Advanced R

Yu Chen

August 29, 2017

Data Structures

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

The six types of atomic vectors are **logical**, **integer**, **double** (or **numeric**), **character**, **complex**, and **raw**. All elements of an atomic vector must be the same type, whereas the elements of a list can be of different types.

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

The function **`is.vector()`** encompasses **`is.list()`**, since a **list** object in R is a type of vector. If you attempt to create an **atomic vector** using different types, the object created will be coerced into **character** type:

```
vec <- c(1,2,3)
is.vector(vec) # TRUE
is.list(vec) # FALSE
is.vector(vec, mode = "list") # FALSE
is.character(vec) # FALSE
is.numeric(Vec) # TRUE
```

You can see here than an atomic vector of integers is a **vector**, not a **list**, and also not of the **character** type. It is, however, **numeric**.

```
vec2 <- list(1,2,3)
is.vector(vec2) # TRUE
is.list(vec2) # TRUE
is.character(vec2) # FALSE

vec3 <- c(1,"ab", TRUE)
is.vector(vec3) # TRUE
is.list(vec3) # FALSE
is.character(vec3) # TRUE
str(vec3)
```

In the final example, when

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

The first example, **`c(1, FALSE)`**, will return an atomic vector with two elements: `1`, and `0`, since `FALSE` can be coerced into an integer.

The second example, **`c("A", 1)`**, will be coerced into characters, since it is of both `character` and `integer` type.

The third example, **`c(list(1))`** will return an atomic vector with one element, a `list`.

The fourth example, **`c(TRUE, 1L)`** will return an atomic vector with two elements, both of `1`. As a side note, what does the `L` after a number mean? It means that the number is an **explicit integer**. There are some use cases for this- for example, with performance:

```
x <- 1:100
typeof(x) # integer

y <- x+1
typeof(y) # double, twice the memory size
object.size(y) # 840 bytes (on win64)

z <- x+1L
typeof(z) # still integer
object.size(z) # 440 bytes (on win64)
```

Why do you need to use `unlist()` to convert a list to an atomic vector? Why does `as.vector()` work?