Basic Data Types and Syntax in R

Jamaal Green

2018-07-27

The Basics

This sheet is to provide some minimal introductory knowledge for brand new beginners and to get us acquainted with the idea of *tidy* data from Hadley Wickham. We will try not to get bogged down into too much unnecessary detail here but we want to introduce (briefly) some of the basic data types in R, assignment and calling functions.

We can use R as a simple calculator by jsut type expressions into our consoles:

```
1 + 200/3
```

```
## [1] 67.66667
```

```
(12 + 35 + 98)/4 * 5
```

```
## [1] 181.25
```

We create new objects using the `<-` operator.

```
my_first_object <- 7 *4

my_first_object
```

```
## [1] 28
```

The basic structure for making an object is `object_name <- value`. For those of you coming from other languages you may wonder why R does not use `=` for assignment. Well, you can, but it is generally considered better practice to stay with the `<-` especially when you start designing your own functions.

Finally, in R we are constantly calling **functions**, pre-defined bits of code that allow us to perform all kinds of operations. The general structure for function calls is thus:

```
function_name(arg1 = value1, arg2 = value2...)
```

Data Types in R

The basic data type in R is the vector. Of vectors there are two types:

1. **Atomic vectors**- these vectors can be of six types logical, integer, double, character, complex and raw. Integer and double vectors are commonly referred as *numeric* vectors.

2. **Lists**- also referred to as recursive vectors because they can hold other lists

The main difference between atomic vectors and lists, aside from the recursive nature of lists, is that vectors are homogenous while lists are heterogeneous. This means that numeric vectors hold only numeric values. If any other value is introduced, like a character, then the entire vector will be converted to a character vector, because that is the most flexible data type out of all the values.

```
# we use the c() command to start a vector

vec1 <- c(4, 89, 91837, 19)

class(vec1)
```

```
## [1] "numeric"
```

```
#note the addition of a character

vec2 <- c(4, 89, 91837, 19, "a")

class(vec2)
```

```
## [1] "character"
```

```
#we use list() to start a list

list_1 <- list(4, 89, 91837, 19)

class(list_1)
```

```
## [1] "list"
```

```
list_2 <- list(4, 89, 91837, 19, "a")

class(list_2)
```

```
## [1] "list"
```

Note how an object of class list remains a list regardless of the objects in it. While we will not focus deeply on this question, do note that you can navigate vectors using indexing like this:

```
#what is the third element of vec_1?

vec1[3]
```

```
## [1] 91837
```

```
#what is the first element of vec_2?

vec2[1]
```

```
## [1] "4"
```

Again, we won't dwell on this but it is important to understand the differences between vectors and lists because the workhorse of our data analysis workflows, the dataframe, is a special form of list that combines multiple vectors of the same *length* into what we commonly recognize as a two dimensional table.

The Dataframe, Tibble and Tidy Data

The dataframe will be the data object you work with throughout most of your R workflows. You can think of it, if it helps, as a rough equivalent of a well formatted excel worksheet. A dataframe has **columns** and **rows**. Dataframe columns are of one of the types of vectors (just look up the page if you can't remember) and, because a dataframe is a special kind of list, dataframe columns can also hold *lists.* This kind of abstraction can make programming very powerful, but it can be a difficult concept to wrap your head around at first so this is all we will say on it.

Tidy Data

"Tidy" data is a concept developed by Hadley Wickham to describe a particular way of organizing data in a table format. Again, the basic idea is that variables are held in columns, observations in rows, and each cell accounts for a singular value. For those of you familiar with databases, this is essentially what we get when discussing "long data".

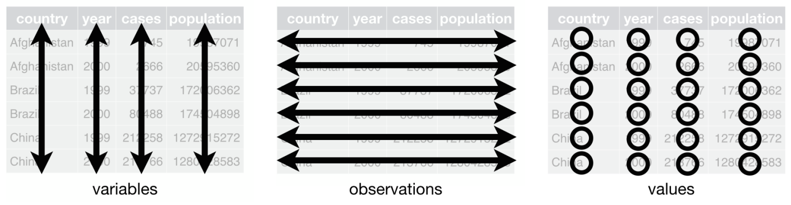

Figure 1: Tidy Data, courtesy of Wickham and Grolemund