Introduction to R Programming

Lists

Pedro Fonseca

28 Abril 2020

Before we start

The `str()` function displays the internal structure of an object:

```
vec <- c(1, 5, 0.1)
str(vec)
```

```
##  num [1:3] 1 5 0.1
```

```
vec_f <- factor(c("good", "bad"))
str(vec_f)
```

```
##  Factor w/ 2 levels "bad","good": 2 1
```

```
mat_char <- matrix(c("a", "u", "mk", "q!"), ncol = 2)
str(mat_char)
```

```
##  chr [1:2, 1:2] "a" "u" "mk" "q!"
```

Before we start

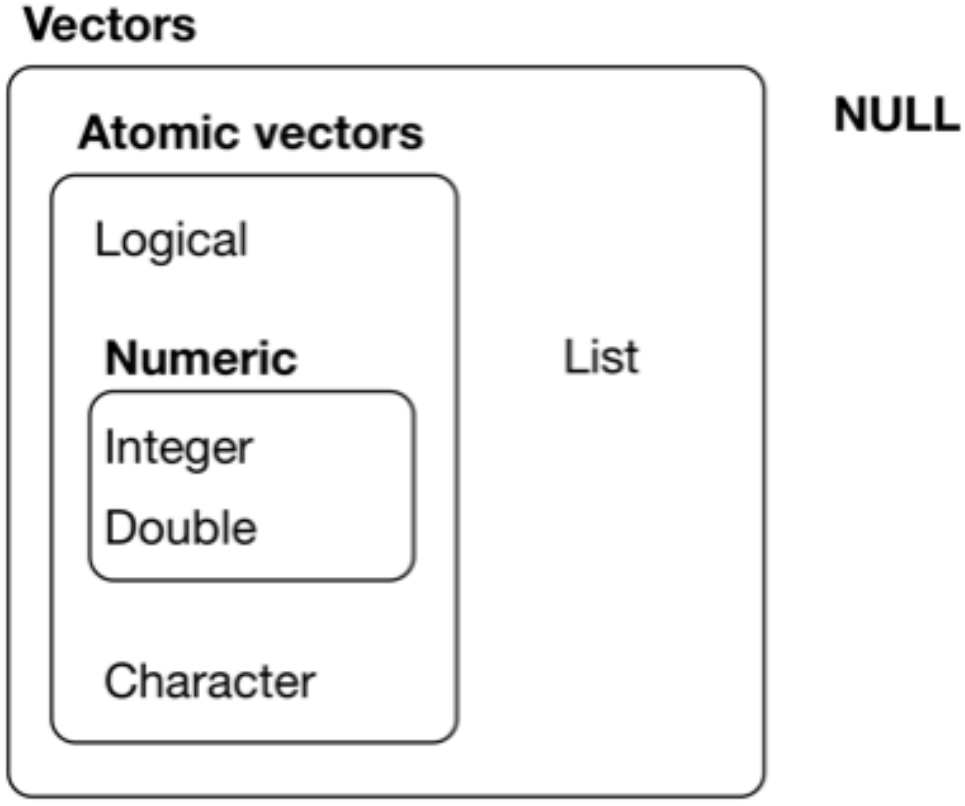

Figure 1: The hierarchy of R's vector types

Why do we need lists?

- Atomic vectors and matrices are convenient data storage structures but they have a limitation: all their elements have to be of the same data type.
- If we provide elements of more than one data type to an atomic vector or matrix, R will coerce to the most general data type.

Why do we need lists?

```
vec1 <- c(TRUE, FALSE)
typeof(vec1)
```

```
## [1] "logical"
```

```
vec2 <- c(TRUE, FALSE, 3)
typeof(vec2)
```

```
## [1] "double"
```

Logical values are coerced to numeric if combined with numeric values.

Why do we need lists?

```
vec3 <- c(TRUE, FALSE, 3, "4")
typeof(vec3)
```

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

Logical and numeric values are coerced to character if combined with character values.

What are lists?

- Lists are a step up in complexity from atomic vectors.
- Atomic vectors are homogeneous, while lists can be heterogeneous.
- Lists can store objects of any type (including lists).
- Lists are recursive vectors.

Creating a list

```r
my_list <- list(
  matrix(c(1, 5, -1, 4), ncol = 2),
  c("S", "M", "S", "W"),
  "I-am-a-string",
  c(TRUE, FALSE, FALSE)
)
```

```r
str(my_list)
```

```
## List of 4
##  $ : num [1:2, 1:2] 1 5 -1 4
##  $ : chr [1:4] "S" "M" "S" "W"
##  $ : chr "I-am-a-string"
##  $ : logi [1:3] TRUE FALSE FALSE
```

Whats inside the list?

```
my_list
```

```
## [[1]]
##      [,1] [,2]
## [1,]    1   -1
## [2,]    5    4
##
## [[2]]
## [1] "S" "M" "S" "W"
##
## [[3]]
## [1] "I-am-a-string"
##
## [[4]]
## [1]  TRUE FALSE FALSE
```

Lists inside lists

Lists can even contain other lists!

```
list_with_lists <- list(
  c(4:1),
  list(
    matrix(c("d", "a", "r", "r"), ncol = 2),
    factor("yes", "no", "yes")
    ),
  list(
    c(TRUE, FALSE, TRUE),
    1:4,
    c(1, 5, 7, 9, 0)
    )
  )
```

Lists inside lists

```
str(list_with_lists)
```

```
## List of 3
##  $ : int [1:4] 4 3 2 1
##  $ :List of 2
##   ..$ : chr [1:2, 1:2] "d" "a" "r" "r"
##   ..$ : Factor w/ 1 level "yes": NA
##  $ :List of 3
##   ..$ : logi [1:3] TRUE FALSE TRUE
##   ..$ : int [1:4] 1 2 3 4
##   ..$ : num [1:5] 1 5 7 9 0
```

Lists inside lists inside lists...

```
a_list_of_lists <- list(list(list(list(c(1, 5),
                                       c("a", "b")))))
str(a_list_of_lists)
```

```
## List of 1
##  $ :List of 1
##   ..$ :List of 1
##   .. ..$ :List of 2
##   .. .. ..$ : num [1:2] 1 5
##   .. .. ..$ : chr [1:2] "a" "b"
```

This is why lists are said to be recursive.

Naming list elements

```
names(my_list) <- c("Numbers", "Letters",
                    "a_lonely_string", "T_F")
str(my_list)
```

```
## List of 4
##  $ Numbers         : num [1:2, 1:2] 1 5 -1 4
##  $ Letters         : chr [1:4] "S" "M" "S" "W"
##  $ a_lonely_string: chr "I-am-a-string"
##  $ T_F             : logi [1:3] TRUE FALSE FALSE
```

Naming list elements

```
my_list
```

```
## $Numbers
##      [,1] [,2]
## [1,]    1   -1
## [2,]    5    4
##
## $Letters
## [1] "S" "M" "S" "W"
##
## $a_lonely_string
## [1] "I-am-a-string"
##
## $T_F
## [1]  TRUE FALSE FALSE
```

Naming list elements

We can also name the elements of the list when we create it:

```
my_list <- list(
  some_numbers = matrix(c(1, 5, -1, 4), ncol = 2),
  some_letters = c("S", "M", "S", "W"),
  a_lonely_string = "I-am-a-string",
  T_or_F = c(TRUE, FALSE, FALSE)
)

str(my_list)
```

```
## List of 4
##  $ some_numbers   : num [1:2, 1:2] 1 5 -1 4
##  $ some_letters   : chr [1:4] "S" "M" "S" "W"
##  $ a_lonely_string: chr "I-am-a-string"
##  $ T_or_F         : logi [1:3] TRUE FALSE FALSE
```

Naming list elements

`list()` does not keep the names of input objects:

```
num_vec <- 1:3
char_mat <- matrix(c("a", "b", "c", "m"), ncol = 2)
a_lonely_string <- "Hello!"
a_factor <- factor(c("Yes", "No", "Yes"))

my_list_2 <- list(
  num_vec,
  char_mat,
  a_lonely_string,
  a_factor,
  my_list
  )
```

Naming list elements

```
str(my_list_2)
```

```
## List of 5
##  $ : int [1:3] 1 2 3
##  $ : chr [1:2, 1:2] "a" "b" "c" "m"
##  $ : chr "Hello!"
##  $ : Factor w/ 2 levels "No","Yes": 2 1 2
##  $ :List of 4
##   ..$ some_numbers   : num [1:2, 1:2] 1 5 -1 4
##   ..$ some_letters   : chr [1:4] "S" "M" "S" "W"
##   ..$ a_lonely_string: chr "I-am-a-string"
##   ..$ T_or_F         : logi [1:3] TRUE FALSE FALSE
```

Naming list elements

```
num_vec <- 1:3
char_mat <- matrix(c("a", "b", "c", "m"), ncol = 2)
a_lonely_string <- "Hello!"
a_factor <- factor(c("Yes", "No", "Yes"))

my_list_2_named <- list(
  numbers = num_vec,
  c_mat = char_mat,
  lonely = a_lonely_string,
  fac = a_factor,
  old_list = my_list
  )
```

Naming list elements

```
str(my_list_2_named)
```

```
## List of 5
##  $ numbers : int [1:3] 1 2 3
##  $ c_mat   : chr [1:2, 1:2] "a" "b" "c" "m"
##  $ lonely  : chr "Hello!"
##  $ fac     : Factor w/ 2 levels "No","Yes": 2 1 2
##  $ old_list:List of 4
##   ..$ some_numbers   : num [1:2, 1:2] 1 5 -1 4
##   ..$ some_letters   : chr [1:4] "S" "M" "S" "W"
##   ..$ a_lonely_string: chr "I-am-a-string"
##   ..$ T_or_F         : logi [1:3] TRUE FALSE FALSE
```

Coercion

We can turn objects into a lists with `as.list()`:

```
char_vec <- c("yes", "no")
as.list(char_vec)
```

```
## [[1]]
## [1] "yes"
##
## [[2]]
## [1] "no"
```

```
str(as.list(char_vec))
```

```
## List of 2
##  $ : chr "yes"
##  $ : chr "no"
```

Coercion

```
num_matrix <- matrix(1:4, ncol = 2)
as.list(num_matrix)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
```

Subsetting

We can subset lists with:

- The `[` operator. Always returns a list.
- The `[[` operator. Returns the object that is inside the subsetted element of the list. Can return objects of any type.
- The `$` operator. Similar to `[[` but works only with named list elements.

Subsetting

```
str(my_list)
```

```
## List of 4
##  $ some_numbers   : num [1:2, 1:2] 1 5 -1 4
##  $ some_letters   : chr [1:4] "S" "M" "S" "W"
##  $ a_lonely_string: chr "I-am-a-string"
##  $ T_or_F         : logi [1:3] TRUE FALSE FALSE
```

```
my_list[1]
```

```
## $some_numbers
##      [,1] [,2]
## [1,]    1   -1
## [2,]    5    4
```

```
typeof(my_list[1])
```

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

Subsetting

```
is.matrix(my_list[1])
```

```
## [1] FALSE
```

```
my_list[[1]]
```

```
##      [,1] [,2]
## [1,]    1   -1
## [2,]    5    4
```

```
typeof(my_list[[1]])
```

```
## [1] "double"
```

```
is.matrix(my_list[[1]])
```

```
## [1] TRUE
```

Subseting

"If list x is a train carrying objects, then x[[5]] is the object in car 5; x[4:6] is a train of cars 4-6" — *@RLangTip*

Subsetting

If element names are available:

```
my_list["some_letters"]
```

```
## $some_letters
## [1] "S" "M" "S" "W"
```

```
my_list[["some_letters"]]
```

```
## [1] "S" "M" "S" "W"
```

```
my_list$some_letters
```

```
## [1] "S" "M" "S" "W"
```

Subsetting

Because it can return only a single value, you must use `[[` with either a single positive integer or a string:

```
a_short_list <- list(a = c(1, 2), b = 3)
a_short_list[[1]]
```

```
## [1] 1 2
```

```
a_short_list[["a"]]
```

```
## [1] 1 2
```

Subsetting

If you do supply a vector to `[[` it indexes recursively:

```
rec_l <- list(a = list(b = list(c = list(d = 3))))
str(rec_l)
```

```
## List of 1
##  $ a:List of 1
##   ..$ b:List of 1
##   .. ..$ c:List of 1
##   .. .. ..$ d: num 3
```

```
rec_l[["a"]][["b"]][["c"]][["d"]]
```

```
## [1] 3
```

```
rec_l[[c("a", "b", "c", "d")]] # Same as above!
```

```
## [1] 3
```

Subsetting

Some examples using `my_list`:

```
my_list
```

```
## $some_numbers
##      [,1] [,2]
## [1,]    1   -1
## [2,]    5    4
##
## $some_letters
## [1] "S" "M" "S" "W"
##
## $a_lonely_string
## [1] "I-am-a-string"
##
## $T_or_F
## [1]  TRUE FALSE FALSE
```

Subsetting

```
my_list[c(2, 3)]
```

```
## $some_letters
## [1] "S" "M" "S" "W"
##
## $a_lonely_string
## [1] "I-am-a-string"
```

```
typeof(my_list[c(2, 3)])
```

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

```
length(my_list[c(2, 3)])
```

```
## [1] 2
```

Subsetting

Extracting an elements from a vector that is inside a list:

```
my_list[[2]][1]
```

```
## [1] "S"
```

or:

```
my_list[[c(2, 1)]]
```

```
## [1] "S"
```

Extracting elements of a vector that is inside a list:

```
my_list[[2]][c(2, 3)]
```

```
## [1] "M" "S"
```

Subsetting

```
my_list[[1]]
```

```
##      [,1] [,2]
## [1,]    1   -1
## [2,]    5    4
```

Extracting elements of a matrix that is inside a list:

```
my_list[[1]][, 2] # Extracts the second column
my_list[[1]][1, ] # Extracts the first row
my_list[[1]][-1, ] # Matrix without the first row
my_list[[1]][1, 1] # Element in position (1,1)
```

Add a new element to a list using [[

```
length(my_list)
```

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

```
my_list[[5]] <- matrix(c(0.21, 0.45, 0.6, 3),
                       ncol = 2)

length(my_list)
```

```
## [1] 5
```

Note that this new element is not named. Using [instead of [[would also work.

Add a new element to a list using $

Adding elements to a list with $ automatically names the new element.

```
my_list$new_thing <- list(c(1, 5), "some-stuff",
                          matrix(1:6, nrow = 2))
str(my_list)
```

```
## List of 6
##  $ some_numbers    : num [1:2, 1:2] 1 5 -1 4
##  $ some_letters    : chr [1:4] "S" "M" "S" "W"
##  $ a_lonely_string: chr "I-am-a-string"
##  $ T_or_F          : logi [1:3] TRUE FALSE FALSE
##  $                 : num [1:2, 1:2] 0.21 0.45 0.6 3
##  $ new_thing       :List of 3
##   ..$ : num [1:2] 1 5
##   ..$ : chr "some-stuff"
##   ..$ : int [1:2, 1:3] 1 2 3 4 5 6
```

Extracting elements from a list inside a list

Element 6 of `my_list` is also list:

```
str(my_list[6])
```

```
## List of 1
##  $ new_thing:List of 3
##   ..$ : num [1:2] 1 5
##   ..$ : chr "some-stuff"
##   ..$ : int [1:2, 1:3] 1 2 3 4 5 6
```

Extracting elements from a list inside a list

Extract the element in the position (2, 3) of the matrix that is inside the list that is itself inside `my_list`:

```
my_list[[6]][[3]][2, 3]
```

```
## [1] 6
```

Since the list that inside `my_list_2` is named, this also works:

```
my_list[["new_thing"]][[3]][2, 3]
```

```
## [1] 6
```

Delete a element of a list

```
str(my_list)
```

```
## List of 6
##  $ some_numbers    : num [1:2, 1:2] 1 5 -1 4
##  $ some_letters    : chr [1:4] "S" "M" "S" "W"
##  $ a_lonely_string: chr "I-am-a-string"
##  $ T_or_F          : logi [1:3] TRUE FALSE FALSE
##  $                 : num [1:2, 1:2] 0.21 0.45 0.6 3
##  $ new_thing       :List of 3
##   ..$ : num [1:2] 1 5
##   ..$ : chr "some-stuff"
##   ..$ : int [1:2, 1:3] 1 2 3 4 5 6
```

Delete an element of a list

NULL is often used to represent the absence of a vector (as opposed to NA which is used to represent the absence of a value in a vector).

```
my_list$new_thing <- NULL
# my_list["new_thing"] <- NULL does the same

str(my_list)
```

```
## List of 5
##  $ some_numbers   : num [1:2, 1:2] 1 5 -1 4
##  $ some_letters   : chr [1:4] "S" "M" "S" "W"
##  $ a_lonely_string: chr "I-am-a-string"
##  $ T_or_F         : logi [1:3] TRUE FALSE FALSE
##  $                : num [1:2, 1:2] 0.21 0.45 0.6 3
```

Delete an element of a list

```
my_list[[5]] <- NULL
str(my_list)
```

```
## List of 4
##  $ some_numbers   : num [1:2, 1:2] 1 5 -1 4
##  $ some_letters   : chr [1:4] "S" "M" "S" "W"
##  $ a_lonely_string: chr "I-am-a-string"
##  $ T_or_F         : logi [1:3] TRUE FALSE FALSE
```

Delete an element of a list

```
my_list["a_lonely_string"] <- NULL
# my_list[2] <- NULL does the same

str(my_list)
```

```
## List of 3
##  $ some_numbers: num [1:2, 1:2] 1 5 -1 4
##  $ some_letters: chr [1:4] "S" "M" "S" "W"
##  $ T_or_F      : logi [1:3] TRUE FALSE FALSE
```

Merge lists

You can merge lists with c():

```
list1 <- list(1, 2)
list2 <- list(c("Sun","Mon"), c(1, 2))
list3 <- list("Hi!")

merged.list <- c(list1, list2, list3)
str(merged.list)
```

```
## List of 5
##  $ : num 1
##  $ : num 2
##  $ : chr [1:2] "Sun" "Mon"
##  $ : num [1:2] 1 2
##  $ : chr "Hi!"
```

Unlist

```r
l <- list(
  x = 1:2,
  y = c("ab", "3"),
  m = matrix(1:4, ncol = 2)
)

unlist(l)
```

```
##   x1   x2   y1   y2   m1   m2   m3   m4
##  "1"  "2" "ab"  "3"  "1"  "2"  "3"  "4"
```

The `unlist` function:

- Keeps names (if available)
- Collapses a list to a vector
- Coerces to the most general data type