Atomic Vectors in R

We'll start with a discussion of the type system in R. Afterward, we'll move into a discussion of atomic vectors and what you can do with them.

Types in R

The concept of *data types* is fundamental to programming.[1] Every "basic unit" of data—every variable in R—has a particular *type*, which tells the processor what you can or can't do with it.

To determine the type of a variable, you can use the `typeof()` function. Although `typeof()` can in fact return any of 24 different types, in practice we'll only concern ourselves with a couple of them.[2] In particular, we'll begin by focusing on the "integer", "double" (floating-point number), "character" (string), and "logical" (boolean) types.

The first two of those allow you to do simple arithmetic in R, *e.g.* you can type in `1+2` into the console and get `3` back. Let's try testing the type of a floating-point number:

```
> typeof(5.5)
[1] "double"
```

(In computer-science parlance, "double" refers to the double-precision floating-point format).

Exercise. First, pick an integer and try running `typeof()` on it. Next, try running `5.5L` and `typeof(5.5L)`. Observe both the *output* and the *warning message* of both commands. Can you figure out how to pass in an integer into `typeof()` so that it returns `"integer"` for the type?[3]

In practice, you don't actually care about the difference between how R handles integers and how R handles doubles. In a certain precise sense, "numeric" objects are a generalized structure which encapsulates all sorts of real numbers (integers, doubles, and more). So, really, what we want to think about are numerics, characters, and logicals.[4]

[1] See Wikipedia: Data type.

[2] The R language specification has a list of possible types.

[3] R will automatically assume that a number is supposed to be a double. To make an integer, you have to explicitly specify it as an integer literal by appending `L` to the end, like with `typeof(3L)` (which has output `"integer"`). The `L` stands for for a "long" integer, which is common programming terminology for a 32-bit integer data type.

[4] The `mode()` function will give you the *mode* of an object as described in The New S Language. Integers and doubles both have the "numeric" mode. If you're wondering how S is relevant to R, it's because R is one of the modern *implementations* of the S language specification. It's open-source and is free software, whereas S-PLUS, the only other modern implementation of S, is proprietary.

Exercise. We can test if something is numeric with the `is.numeric()` function. Verify that `3`, `3.1`, and `5L` are all numeric.

All strings have type "character" and are defined by wrapping text in between single or double quotes (there's no difference). For example, we have `typeof("qwerty") = typeof('qwerty') = "character"`.

Finally, the boolean values `TRUE` and `FALSE` have type "logical."

Exercise. Try running the following commands: `TRUE`, `FALSE`, `T`, `F`. Think about the following commands and predict their outcome: `TRUE == FALSE`, `TRUE & (T | FALSE)`. Next, run them and check your predictions. Finally, explain the output of `F = T; F & TRUE`.[5]

Remark. In your code, *always* use `TRUE` and `FALSE` instead of `T` and `F` to avoid potential confusion.

Atomic vectors

Atomic vectors in R are broadly analogous to lists in Python, with the exception that *they can only contain a single type*. Vectors are formed with the `c()` function, which stands for *combine*. For example:

```
> c(1,2)
[1] 1 2
> c(1.1, 4.5, 2.7)
[1] 1.1 4.5 2.7
> c('test', 'test2')
[1] "test"  "test2"
```

You should not think of vectors as being some sort of *enclosing structure* with values within it. Rather, the vector structure is intrinsically associated with every value, such that a single value (say, `4` or `"test"`) are themselves vectors of length 1.

Remark. You can make an empty vector with just `c()`.

Exercise. Is there any distinction between an atomic vector and a single value of the same type? There's a function you can use to check whether or not something is an atomic vector: look back to previous exercises and try to figure out what it is.[6]

Exercise. Determine how vectors behave when you nest them inside each other.

[5] You'll notice that `T` and `F` aren't equivalent to the primitive boolean values `TRUE` and `FALSE` but rather system-predefined variables initialized to `T = TRUE` and `F = FALSE`. That's why you can set `T = F` to set `T` to be equal to `FALSE`, and even change them to non-logical values altogether such as with `T = 2^4 + 1`.

[6] Similar to `is.numeric()`, we can use `is.atomic()` to check if something is an atomic vector. You'll find that single values also count as being atomic vectors.

Exercise. If you have an atomic vector `v = c(1, 2, 3)`, how can you append the value `4` to the end, forming `c(1, 2, 3, 4)`?

Recall that vectors can only be associated with a single type. What happens when you try to add an incompatible type to a vector?

Exercise. Before doing anything, consider the three primary types which we'll be handling: numeric, logical, and character. Which is the most specific and which is the most general? Based on that, what do you expect will happen when we add an incompatible type to a vector?

Exercise. Look at the output of `c(5, FALSE, TRUE, 10)`, `c('asdf', FALSE)`, and `c(TRUE, 'qwerty', 10.10)`. What is the underlying principle behind vector type coercion?[7]

Of course, you can also explicitly coerce vectors into different types with, say, `as.character()` and `as.numeric()`. (Try this out!) But in general, coercing a more general type to a more specific type will result in an error.[8] In particular, you will introduce `NA` values into a vector when type coercion is not possible:

```
> as.logical(c('TRUE', 'str1', 'str2'))
[1] TRUE   NA   NA
```

Remark. The `NA` is a special value that automatically takes on the type of the enclosing vector as needed. By default, `NA` is of type "logical", but you can access other `NA` types with `NA_real_` and `NA_character_`.

Tips and tricks

Since `TRUE` and `FALSE` can be automatically coerced to `1` and `0` respectively, you can use the `sum()` and `mean()` functions (among others) on logical vectors to easily calculate the number and proportion of `TRUE` entries.

The length of a vector can be determined with the `length()` function.

Supplemental exercises

Advanced R, 2.1.3.3. Test your knowledge of vector coercion rules by predicting the output of the following uses of `c()`:

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

[7] Vectors are automatically converted to the most general type necessary to accommodate all of the information inside. In order of increasing generality, we have the types logical, numeric, and character (with integer < double within the numeric category).

[8] In specific cases, conversion from more to less general works, *e.g.*, `as.logical('TRUE')` or `as.numeric('100')`.

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

Advanced R, 2.1.3.5. Why is `1 == "1"` true? Why is `-1 < FALSE` true?

Advanced R, 2.1.3.6. Why is the default missing value, `NA`, a logical vector? What's special about logical vectors? (Hint: think about `c(FALSE, NA_character_)`.)