R notes – Day 1

Hieu Phay

2022-05-10

1 R objects

R is a **functional programming language**.

Which means that it operates around the **object-function** metaphor:

- Everything that exists in an R environment is an **object**
- Everything that happens inside an R environment is a **function**

Today we delve into *objects*

1.1 Asssignment

Assignment is done by the sign (`<-`) symbol.

(`=` works too, but the convention is that we don't use it for assignments)

```
x <- 2
a <- "Thanh"
y <- TRUE
```

Object names must only contain letters (uppercase and lowercase), underscore `_`. Digits is can also be used except at the beginning of the object name (e.g. `x_1` is fine, but `1_x` is not accepted)

1.2 Listing and removing objects

Listing objects:

```
ls()
```

```
## [1] "a" "x" "y"
```

Remove objects:

```
skldfjsdklj <- 1
rm(skldfjsdklj)
```

1.3 How to inspect things

You can just type it out!

```
y
```

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

... or use `str()` (for complicated objects – more on that later).

2 Data types

Question: Why are objects different? (Thanh)

Some basic data type:

- Boolean: `TRUE` or `FALSE`

- Integer: e.g. `1`, `2`,...

- Float: e.g. `1.0` (is annoying to compares)

- Strings: e.g. `"I am a string!!"`

- ...

Be careful with comparisons using floats!! (floating-point errors)

```
0.1 + 0.2
```

```
## [1] 0.3
```

```
(0.1 + 0.2) == 0.3
```

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

3 Working with vectors

The **atomic** object in R is a `vector` (instead of scalars)

Vector: A list of value in the same **type**.

3.1 Create a vectors

Vectors are created using `c()`

Sequences is created with `:`

```
1:40
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

3.2 Concaternate vectors

`c()` is also concaternate vectors.

```
c(TRUE, 5, 13.5, "Hello")
```

```
## [1] "TRUE"  "5"     "13.5"  "Hello"
```

With different types: Data are coerced into the same type. - Boolean -> Interger -> Float -> String

Almost all functions works with vectors out of the box.

```
c(TRUE, FALSE) + c(3, 4)
```

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

If vector lengths are uneven

```
1:10 + c(TRUE, FALSE)
```

```
##  [1]  2  2  4  4  6  6  8  8 10 10
```

```r
1:10 + c(TRUE, FALSE, FALSE)
```

```
## Warning in 1:10 + c(TRUE, FALSE, FALSE): longer object length is not a multiple
## of shorter object length
```

```
##  [1]  2  2  3  5  5  6  8  8  9 11
```

3.3 Vector indexing

- Index by position number

```r
letters[c(1:4, 6:length(letters))]
```

```
##  [1] "a" "b" "c" "d" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
## [20] "u" "v" "w" "x" "y" "z"
```

- Index by a logic (boolean) vector

```r
letters[letters == "f"]
```

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

4 Puzzle

Puzzle source: https://adventofcode.com/2021/day/1

```r
input <- scan("~/Desktop/R-notes/day-1_puzzle")
```

4.1 Part 1: Count the number of times a depth measurement increases from the previous measurement.

Hint: Create a logic vector and use `sum()`

```r
sum(c(TRUE, FALSE, TRUE))
```

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

Solution

Here's the long answer – using only simple vector solution

```r
# Create a lag vector and a lead vector
input_lag <- input[2:length(input)]
input_lead <- input[1:(length(input) - 1)]

# Compare the lag vector with the lead vector
is_increase <- input_lag > input_lead

# Count the number of changes
n_increase <- sum(is_increase)

# Print the result
print(n_increase)
```

```
## [1] 1521
```

In R there are convenient functions for many things. In this case, we can use the function `diff()`, which compute the absolute differences of our vector

```r
changes <- diff(input)
is_increase <- changes > 0
n_increase <- sum(is_increase)

print(n_increase)
```

```
## [1] 1521
```

4.2 Part 2: Count the number of times the 3-period moving sum of depth measurement increases from the previous measurement

Solution

The second part of the puzzle is very similar – we only have to compute the 3-period moving sum of our input vector.

```r
# Compute the 3-period moving sum
moving_sum3 <- input[1:(length(input) - 2)] +
  input[2:(length(input) - 1)] + input[3:length(input)]


# The following part is similar to Part 1 of the puzzle
changes <- diff(moving_sum3)
is_increase <- changes > 0
n_increase <- sum(is_increase)

print(n_increase)
```

```
## [1] 1543
```