YaRrr! The Pirate's Guide to R

Nathaniel D. Phillips

2017-02-21

Contents

Chapter 1

Preface

Chapter 2

Getting Started

Chapter 3

Jump In!

Chapter 4

The Basics

Chapter 5

Scalers and vectors

Chapter 6

Vector functions

Chapter 7

Indexing Vectors

Chapter 8

Chapter solutions

8.1 Chapter 4: The Basics

2. Which (if any) of the following objects names is/are invalid?

```
thisone <- 1
THISONE <- 2
this.one <- 3
This.1 <- 4
ThIS.....ON...E <- 5
This!One! <- 6           # only this one!
lkjasdfkjsdf <- 7
```

3. 2015 was a good year for pirate booty - your ship collected 100,000 gold coins. Create an object called `gold.in.2015` and assign the correct value to it.

```
gold.in.2015 <- 100800
```

4. Oops, during the last inspection we discovered that one of your pirates Skippy McGee hid 800 gold coins in his underwear. Go ahead and add those gold coins to the object `gold.in.2015`. Next, create an object called `plank.list` with the name of the pirate thief.

```
gold.in.2015 <- gold.in.2015 + 800
plank.list <- "Skippy McGee"
```

5. Look at the code below. What will R return after the third line? Make a prediction, then test the code yourself.

```
a <- 10
a + 10
a       # It will return 10 because we never re-assigned a!
```

8.2 Chapter 5: Scalers and vectors

1. Create the vector [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] in three ways: once using `c()`, once using `a:b`, and once using `seq()`.

```
c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

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

```
1:10
```

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

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

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

2. Create the vector [2.1, 4.1, 6.1, 8.1] in two ways, once using `c()` and once using `seq()`

```
c(2.1, 6.1, 6.1, 8.1)
```

```
## [1] 2.1 6.1 6.1 8.1
```

```
seq(from = 2.1, to = 8.1, by = 2)
```

```
## [1] 2.1 4.1 6.1 8.1
```

3. Create the vector [0, 5, 10, 15] in 3 ways: using `c()`, `seq()` with a `by` argument, and `seq()` with a `length.out` argument.

```
c(0, 5, 10, 15)
```

```
## [1]  0  5 10 15
```

```
seq(from = 0, to = 15, by = 5)
```

```
## [1]  0  5 10 15
```

```
seq(from = 0, to = 15, length.out = 4)
```

```
## [1]  0  5 10 15
```

4. Create the vector [101, 102, 103, 200, 205, 210, 1000, 1100, 1200] using a combination of the `c()` and `seq()` functions

```
c(seq(from = 101, to = 103, by = 3),
  seq(from = 200, to = 210, by = 5),
  seq(from = 1000, to = 1200, by = 100))
```

```
## [1]  101  200  205  210 1000 1100 1200
```

5. A new batch of 100 pirates are boarding your ship and need new swords. You have 10 scimitars, 40 broadswords, and 50 cutlasses that you need to distribute evenly to the 100 pirates as they board. Create a vector of length 100 where there is 1 scimitar, 4 broadswords, and 5 cutlasses in each group of 10. That is, in the first 10 elements there should be exactly 1 scimitar, 4 broadswords and 5 cutlasses. The next 10 elements should also have the same number of each sword (and so on).

```
swords <- rep(c("scimitar", rep("broadswoard", 4), rep("cutlass", 5)), times = 100)
head(swords)
```

```
## [1] "scimitar"    "broadswoard" "broadswoard" "broadswoard" "broadswoard"
## [6] "cutlass"
```

6. Create a vector that repeats the integers from 1 to 5, 10 times. That is [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, ...]. The length of the vector should be 50!

```
rep(1:5, times = 10)
```

```
##  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
## [36] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

7. Now, create the same vector as before, but this time repeat 1, 10 times, then 2, 10 times, etc., That is [1, 1, 1, ..., 2, 2, 2, ..., ... 5, 5, 5]. The length of the vector should also be 50

```
rep(1:5, each = 10)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4
## [36] 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5
```

8. Create a vector containing 50 samples from a Normal distribution with a population mean of 20 and standard deviation of 2.

```
rnorm(n = 50, mean = 20, sd = 2)
```

```
##  [1] 21.57679 20.27152 21.59335 20.20740 18.87735 21.82037 19.17224
##  [8] 24.48366 18.45996 24.64259 23.44068 17.63007 22.00316 19.94066
## [15] 21.83414 22.18860 21.30814 24.53073 19.67244 19.92957 19.83128
## [22] 21.01343 20.08321 21.34635 17.73714 18.42768 20.19274 18.39513
## [29] 20.73091 20.67682 16.01616 21.96972 21.88846 17.06943 16.90655
## [36] 19.18042 19.06928 19.39397 19.52724 22.59737 19.13177 19.22890
## [43] 21.58850 20.38277 18.30072 20.11989 19.89328 22.10367 19.42043
## [50] 19.57934
```

9. Create a vector containing 25 samples from a Uniform distribution with a lower bound of -100 and an upper bound of -50.

```
runif(n = 25, min = -100, max = -50)
```

```
##  [1] -58.42677 -63.11824 -61.29505 -96.01717 -79.79146 -81.19144 -61.69489
##  [8] -64.18679 -52.28968 -55.43486 -80.57633 -70.13589 -68.45541 -83.98283
## [15] -90.09987 -68.60028 -78.45546 -98.83198 -95.89134 -88.96217 -82.36385
## [22] -86.61534 -90.95168 -67.64108 -64.61321
```

8.3 Chapter 4: Vector Functions

1. Create a vector that shows the square root of the integers from 1 to 10.

```
(1:10) ^ .5
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
##  [8] 2.828427 3.000000 3.162278
```

```
#or

sqrt(1:10)
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
##  [8] 2.828427 3.000000 3.162278
```

2. Renata thinks that she finds more treasure when she's had a mug of grogg than when she doesn't. To test this, she recorded how much treasure she found over 7 days without drinking any grogg (ie., sober), and then did the same over 7 days while drinking grogg (ie., drunk). Here are her results:

How much treasure did Renata find on average whe she was sober? What about when she was drunk?

```
sober <- c(2, 0, 3, 1, 0, 3, 5)
drunk <- c(0, 0, 1, 0, 1, 2, 2)

mean(sober)
```

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

```
mean(drunk)
```

Table 8.1: Renata's treasure haul when she was sober and when she was drunk

day	sober	drunk
Monday	2	0
Tuesday	0	0
Wednesday	3	1
Thursday	1	0
Friday	0	1
Saturday	3	2
Sunday	5	2

```
## [1] 0.8571429
```

3. Using Renata's data again, create a new vector called `difference` that shows how much more treasure Renata found when she was drunk and when she was not. What was the mean, median, and standard deviation of the difference?

```
difference <- sober - drunk

mean(difference)
```

```
## [1] 1.142857
```

```
median(difference)
```

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

```
sd(difference)
```

```
## [1] 1.345185
```

4. There's an old parable that goes something like this. A man does some work for a king and needs to be paid. Because the man loves rice (who doesn't?!), the man offers the king two different ways that he can be paid. *You can either pay me 100 kilograms of rice, or, you can pay me as follows: get a chessboard and put one grain of rice in the top left square. Then put 2 grains of rice on the next square, followed by 4 grains on the next, 8 grains on the next…and so on, where the amount of rice doubles on each square, until you get to the last square. When you are finished, give me all the grains of rice that would (in theory), fit on the chessboard.* The king, sensing that the man was an idiot for making such a stupid offer, immediately accepts the second option. He summons a chessboard, and begins counting out grains of rice one by one… Assuming that there are 64 squares on a chessboard, calculate how many grains of rice the main will receive. If one grain of rice weights 1/64000 kilograms, how many kilograms of rice did he get? *Hint: If you have trouble coming up with the answer, imagine how many grains are on the first, second, third and fourth squares, then try to create the vector that shows the number of grains on each square. Once you come up with that vector, you can easily calculate the final answer with the `sum()` function.*

```
# First, let's create a vector of the amount of rice on each square:
rice <- 2 ^ (1:64)

# Here are the first few spaces
head(rice)
```

```
## [1]  2  4  8 16 32 64
```

```
# The result is just the sum!
rice.total <- sum(rice)
rice.total
```

```
## [1] 3.689349e+19
```

```
# How much does that weigh? Each grain weights 1/6400 kilograms:
rice.kg <- sum(rice) * 1/6400
rice.kg
```

```
## [1] 5.764608e+15
```

```
# That's 5,800,000,000,000,000 kilograms of rice. Let's keep going....
# A kg of rice is 1,300 calories

rice.cal <- rice.kg * 1300
rice.cal
```

```
## [1] 7.49399e+18
```

```
# How many people can that feed for a year?
# A person needs about 2,250 calories a day, or 2,250 * 365 per year

rice.people.year <- rice.cal / (2250 * 365)
rice.people.year
```

```
## [1] 9.125102e+12
```

```
# So, that amount of rice could feed 9,100,000,000,000 for a year
# Assuming that the averge lifespan is 70 years, how many lifespans could this feed?

rice.people.life <- rice.people.year / 70
rice.people.life
```

```
## [1] 130358595868
```

```
# Ok...so it could feed 130,000,000,000 (130 billion) people over their lives

# Conclusion: King done screwed up.
```

```
# I want to remove the white separations from the following chunk output

1+1
```

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

```
2+4
```

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

Chapter 9

Placeholder

Bibliography