Chapter 2: Data Structures

Emeka Mbazor

3/23/2020

Vector Exercises

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

The six types of atomic vectors are: * logical * integer * double * character * complex * real

An atomic vector can only contain elements of a single type while a list can contain elements of any type (including lists themselves).

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

`is.vector()` returns true if the object is an atomic vector or a list and `is.numeric` returns true if the object is an integer vector or a double vector. But `is.list()` only returns true if the object is a list and `is.character()` only returns true if the object is a character vector.

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

```
c(1, FALSE)
```

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

```
c("a", 1)
```

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

```
c(list(1), "a")
```

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

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

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

```
[1, 0]
["a", "1"]
[ {1}, {"a"}]
[1, 1]
```

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

`as.vector()` doesn't convert a list to an atomic vector because a list is already a vector.

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

`1 == "1"` is true because 1 is converted into the character `"1"` and "1" is equal to itself. `-1 < FALSE` is true because `FALSE` is converted into the integer `0` and -1 is less than 0. `"one" < 2` is false because 2 is converted into the character `"2"` and 2 comes before "o" on the ASCII table.

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

`NA` is a logical vector because logical vectors are the most flexible.

Factors 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.)

The `comment` attribute is special because it's not printed by `print` or `print.default`

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

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

When you modify a factor's levels the integer values stay the same but the levels are changed.

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

```
f2 <- rev(factor(letters))

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

The first line reverses the levels after the factor is made and makes the data appear reversed as a result. The second line creates a factor with the levels already reversed so the data remains intact (in its original order).

Matrices and Arrays Exercises

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

```
a <- 1:3
dim(a)
```

```
## NULL
```

`dim()` returns `NULL` when applied to a vector.

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

`is.array(x)` should return true because a matrix is just a special kind of array.

```
x <- matrix(1:12, nrow = 2, ncol = 6)
is.matrix(x)
```

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

```
is.array(x)
```

```
## [1] 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))

str(x1)
```

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

```
str(x2)
```

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

```
str(x3)
```

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

The following objects are three dimensional arrays and they differ from 1:5 in structure.

Data frames Exercises

1. What attributes does a data frame possess?

- names
- dimensions

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

```
a <- 1:3
b <- c("1", "2", "3")
dfr <- data.frame(a, b)

dfr
```

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

```
matrix <- as.matrix(dfr)
matrix
```

```
##      a   b
## [1,] "1" "1"
## [2,] "2" "2"
## [3,] "3" "3"
```

`as.matrix` coerces all values into the most flexible type.

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

```
a <- c()
b <- c()

str(data.frame(a,b))
```

```
## 'data.frame':    0 obs. of  0 variables
```

Yes.