On Loops

Sander Renes

03-11-2021

Functions needed
bind_rows()
dim()
filter()
for
geom_point
gglot()
group_by
list.files()
lm()
max()
min()
mutate()
near()
paste0()
print
read_csv()
readRDS
rename()
rnorm()
saveRDS()
select()
seq_along()
setwd()
stargazer()
stargazer()
str()
summarise()
tibble()
while

Loops

Repeating code by copy-paste is a waste of time, and is asking for trouble. So we don't, we use loops and functions to reduce the amount of times we need repetition.

Before you write a loop always make sure you understand your basics
1 what is the final goal
2 what is the task I have to repeat to get there
3a what am I repeating the task over?
3b do I have a list, description, or stopping condition?

Iterations and Indices.

Rember the basic vector-types in R:

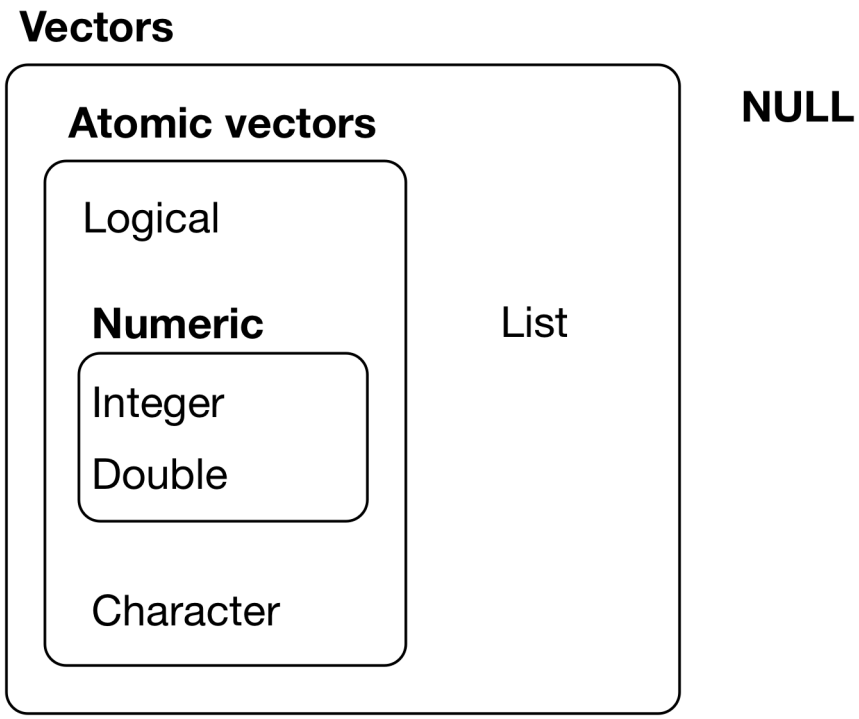

Figure 1: Lists and vector types

Every vector or list contains a bunch of values. The left-hand vectors all have items with the same underlying type of data, items in a list can be of any type, including vectors or lists. There are several ways to access the information in a vector, the easiest one is through the indices and square brackets [] as in the example below:

```
x <- c(1, 3, 2, 5.2, -4, 5, 12)
str(x)
```

```
##  num [1:7] 1 3 2 5.2 -4 5 12
```

```
str(x[3])
```

```
##  num 2
```

```
str(x[-4])
```

```
##  num [1:6] 1 3 2 -4 5 12
```

Indices start at 1 and simply add up. Using the minus-sign, deselects the item, as you can see, we took the fourth element out.

Lists are more complicated, since an item in a list can be anything, furhtermore you have to be carefull about whether you want the actual item in the list, or a list containing the output:

```
x <- c(1, 2, 3, 4)
y <- list(str_split(letters[1:6], pattern=""))
z <- list(x,y)
```

get a list of lists

```
str(z)
```

```
## List of 2
##  $ : num [1:4] 1 2 3 4
##  $ :List of 1
##   ..$ :List of 6
##   .. ..$ : chr "a"
##   .. ..$ : chr "b"
##   .. ..$ : chr "c"
##   .. ..$ : chr "d"
##   .. ..$ : chr "e"
##   .. ..$ : chr "f"
```

get a single list of numbers

```
str(z[1])
```

```
## List of 1
##  $ : num [1:4] 1 2 3 4
```

get a single number or letter

```
str(z[[1]][[1]])
```

```
##  num 1
```

```
str(z[[2]][[1]][[4]])
```

```
##  chr "d"
```

Note that in the last examples, we take the lists of lists (Z), ask for the second lists (y) by the first set of brackets [[2]], but this is also a lists (the list containing element (y)), so we select the first (and only) element of this list, and then finaly select an element from y with the last set of brackets [[4]]. In this sort of complicated entities, lists of list and tibles and dataframes, it is important to check whether you get your indices right! So after writing code that does something like this, whether it is subsetting, selecting entries, or working with elements of lists or factors, select some elements and see if they behave as expected. Do you get the output you expect? Is it in the stucture you expect? With tibles and dataframes, there are a few more options to select elements or subsets as we will see in the later exercises.

debugging

When programming, things can and will go wrong. Solving issues and making sure that your output is as you expected, is probably the core task of programming. Any good idea will go bad, if your code does something you do not expect it to do. The trick is to go through your loop line-by-line. Set the counter variable for some useful value, go to the first line of code, check its output. Is it what you expect? Then go to the second line, is the output what you expect? etc.