intro_to_r

Nicholus Tint Zaw

2022-08-14

What we are going to cover in this exercise?

1. Basic Set-up of R/R-studio and it IDE features
2. Vectors
3. Data Type

Vectors

length of vector with example

```r
# numeric vector of length 6
my_numbers <- c(1, 2, 3, 4, 5, 6)

# character vector of length 3
my_characters <- c("public", "policy", "101")

# vectors of length 1
tis_a_vector <- 1919
technically_a_logical_vector <- TRUE
```

how to create vectors

```r
# use `c()` function to combine vectors
c(c(1, 2, 3), c(4, 5, 6))
```

```
## [1] 1 2 3 4 5 6
```

```r
c(tis_a_vector, 1920)
```

```
## [1] 1919 1920
```

```r
# create vectors
c("a", "a", "a", "a")
```

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

```r
rep("a", 4)
```

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

```r
rep(c("a", 5), 10)
```

```
##  [1] "a" "5" "a" "5" "a" "5" "a" "5" "a" "5" "a" "5" "a" "5" "a" "5" "a" "5" "a"
## [20] "5"
```

```r
rep(c("a", 5), each = 10)
```

```
##  [1] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "5" "5" "5" "5" "5" "5" "5" "5" "5"
## [20] "5"
```

```r
# other ways to create vectors
c(2, 3, 4, 5)
```

```
## [1] 2 3 4 5
```

```r
2:5
```

```
## [1] 2 3 4 5
```

```r
seq(2, 5)
```

```
## [1] 2 3 4 5
```

```r
# creating random vectors
(my_random_normals <- rnorm(5))
```

```
## [1]  0.1977082 -1.1213204 -0.6004358  0.7225797  1.1875859
```

```r
(my_random_uniforms <- runif(5))
```

```
## [1] 0.4626758 0.4701143 0.8619882 0.8909810 0.6074989
```

```r
set.seed(132413523)
(my_random_normals <- rnorm(5))
```

```
## [1]  1.0310895 -0.2333208  0.8737198  0.8594548  0.2225077
```

```r
(my_random_uniforms <- runif(5))
```

```
## [1] 0.6203890 0.4066186 0.4997983 0.2896616 0.2411608
```

```r
## Creating empty vectors of a given type
my_integers <- integer(1e6) # 1 million 0

my_chrs <- character(4e5) # 400000
# check here for scientific notification explination
# https://calculator.name/scientific-notation-to-decimal/4e-5
my_chrs[1:10]
```

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

Vectors operation

```r
## Binary operators are vectorized
my_numbers <- 1:6

# this adds the vectors item by item
my_numbers + my_numbers
```

```
## [1]  2  4  6  8 10 12
```

```r
# this adds 6 to each object  (called recycling)
my_numbers + 6
```

```
## [1]  7  8  9 10 11 12
```

```r
# this compares the values in two vectors item by item
my_numbers > c(1, 1, 3, 3, pi, pi)
```

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

```r
# check this - interprets the result!
my_numbers > c(1, 1, 3, 3)
```

```
## Warning in my_numbers > c(1, 1, 3, 3): longer object length is not a multiple of
## shorter object length
```

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

```r
# this compares each value of the my_numbers vector with 4
my_numbers > 4 # behind the scenes 4 is recycled
```

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

```r
my_numbers == 3
```

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

```r
## Warning: Vector recycling
a <- 1:6 + 1:5 # different length
```

```
## Warning in 1:6 + 1:5: longer object length is not a multiple of shorter object
## length
```

```r
a # you can see the recycling
```

```
## [1]  2  4  6  8 10  7
```

```r
# another example: step-by-step mechanism of vector recycling
a <- c(1, 2, 3, 4, 5, 6) + c(1, 2, 3, 4, 5)
```

```
## Warning in c(1, 2, 3, 4, 5, 6) + c(1, 2, 3, 4, 5): longer object length is not a
## multiple of shorter object length
```

```r
# 1 + 1,
# 2 + 2,
# 3 + 3,
# 4 + 4,
# 5 + 5,
# !!!6 + 1!!! Recycled.
a
```

```
## [1]  2  4  6  8 10  7
```

Vectors with function operation

```r
# Vectorized functions built into R
a_vector <- rnorm(100)
sqrt(a_vector) # take the square root of each number
```

```
## Warning in sqrt(a_vector): NaNs produced
```

```
##   [1]        NaN 1.03364302        NaN 1.06419740 1.07046432 1.09150534
##   [7] 0.62123543 1.28367757 1.01684639        NaN        NaN        NaN
##  [13] 1.03167659        NaN 0.50932441        NaN        NaN        NaN
##  [19] 0.33052114        NaN 0.91740861 0.88825618 0.90567404        NaN
##  [25] 1.14838028 0.82172889        NaN 1.12041004 1.14132700        NaN
##  [31] 0.56595459 0.34644496        NaN        NaN        NaN        NaN
##  [37]        NaN 0.52693428 0.65956033        NaN        NaN 1.30828065
##  [43] 1.11855862        NaN 0.95075126        NaN        NaN        NaN
##  [49]        NaN 0.80783054        NaN        NaN 1.27237749 0.97418225
##  [55]        NaN 0.32349646 0.30003538        NaN 1.28595457 0.58116480
##  [61] 1.02452131 0.19260634        NaN        NaN 0.34714205 0.83157069
##  [67]        NaN 1.16803968 1.12071567 1.16463146        NaN 1.17488768
##  [73] 0.50899150        NaN        NaN        NaN 1.24491393        NaN
##  [79] 0.66468068 0.74836687 1.15896144        NaN        NaN        NaN
##  [85]        NaN        NaN 0.55687265        NaN        NaN 1.43671348
##  [91]        NaN 0.44029084        NaN        NaN        NaN        NaN
##  [97] 0.09766095        NaN 0.68517733 0.48876142
```

```r
log(a_vector) # take the natural log of each number
```

```
## Warning in log(a_vector): NaNs produced
```

```
##   [1]         NaN  0.06617895         NaN  0.12444180  0.13618499  0.17511558
##   [7] -0.95209032  0.49945813  0.03341212         NaN         NaN         NaN
##  [13]  0.06237047         NaN -1.34934025         NaN         NaN         NaN
##  [19] -2.21416930         NaN -0.17240461 -0.23699018 -0.19815164         NaN
##  [25]  0.27670499 -0.39268952         NaN  0.22738944  0.26438323         NaN
##  [31] -1.13848286 -2.12006265         NaN         NaN         NaN         NaN
##  [37]         NaN -1.28135889 -0.83236365         NaN         NaN  0.53742759
##  [43]  0.22408183         NaN -0.10100562         NaN         NaN         NaN
##  [49]         NaN -0.42680594         NaN         NaN  0.48177438 -0.05231375
##  [55]         NaN -2.25713422 -2.40770972         NaN  0.50300260 -1.08544184
##  [61]  0.04845098 -3.29421369         NaN         NaN -2.11604243 -0.36887793
##  [67]         NaN  0.31065371  0.22793494  0.30480939         NaN  0.32234510
##  [73] -1.35064793         NaN         NaN         NaN  0.43813280         NaN
##  [79] -0.81689706 -0.57972391  0.29504860         NaN         NaN         NaN
##  [85]         NaN         NaN -1.17083739         NaN         NaN  0.72471641
##  [91]         NaN -1.64063956         NaN         NaN         NaN         NaN
##  [97] -4.65250699         NaN -0.75615520 -1.43176161
```

```r
exp(a_vector) # e to the power of each number
```

```
##   [1] 0.1414270 2.9107707 0.7799524 3.1034553 3.1452515 3.2916334 1.4709868
##   [8] 5.1956831 2.8122267 0.4839415 0.2641854 0.4026690 2.8989731 0.3131237
##  [15] 1.2961669 0.3883621 0.3507876 0.9152563 1.1154347 0.2928736 2.3201656
##  [22] 2.2011920 2.2710572 0.9139639 3.7388470 1.9645012 0.2277146 3.5089563
##  [29] 3.6789497 0.3466362 1.3775473 1.1275240 0.4477635 0.5822282 0.5513689
##  [36] 0.3246191 0.0662677 1.3200370 1.5449937 0.4211469 0.3068676 5.5378053
##  [43] 3.4944409 0.6499619 2.4692833 0.5935017 0.5804447 0.6817096 0.9762787
##  [50] 1.9205089 0.4069386 0.3911420 5.0477595 2.5832055 0.2991221 1.1103219
##  [57] 1.0941975 0.5050481 5.2261724 1.4017935 2.8566337 1.0377939 0.4974876
##  [64] 0.1984679 1.1280693 1.9967279 0.8915051 3.9130483 3.5113606 3.8820619
##  [71] 0.4960876 3.9763371 1.2957275 0.6720951 0.3194708 0.3495058 4.7105784
##  [78] 0.7874992 1.5555052 1.7507652 3.8312520 0.3081388 0.6180296 0.6314657
##  [85] 0.1652591 0.7460150 1.3635712 0.5997400 0.7897056 7.8785639 0.4173291
##  [92] 1.2139215 0.1722686 0.9774360 0.4339222 0.7774891 1.0095833 0.4420773
##  [99] 1.5991432 1.2698360
```

```r
round(a_vector, 2) # round each number
```

```
##   [1] -1.96  1.07 -0.25  1.13  1.15  1.19  0.39  1.65  1.03 -0.73 -1.33 -0.91
##  [13]  1.06 -1.16  0.26 -0.95 -1.05 -0.09  0.11 -1.23  0.84  0.79  0.82 -0.09
##  [25]  1.32  0.68 -1.48  1.26  1.30 -1.06  0.32  0.12 -0.80 -0.54 -0.60 -1.13
##  [37] -2.71  0.28  0.44 -0.86 -1.18  1.71  1.25 -0.43  0.90 -0.52 -0.54 -0.38
##  [49] -0.02  0.65 -0.90 -0.94  1.62  0.95 -1.21  0.10  0.09 -0.68  1.65  0.34
##  [61]  1.05  0.04 -0.70 -1.62  0.12  0.69 -0.11  1.36  1.26  1.36 -0.70  1.38
##  [73]  0.26 -0.40 -1.14 -1.05  1.55 -0.24  0.44  0.56  1.34 -1.18 -0.48 -0.46
##  [85] -1.80 -0.29  0.31 -0.51 -0.24  2.06 -0.87  0.19 -1.76 -0.02 -0.83 -0.25
##  [97]  0.01 -0.82  0.47  0.24
```

```r
# vector with string function
a_chr_vector <- c("a", "w", "e", "s", "o", "m", "e")
paste0(a_chr_vector, a_chr_vector) # combine strings
```

```
## [1] "aa" "ww" "ee" "ss" "oo" "mm" "ee"
```

```r
paste0(a_chr_vector, collapse = "") # combine strings
```

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

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
str_to_upper(a_chr_vector) # make each chr uppercase
```

```
## [1] "A" "W" "E" "S" "O" "M" "E"
```

```r
str_replace(a_chr_vector, "e", "3")
```

```
## [1] "a" "w" "3" "s" "o" "m" "3"
```

```r
# Functions that reduce vectors
sum(a_vector)  # add all the numbers
```

```
## [1] -0.0405124
```

```r
median(a_vector) # find the median
```

```
## [1] -0.006642382
```

```r
length(a_vector) # how long is the vector
```

```
## [1] 100
```

```
any(a_vector > 1) # TRUE if any number in a_vector > 1
```

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

Data Type

data frame

```
## Tibble columns are vectors
care_data <- tibble(
  id = 1:5,
  n_kids = c(2, 4, 1, 1, NA),
  child_care_costs = c(1000, 3000, 300, 300, 500),
  random_noise = rnorm(5, sd = 5)*30
)
View(care_data)

## Subsetting
# three ways to pull out a column as a tibble
care_data %>% pull(n_kids) # tidy way
```

```
## [1]  2  4  1  1 NA
```

```
care_data$n_kids # base R way
```

```
## [1]  2  4  1  1 NA
```

```
care_data[["n_kids"]] # base R way
```

```
## [1]  2  4  1  1 NA
```

```
care_data %>% select(n_kids) # tidy way
```

```
## # A tibble: 5 x 1
##   n_kids
##    <dbl>
## 1      2
## 2      4
## 3      1
## 4      1
## 5     NA
```

```
care_data["n_kids"] # base R way
```

```
## # A tibble: 5 x 1
##   n_kids
##    <dbl>
## 1      2
## 2      4
## 3      1
## 4      1
## 5     NA
```

Data Type

```
## Type issues
care_data %>%
  mutate(spending_per_child = n_kids / child_care_costs)
```

```
## # A tibble: 5 x 5
##      id n_kids child_care_costs random_noise spending_per_child
##   <int>  <dbl>            <dbl>        <dbl>              <dbl>
## 1     1      2             1000        -50.0            0.002
## 2     2      4             3000        109.             0.00133
## 3     3      1              300        -94.5            0.00333
## 4     4      1              300         30.7            0.00333
## 5     5     NA              500        114.            NA
```

```
care_data <- care_data %>%
  mutate(spending_per_child = n_kids / child_care_costs)

glimpse(care_data)
```

```
## Rows: 5
## Columns: 5
## $ id                 <int> 1, 2, 3, 4, 5
## $ n_kids             <dbl> 2, 4, 1, 1, NA
## $ child_care_costs   <dbl> 1000, 3000, 300, 300, 500
## $ random_noise       <dbl> -49.99721, 109.42974, -94.45097, 30.70099, 114.28161
## $ spending_per_child <dbl> 0.002000000, 0.001333333, 0.003333333, 0.003333333,~
```

```
# logical, also known as booleans
type_logical <- FALSE
type_logical <- TRUE

# integer and double, together are called: numeric
type_integer <- 1L
type_double <- 1.0

type_character <- "abbreviated as chr"
type_character <- "also known as a string"
```

Testing types

```
a <- "1"
typeof(a)
```

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

```
is.integer(a)
```

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

```
is.character(a)
```

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

```
typeof(care_data$child_care_costs)
```

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

```
typeof(care_data$n_kids)
```

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

Type coercion

```
a <- "4"
a + 3 # check - is there any issue with this syntax?
```

```
## Error in a + 3: non-numeric argument to binary operator
```

```
as.integer(a) + 3
```

```
## [1] 7
```

```
as.numeric(a) + 3
```

```
## [1] 7
```

```
care_data %>%
  mutate( n_kids = as.integer(n_kids),
          spending_per_kid = child_care_costs / n_kids)
```

```
## # A tibble: 5 x 6
##      id n_kids child_care_costs random_noise spending_per_child spending_per_kid
##   <int>  <int>            <dbl>        <dbl>              <dbl>            <dbl>
## 1     1      2             1000        -50.0            0.002                500
## 2     2      4             3000        109.             0.00133              750
## 3     3      1              300        -94.5            0.00333              300
## 4     4      1              300         30.7            0.00333              300
## 5     5     NA              500        114.            NA                     NA
## NAs introduced by coercion
as.integer("Unknown")
```

```
## Warning: NAs introduced by coercion

## [1] NA
```

```
## `NA`s are contagious
NA + 4
```

```
## [1] NA
```

```
max(c(NA, 4, 1000))
```

```
## [1] NA
```

```
max(c(NA, 4, 1000), na.rm = TRUE)
```

```
## [1] 1000
```

```
max(c(NA, 4, 1000), na.rm = FALSE)
```

```
## [1] NA
```

```
## Automatic coercion
paste0("str", "ing")
```

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

```
paste0(1L, "ing")
```

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

```
TRUE + 4
```

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

```
FALSE + 4
```

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

```
paste0(FALSE, "?")
```

```
## [1] "FALSE?"
```

```
paste0(TRUE, "?")
```

```
## [1] "TRUE?"
```

```
mean(c(TRUE, TRUE, FALSE, FALSE, TRUE))
```

```
## [1] 0.6
```

```
## `NA`s are contagious, redux.
b <- c(NA, 3, 4, 5)
sum(b)
```

```
## [1] NA
```

```
sum(b, na.rm = TRUE)
```

```
## [1] 12
```

Subsetting vectors

```
letters[[3]] # Use `[[` for subsetting a single value
```

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

```
letters[c(25,5,19)] # Use `[` for subsetting multiple values
```

```
## [1] "y" "e" "s"
```

```
my_numbers <- c(2, 4, 6, 8, 10)

# get all numbers besides the 1st
my_numbers[-1]
```

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

```
# get all numbers besides the 1st and second
my_numbers[-c(1,2)]
```

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

```
# get all numbers where true
my_numbers[c(TRUE, FALSE, FALSE, TRUE, FALSE)]
```

```
## [1] 2 8
```

```
my_numbers[my_numbers > 4]
```

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

```
# example
numerators <- rep(1, 11)
denominators <- 2 ^ c(0:10)
sum(numerators/denominators)
```

```
## [1] 1.999023
```

```
# a list holding multiple types
a_list <- list(1L, "fun", c(1,2,3))
typeof(a_list)
```

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

```
typeof(a_list[[1]])
```

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

```
typeof(a_list[[2]])
```

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

```
typeof(a_list[[3]])
```

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