Exercises-1: Introduction to R- types

Nathan Green, Imperial College London

07/09/2019

R provides many functions to examine features of vectors and other objects, for example

- `class()` - what kind of object is it (high-level)
- `typeof()` - what is the object's data type (low-level)
- `length()` - how long is it? What about two dimensional objects
- `attributes()` - does it have any metadata

1. Type `x <- "dataset"` and `y <- 1:10`. What are the type and attributes of `x` and `y`?

```
x <- "dataset"
y <- 1:10

typeof(x)
```

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

```
attributes(x)
```

```
## NULL
```

```
y
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
typeof(y)
```

```
## [1] "integer"
```

```
length(y)
```

```
## [1] 10
```

```
z <- as.numeric(y)
z
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
typeof(z)
```

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

R has many data structures. These include

- atomic vector
- list
- matrix
- data frame
- factors
- Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types:

- atomic vectors
- lists

although the term "vector" most commonly refers to the atomic types not to lists.

The Different Vector Modes

A vector is a collection of elements that are most commonly of mode character, logical, integer or numeric.

You can create an empty vector with `vector()`. (By default the mode is logical. You can be more explicit as shown in the examples below.) It is more common to use direct constructors such as `character()`, `numeric()`, etc.

2. Create a character vector of length 5.

```
vector("character", length = 5)
```

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

```
#a vector of mode 'character' with 5 elements
```

3. Create character, numeric, logical vectors directly. What do you see?

```
character(5) # the same thing, but using the constructor directly
```

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

```
numeric(5)   # a numeric vector with 5 elements
```

```
## [1] 0 0 0 0 0
```

```
logical(5)   # a logical vector with 5 elements
```

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

You can also create vectors by directly specifying their content. R will then guess the appropriate mode of storage for the vector. For instance:

`x <- c(1, 2, 3)` will create a vector x of mode numeric. These are the most common kind, and are treated as double precision real numbers. If you wanted to explicitly create integers, you need to add an L to each element (or coerce to the integer type using as.integer()).

4. Create a vector of 3 integers.

```
x1 <- c(1L, 2L, 3L)
```

Using TRUE and FALSE will create a vector of mode logical:

`y <- c(TRUE, TRUE, FALSE, FALSE)` While using quoted text will create a vector of mode character:

`z <- c("Sarah", "Tracy", "Jon")`

Examining Vectors

The functions `str()` provide useful information about your vectors and R objects in general.

5. View the structure of `z` above.

Adding Elements

The function `c()` (for combine) can also be used to add elements to a vector.

6. Attach `Annette` to the end of `z`. Attach `Greg` to the start.

```
z <- c("Sarah", "Tracy", "Jon")
z <- c(z, "Annette")
z
```

```
## [1] "Sarah"   "Tracy"   "Jon"     "Annette"
```

```
z <- c("Greg", z)
z
```

```
## [1] "Greg"    "Sarah"   "Tracy"   "Jon"     "Annette"
```

Vectors from a Sequence of Numbers

You can create vectors as a sequence of numbers using `:`.

7. Create a series from 1 to 10 using `:`. Create the same output using `seq()`. Change the step size to 0.1.

```
series <- 1:10
seq(10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
seq(from = 1, to = 10, by = 0.1)
```

```
##  [1]  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3
## [15]  2.4  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7
## [29]  3.8  3.9  4.0  4.1  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.0  5.1
## [43]  5.2  5.3  5.4  5.5  5.6  5.7  5.8  5.9  6.0  6.1  6.2  6.3  6.4  6.5
## [57]  6.6  6.7  6.8  6.9  7.0  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9
## [71]  8.0  8.1  8.2  8.3  8.4  8.5  8.6  8.7  8.8  8.9  9.0  9.1  9.2  9.3
## [85]  9.4  9.5  9.6  9.7  9.8  9.9 10.0
```

8. Create the sequence above in reverse from 10 to 1.

```
seq(from = 10, to = 1, by = -0.1)
```

```
##  [1] 10.0  9.9  9.8  9.7  9.6  9.5  9.4  9.3  9.2  9.1  9.0  8.9  8.8  8.7
## [15]  8.6  8.5  8.4  8.3  8.2  8.1  8.0  7.9  7.8  7.7  7.6  7.5  7.4  7.3
## [29]  7.2  7.1  7.0  6.9  6.8  6.7  6.6  6.5  6.4  6.3  6.2  6.1  6.0  5.9
## [43]  5.8  5.7  5.6  5.5  5.4  5.3  5.2  5.1  5.0  4.9  4.8  4.7  4.6  4.5
## [57]  4.4  4.3  4.2  4.1  4.0  3.9  3.8  3.7  3.6  3.5  3.4  3.3  3.2  3.1
## [71]  3.0  2.9  2.8  2.7  2.6  2.5  2.4  2.3  2.2  2.1  2.0  1.9  1.8  1.7
## [85]  1.6  1.5  1.4  1.3  1.2  1.1  1.0
```