatype allowed in vector and matrix

4. Can you have a list that is a matrix? Can a data frame have a column that is a matrix?

- No

Vectors

Vectores can be atomic vectors, i.e. all elements of the same type, or lists, with mixed types. They have 3 basic properties:

- Type, `typeof()`, what it is.
- Length, `length()`, how many elements it contains.
- Attributes, `attributes()`, additional arbitrary metadata.

```
v <- c(1,2,3)
names(v) <- c('one', 'two', 'three')
l <- list(v)

typeof(v)
```

```
## [1] "double"
```

```
length(v)
```

```
## [1] 3
```

```
attributes(v)
```

```
## $names
## [1] "one"   "two"   "three"
```

```
# Testing identity
is.vector(v) # Vector with no attributes other than name
```

```
## [1] TRUE
```

```r
is.atomic(v) # Atomic vector
```

```
## [1] TRUE
```

```r
is.list(l) # List
```

```
## [1] TRUE
```

```r
is.atomic(v) || is.list(v) # Either type of vector
```

```
## [1] TRUE
```

Atomic vectors

Four common types: logical, integer, double (often called numeric), and character. Two rare types: complex and raw.

Atomic vectors are usually created with `c()`, short for combine:

```r
# By default all numbers are stored as doubles
dbl_var <- c(1, 2.5, 4.5)
# With the L suffix, you get an integer rather than a double
int_var <- c(1L, 6L, 10L)

# Missing values specified with NA
miss <- c(1, 2, 3, NA)
# NA a logical by default, but always coerced to correct type, but can also specify explicitely
c(1, 2, 3, NA_real_)
```

```
## [1]  1  2  3 NA
```

```r
c(1L, 2L, 3L, NA_integer_)
```

```
## [1]  1  2  3 NA
```

```r
c('1', '2', '3', NA_character_)
```

```
## [1] "1" "2" "3" NA
```

Types and tests

Given a vector, you can determine its type with `typeof()`, or check if it's a specific type with an "is" function: `is.character()`, `is.double()`, `is.integer()`, `is.logical()`, or, more generally, `is.atomic()`.

```r
int_var <- c(1L, 6L, 10L)
typeof(int_var)
```

```
## [1] "integer"
```

```r
is.integer(int_var)
```

```
## [1] TRUE
```

```r
is.atomic(int_var)
```

```
## [1] TRUE
```

```r
dbl_var <- c(1, 2.5, 4.5)
typeof(dbl_var)
```

```
## [1] "double"
```

```r
is.double(dbl_var)
```

```
## [1] TRUE
```

```r
is.atomic(dbl_var)
```

```
## [1] TRUE
```

```r
# is.numeric() returns T for double or integer
is.numeric(int_var)
```

```
## [1] TRUE
```

```r
is.numeric(dbl_var)
```

```
## [1] TRUE
```

Coercion

When you attempt to combine different types they will be **coerced** to the most flexible type. Types from least to most flexible are: logical, integer, double, and character. If confusion is likely, explicitly coerce with `as.character()`, `as.double()`, `as.integer()`, or `as.logical()`.

Lists

Lists are vectors whose elements can be of any type, including lists. `list()` creates a list out of it's elements, `c()` will combine lists together, `unlist()` turns a list into an atomic vector, making the required coercions.

```r
z <- list(1:3, "a", c(TRUE, FALSE, TRUE), c(2.3, 5.9))
str(z)
```

```
## List of 4
##  $ : int [1:3] 1 2 3
##  $ : chr "a"
##  $ : logi [1:3] TRUE FALSE TRUE
##  $ : num [1:2] 2.3 5.9
```

```r
# Difference between list and c
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
```

```r
str(y)
```

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

```r
# unlist
str(unlist(x))
```

```
##  num [1:4] 1 2 3 4
```

```r
str(unlist(z)) # Coercion to string
```

```
##  chr [1:9] "1" "2" "3" "a" "TRUE" "FALSE" "TRUE" "2.3" "5.9"
```

Exercises

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

- double, integer, logical, character; complex, raw
- elements of a list can be of different types; atomic vectors are all of the same type

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

- the first 2 will match 2 types of data, while the second will only match one each

3. Test your knowledge of vector coercion rules by predicting the output of the following uses of `c()`:

```r
c(1, FALSE) # c(1,0)
```

```
## [1] 1 0
```

```r
c("a", 1) # c('a', '1')
```

```
## [1] "a" "1"
```

```r
c(list(1), "a") # list(1, 'a')
```

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

```r
c(TRUE, 1L) # c(1L, 1L)
```

```
## [1] 1 1
```

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

- a list is already a vector

5. Why is `1 == "1"` true? Why is `-1 < FALSE` true? Why is `"one" < 2` false?

- Coercion

6. Why is the default missing value, `NA`, a logical vector? What's special about logical vectors? (Hint: think about `c(FALSE, NA_character_)`.)

- logical is the least flexible type

Attributes

All objects can have arbitrary additional attributes, used to store metadata about the object. Attributes can be thought of as a named list (with unique names). Attributes can be accessed individually with `attr()` or all at once (as a list) with `attributes()`.

```r
y <- 1:10
names(y) <- 1:10
attr(y, "my_attribute") <- "This is a vector"
attr(y, "my_attribute")
```

```
## [1] "This is a vector"
```

```
str(attributes(y))
```

```
## List of 2
##  $ names       : chr [1:10] "1" "2" "3" "4" ...
##  $ my_attribute: chr "This is a vector"
```

```
str(y)
```

```
##  Named int [1:10] 1 2 3 4 5 6 7 8 9 10
##  - attr(*, "names")= chr [1:10] "1" "2" "3" "4" ...
##  - attr(*, "my_attribute")= chr "This is a vector"
```

Names

You can name a vector in three ways:

```
# When creating it
c(a = 1, b = 2, c = 3)
```

```
## a b c
## 1 2 3
```

```
# Modifying in place
x <- 1:3; names(x) <- c("a", "b", "c"); x
```

```
## a b c
## 1 2 3
```

```
setNames(1:3, c("a", "b", "c"))
```

```
## a b c
## 1 2 3
```

Factors

A factor is a vector that can contain only predefined values, and is used to store categorical data. 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.

```
x <- factor(c("a", "b", "b", "a"))
x
```

```
## [1] a b b a
## Levels: a b
```

```
class(x)
```

```
## [1] "factor"
```

```
levels(x)
```

```
## [1] "a" "b"
```

```
# You can't use values that are not in the levels
x[2] <- "c"
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = "c"): invalid factor level, NA
## generated
```

```
x
```

```
## [1] a    <NA> b    a
## Levels: a b
```

Factors are useful when you know the possible values a variable may take, even if you don't see all values in a given dataset. Using a factor instead of a character vector makes it obvious when some groups contain no observations:

```
sex_char <- c("m", "m", "m")
sex_factor <- factor(sex_char, levels = c("m", "f"))

table(sex_char)
```

```
## sex_char
## m
## 3
```

```
table(sex_factor)
```

```
## sex_factor
## m f
## 3 0
```

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? Is the attribute missing, or is there something else special about it? (Hint: try using help.)

- comment attributes are not printed by default

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

   ```
   f1 <- factor(letters)
   levels(f1) <- rev(levels(f1))
   levels(f1) <- 1:26
   ```

- the mapping between integers and levels is changed, so the vector is now labelled wrong

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

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

- First changes order of vector, second changes the order of the integers that are used to represent the levels