R for Data Analytics Part 1, Lecture 2

Michèle Fille

Table of contents

# Lecture 2 - R Fundamentals: Functions and Vectors

## 2.1. Functions

### Exercise 2.1. Functions

#### Exercise 2.1 – Task 1: Calling Built-in Functions

a) Create a variable my\_name that contains your name

b) Create a variable name\_length that holds how many letters (including spaces) are in your name.

* Hint: Use nchar().

c) Print the number of letters in your name

d) Create a variable now\_doing that is your name followed by “is programming!”.

* Hint: Use paste().

e) Make the now\_doing variable upper case. Hint: Use the toupper() function.

my\_name <- "Michèle Fille"  
name\_length <- nchar(my\_name)  
name\_length

[1] 13

now\_doing <- paste(my\_name, "is programming")  
toupper(now\_doing)

[1] "MICHÈLE FILLE IS PROGRAMMING"

#### Exercise 2.1 – Task 2: Calling Built-in Functions (continued)

a) Pick two of your favorite numbers (between 1 and 100) and assign them to variables fav\_1 and fav\_2

b) Divide each number by the square root of 201 and save the new value in the original variable.

c) Create a variable raw\_sum that is the sum of the two variables. Use the sum() function for practice.

d) Create a variable round\_sum that is the raw\_sum rounded to 1 decimal place. Use the round() function.

e) Create two new variables round\_1 and round\_2 that are your fav\_1 and fav\_2 variables rounded to 1 decimal places.

f) Create a variable sum\_round that is the sum of the rounded values.

g) Which is bigger, round\_sum or sum\_round? (You can use the max() function!)

fav\_1 <- 7  
fav\_2 <- 11  
  
fav\_1 <- fav\_1 / (sqrt(201))  
fav\_2 <- fav\_2 / (sqrt(201))  
fav\_1

[1] 0.4937419

fav\_2

[1] 0.7758802

raw\_sum <- sum(fav\_1,fav\_2)  
raw\_sum

[1] 1.269622

round\_sum <- round(raw\_sum, 1)  
round\_sum

[1] 1.3

round\_1 <- round(fav\_1, 1)  
round\_2 <- round(fav\_2, 1)  
round\_1

[1] 0.5

round\_2

[1] 0.8

sum\_round <- sum(round\_1, round\_2)  
sum\_round

[1] 1.3

max(round\_sum, sum\_round)

[1] 1.3

#### Exercise 2.1 – Task 3: Writing and Executing Functions

a) Define a function add\_three() that takes a single argument and returns a value that is 3 greater than the input.

b) Create a variable ten that is the result of passing 7 to your add\_three function.

c) Define a function imperial\_to\_metric() that takes in 2 arguments: a number of feet and a number of inches. The function should return the equivalent length in meters.

* Remark: Look up the conversion formula with your preferred web browser.

d) Create a variable height\_in\_meters by passing your height in imperial to your imperial\_to\_metric() function.

add\_three <- function(num){  
 result <- num + 3  
 return(result)  
}  
  
ten <- add\_three(7)  
ten

[1] 10

imperial\_to\_metric <- function(feet, inches) {  
 total\_inches <- feet \* 12 + inches  
 meters <- total\_inches \* 0.0254  
 meters # return the value in meters  
}  
  
height\_in\_meters <- imperial\_to\_metric(5,4)  
height\_in\_meters

[1] 1.6256

### Self-Study 2.1. Functions

#### Self-Study 2.1 - Task 1: Using Built-In String Functions

a) Create a variable lyric that contains the text “I like to eat apples and bananas”.

b) Use the substr() function to extract the 1st through 13th letters from lyric, and store the result in a variable called intro.

* Hint: Use ?substr to see more about this function.

c) Use the substr() function to extract the 15th through the last letter of lyric, and store the result in a variable called fruits.

* Hint: Use nchar() to determine how many total letters there are!

d) Use the gsub() function to substitute all the “a”s in fruits with “ee”. Store the result in a variable called fruits\_e.

* Hint: use ?gsub to see more about the function and see <http://www.endmemo.com/program/R/sub.php> for a simple example.

e) Use the gsub() function to substitute all the “a”s in fruits with “o”. Store the result in a variable called fruits\_o

f) Create a new variable lyric\_e that is the intro combined with the new fruits\_e ending. Print out this variable.

g) Without making a new variable, print out the intro combined with the new fruits\_o ending.

lyric <- "I like to eat apples and bananas."  
  
intro <- substr(lyric, start = 1, stop = 13)  
intro

[1] "I like to eat"

fruits <- substr(lyric, start = 15, stop = nchar(lyric))  
fruits

[1] "apples and bananas."

fruits\_e <- gsub("a", "ee", fruits)  
fruits\_e

[1] "eepples eend beeneenees."

fruits\_o <- gsub("a", "o", fruits)  
fruits\_o

[1] "opples ond bononos."

lyric\_e <- paste(intro, fruits\_e)  
lyric\_e

[1] "I like to eat eepples eend beeneenees."

paste(intro, fruits\_o)

[1] "I like to eat opples ond bononos."

#### Self-Study 2.1 - Task 2: Functions and Conditionals

a) Define a function is\_twice\_as\_long() that takes in two character strings, and returns whether or not (e.g., a boolean) the length of one argument is greater than or equal to twice the length of the other.

* Hint: Compare the length difference to the length of the smaller string

b) Call your is\_twice\_as\_long() function by passing it different length strings to confirm that it works.

* Hint: Make sure to check when either argument is twice as long, as well as when neither are!

c) Define a function describe\_difference() that takes in two strings. The function should return one of the following sentences as appropriate:

* “Your first string is longer by N characters”
* “Your second string is longer by N characters”
* “Your strings are the same length!”

d) Call your describe\_difference() function by passing it different length strings to confirm that it works.

* Hint: Make sure to check all 3 conditions!

is\_twice\_as\_long <- function(string\_1, string\_2){  
 len\_1 <- nchar(string\_1)  
 len\_2 <- nchar(string\_2)  
   
 if (len\_1 >= 2 \* len\_2 || len\_2 >= 2 \* len\_1) {  
 return(TRUE)  
 } else {  
 return(FALSE)  
 }  
}  
  
is\_twice\_as\_long("Michele", "M")

[1] TRUE

is\_twice\_as\_long("M", "Michele")

[1] TRUE

is\_twice\_as\_long("Michele", "Michele")

[1] FALSE

# or  
  
is\_twice\_as\_long\_gwen <- function(str1, str2) {  
 diff <- abs(nchar(str1) - nchar(str2))  
 min\_length <- min(nchar(str1), nchar(str2))  
 diff >= min\_length # if difference is more than short  
}  
  
is\_twice\_as\_long\_gwen("Michele", "M")

[1] TRUE

describe\_difference <- function(string\_1, string\_2){  
 len\_1 = nchar(string\_1)  
 len\_2 = nchar(string\_2)  
   
 if (len\_1 > len\_2) {  
 difference = len\_1 - len\_2  
 return(sprintf("Your first string is longer by %d characters", difference))  
 } else if (len\_2 > len\_1) {  
 difference = len\_2 - len\_1  
 return(sprintf("Your second string is longer by %d characters", difference))  
 } else {  
 return("Your strings are the same length!")  
 }  
}  
  
describe\_difference("Michele", "M")

[1] "Your first string is longer by 6 characters"

describe\_difference("M", "Michele")

[1] "Your second string is longer by 6 characters"

describe\_difference("Michele", "Michele")

[1] "Your strings are the same length!"

# or  
  
describe\_difference\_gwen <- function(first, second) {  
 diff <- nchar(first) - nchar(second)  
 if (diff > 0) {  
 sentence <- paste("Your first string is longer by", diff, "characters")  
 } else if (diff < 0) {  
 sentence <- paste("Your second string is longer by", -diff, "characters")  
 } else {  
 sentence <- "Your strings are the same length!"  
 }  
 sentence # return the sentence  
}

### sprintf (additional input from Michèle)

In R, sprintf() supports several placeholders for different types of values. Here are the main ones:

* %d: Represents an integer value.
* %f: Represents a floating-point value (decimal number).
* %s: Represents a string value.
* %x, %X: Represents an integer value in hexadecimal format (lowercase or uppercase).
* %o: Represents an integer value in octal format.
* %e, %E: Represents a floating-point value in scientific notation (lowercase or uppercase).
* %g, %G: Represents a floating-point value, using %f or %e depending on the magnitude of the value (lowercase or uppercase).

## 2.2. Vectors

### Exercise 2.2. Vectors

#### Exercise 2.2 – Task 1: Creating Vectors and Operating on Vectors

a) Create a vector names that contains your name and the names of 2 people next to you. Print the vector.

b) Use the colon operator : to create a vector n of numbers from 10 to 49.

* Use the length() function to get the number of elements in n.
* Add 1 to each element in n and print the result.

c) Create a vector m that contains the numbers 10 to 1 (in that order).

* Hint: use the seq() function.

d) Subtract the m FROM n.

* Remark: Note the recycling!

names <- c("Michèle", "Rahel", "Jervin")  
print(names)

[1] "Michèle" "Rahel" "Jervin"

n <- 10:49  
print(n)

[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34  
[26] 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49

length(n)

[1] 40

print(n+1)

[1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35  
[26] 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

m <- seq(10,1)  
print(m)

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

result <- n-m  
print(result)

[1] 0 2 4 6 8 10 12 14 16 18 10 12 14 16 18 20 22 24 26 28 20 22 24 26 28  
[26] 30 32 34 36 38 30 32 34 36 38 40 42 44 46 48

#### Exercise 2.2 – Task 2: Indexing and Filtering Vectors

a) Create a vector first\_ten that has the values 10 through 20 in it (using the colon : operator).

b) Create a vector next\_ten that has the values 21 through 30 in it (using the seq() function).

c) Create a vector all\_numbers by combining the previous two vectors.

d) Create a variable eleventh that contains the 11th element in all\_numbers.

e) Create a vector some\_numbers that contains the 2nd through the 5th elements of all\_numbers.

f) Create a vector even that holds the even numbers from 1 to 100.

g) Using the all() function and the %% (modulo) operator, confirm that all of the numbers in your even vector are even.

first\_ten <- 10:20  
print(first\_ten)

[1] 10 11 12 13 14 15 16 17 18 19 20

next\_ten <- seq(21,30)  
print(next\_ten)

[1] 21 22 23 24 25 26 27 28 29 30

all\_numbers <- c(first\_ten, next\_ten)  
print(all\_numbers)

[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

eleventh <- all\_numbers[11]  
print(eleventh)

[1] 20

some\_numbers <- all\_numbers[2:5]  
print(some\_numbers)

[1] 11 12 13 14

even <- seq(2,100,2) # need to start at 2, as it is 1st even num  
print(even)

[1] 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38  
[20] 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76  
[39] 78 80 82 84 86 88 90 92 94 96 98 100

all(even %% 2 == 0)

[1] TRUE

### Self-Study 2.2. Vectors

#### Self-Study 2.2 - Task 1: Creating Vectors and Operating on Vectors

a) Use the seq() function to produce a range of numbers from -5 to 10 in 0.1 increments. Store it in a variable x\_range.

1. Create a vector sin\_wave by calling the sin() function on each element in x\_range.

c) Plot your sine wave using ggplot2.

* Hint: To pass the data to ggplot, combine it into a data frame that contains x\_range as the first column and sin\_wave as the second column.

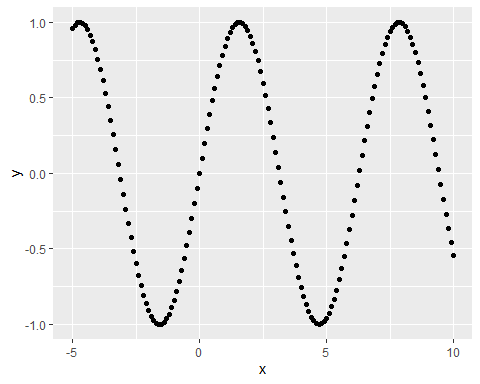
d) Create a vector cos\_wave by calling the cos() function on each element in x\_range. Plot your sine wave using ggplot2.

e) Create a vector wave by multiplying sin\_wave and cos\_wave together, then adding sin\_wave to the product. Plot the result using ggplot2.

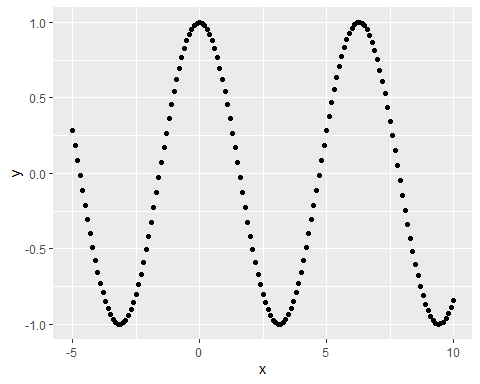
x\_range <- seq(-5,10,0.1)  
print(x\_range)

[1] -5.0 -4.9 -4.8 -4.7 -4.6 -4.5 -4.4 -4.3 -4.2 -4.1 -4.0 -3.9 -3.8 -3.7 -3.6  
 [16] -3.5 -3.4 -3.3 -3.2 -3.1 -3.0 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4 -2.3 -2.2 -2.1  
 [31] -2.0 -1.9 -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1 -1.0 -0.9 -0.8 -0.7 -0.6  
 [46] -0.5 -0.4 -0.3 -0.2 -0.1 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9  
 [61] 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 2.4  
 [76] 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 3.8 3.9  
 [91] 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0 5.1 5.2 5.3 5.4  
[106] 5.5 5.6 5.7 5.8 5.9 6.0 6.1 6.2 6.3 6.4 6.5 6.6 6.7 6.8 6.9  
[121] 7.0 7.1 7.2 7.3 7.4 7.5 7.6 7.7 7.8 7.9 8.0 8.1 8.2 8.3 8.4  
[136] 8.5 8.6 8.7 8.8 8.9 9.0 9.1 9.2 9.3 9.4 9.5 9.6 9.7 9.8 9.9  
[151] 10.0

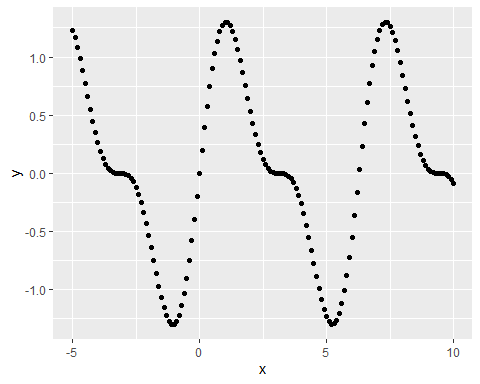
sin\_wave <- sin(x\_range)  
  
library("ggplot2")  
df\_sin\_wave <- data.frame(x = x\_range, y = sin\_wave)  
ggplot(df\_sin\_wave, aes(x = x, y = y)) +  
 geom\_point()



cos\_wave <- cos(x\_range)  
  
df\_cos\_wave <- data.frame(x = x\_range, y = cos\_wave)  
ggplot(df\_cos\_wave, aes(x = x, y = y)) +  
 geom\_point()



wave <- (sin\_wave \* cos\_wave) + sin\_wave  
  
df\_wave <- data.frame(x = x\_range, y = wave)  
ggplot(df\_wave, aes(x = x, y = y)) +  
 geom\_point()



#### Self-Study 2.2 - Task 2: Indexing and Filtering Vectors

a) Create a vector phone\_numbers that contains the numbers 8, 6, 7, 5, 3, 0, 9.

b) Create a vector prefix that has the first three elements of phone\_numbers.

c) Create a vector small that has the values of phone\_numbers that are less than or equal to 5.

d) Create a vector large that has the values of phone\_numbers that are strictly greater than 5.

e) Replace the values in phone\_numbers that are larger than 5 with the number 5.

f) Replace every odd-numbered value in phone\_numbers with the number 0.

phone\_numbers <- c(8, 6, 7, 5, 3, 0, 9)  
print(phone\_numbers)

[1] 8 6 7 5 3 0 9

prefix <- phone\_numbers[1:3]  
print(prefix)

[1] 8 6 7

small <- phone\_numbers[phone\_numbers <= 5]  
print(small)

[1] 5 3 0

large <- phone\_numbers[phone\_numbers > 5]  
print(large)

[1] 8 6 7 9

phone\_numbers[phone\_numbers > 5] <- 5  
print(phone\_numbers)

[1] 5 5 5 5 3 0 5

phone\_numbers[phone\_numbers %% 2 == 1] <- 0  
print(phone\_numbers)

[1] 0 0 0 0 0 0 0

#### Self-Study 2.2 - Task 3: Vector Practice

a) Create a vector words of 6 (or more) words. You can Google for a “random word generator” if you wish!

b) Create a vector words\_of\_the\_day that is your words vector with the string “is the word of the day!” pasted on to the end.

* BONUS: Surround the word in quotes (e.g., 'data' is the word of the day!).
* Remark: Note that the results are more obviously correct with single quotes.

c) Create a vector a\_f\_words which are the elements in words that start with “a” through “f”.

* Hint: Use a comparison operator to see if the word comes before “f” alphabetically!
* Hint: Make sure all the words are lower-case, and only consider the first letter of the word!

d) Create a vector g\_m\_words which are the elements in words that start with “g” through “m”.

e) Define a function word\_bin that takes in three arguments: a vector of words, and two letters. The function should return a vector of words that go between those letters alphabetically.

f) Use your word\_bin function to determine which of your words start with “e” through “q”.

words <- c("flower", "Ralunkel", "viola", "dog", "Good", "Light", "perfume", "Cookie")  
  
words\_of\_the\_day <- paste0("'", words, "' is the word of the day!") # no space between ' and the word form words vector  
words\_of\_the\_day

[1] "'flower' is the word of the day!" "'Ralunkel' is the word of the day!"  
[3] "'viola' is the word of the day!" "'dog' is the word of the day!"   
[5] "'Good' is the word of the day!" "'Light' is the word of the day!"   
[7] "'perfume' is the word of the day!" "'Cookie' is the word of the day!"

words\_lower <- tolower(words) # make sure all words are lowercase  
a\_f\_words <- words\_lower[substring(words\_lower, 1, 1) <= "f"] # make a substring of first char, and check if it is smaller or equal to f  
a\_f\_words

[1] "flower" "dog" "cookie"

g\_m\_words <- words\_lower[words\_lower >= "g" & substring(words\_lower, 1, 1) <= "m"]  
g\_m\_words

[1] "good" "light"

word\_bin <- function(vector\_of\_words, letter\_1, letter\_2){  
 # all input to lower  
 vector\_of\_words <- tolower(vector\_of\_words)  
 letter\_1 <- tolower(letter\_1)  
 letter\_2 <- tolower(letter\_2)  
   
 words\_result <- vector\_of\_words[substring(vector\_of\_words, 1, 1) >= letter\_1 & substring(vector\_of\_words, 1, 1) <= letter\_2]  
 return(words\_result)  
}  
  
word\_bin(words, "e", "q")

[1] "flower" "good" "light" "perfume"