Chapter 2 Exercises

Derek Chiu

July 14, 2016

2.1.3 Exercises

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

   The six types of atomic vector are `integer`, `double`, `character`, `logical`, `complex`, and `raw`. A list can have elements of different types whereas an atomic vector can only have elements of one type.

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

   The first two checks have names that don't verify exactly what they describe, whereas the latter two checks indeed verify the types described.

3. Test your knowledge of vector coercion rules by predicting the output of the following uses of `c()`: `c(1, FALSE) c("a", 1) c(list(1), "a") c(TRUE, 1L)`

   ```
   c(1, 0)
   c("a", "1")
   list(1, "a")
   c(1L, 1L)
   ```

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

   Because `is.vector()` returns true for both lists and atomic vectors, `as.vector()` doesn't change the structure of a list to an atomic vector.

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

   The logical operator `==` coerces `1` to a `"1"`. The logical operator `<` coerces `FALSE` to a `0`. The logical operator `<` coerces `2` to `"2"`, which is alphabetically not greater than `"one"`.

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

   So that it has the highest flexibility of coercion amongst the four most common types of atomic vectors.

2.2.2 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 treated specially and is thus restricted to the values that can be set.

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

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

The order of elements changes as well.

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` has elements reversed but with levels in the original order.
`f3` has levels reversed but with elements in the original order.

2.3.1 Exercises

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

`NULL`.

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

`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` is `1:5` as a 5-dimensional array of 1-by-1 matrices.
`x2` is `1:5` as a row vector.
`x3` is `1:5` as a column vector.

2.4.5 Exercises

1. What attributes does a data frame possess?

A data frame can have heterogeneous column types while possessing the structure of a matrix.

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

Coerces all columns to one type.

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

Yes.