Assignment 2.1

2022 Programming in Psychological Science

Contents

Assignment description

Work through the following 32 R problems. This assignment is worth 8 points (.25 points per problem). You should also complete Assignment 2.2P OR Assignment 2.2R for an additional 2 points. The total of Assignments 2.1 and your choice of either 2.2P (Python) or 2.2R (R challenge) are worth 10 points and 15% of your final course grade. You must work individually, but feel free to ask questions in class and on Slack!

You must submit the problems as a R script (.R file). All answers must have comments and be labeled with the problem number. Some answers will only be comments without associated R code. For questions about output, report your output in a comment under the relevant line of code.

Many problems have more than one solution. Best solutions are those that follow style guidelines (https://style.tidyverse.org/), have clear variable names, and have comments so that other programmers (and your future self) can read and adapt the code.

Problems

Q2.1.1

Does this code run properly? If not, what can you do to fix this code?

```
best_sport <- "voetbal"
if (best_sport = "soccer") {
  print(TRUE)
}
```

Q2.1.2

```
shot <- "left"
jumped <- "right"
goals <- 0
if (shot != jumped) {
  goals <- goals + 1
}
print(goals)
```

What happens if you change `shot` to `"right"`?

Why? What happens if **instead** you change `jumped` to `"forward"`? Why?

What happens if **instead** you change `jumped` to `"Left"` with a capital `L`? Why?

Q2.1.3

```
shot <- "missed"
jumped <- "right"
our_goals <- 1
their_goals <- 1
if (shot != jumped) {
  our_goals <- our_goals + 1
  } else if (shot == "missed") {
    print("Coach OUTRAGED")
  } else {
    our_goals <- our_goals + 0
    print("Sad sport fans.")
}
print(our_goals)
```

How can you change this **if statement** to print `"Coach OUTRAGED"` when `shot` equals `"missed"`?

Q2.1.4

```
shot <- "missed"
jumped <- "right"
our_goals <- 1
their_goals <- 1
if (shot != jumped) {
```

```
  our_goals <- our_goals + 1
  } else if (shot = "missed") {
    print("Coach OUTRAGED")
  } else if (shot == jumped) {
    our_goals <- our_goals + 0
    print("Sad sport fans.")
  } else {
    print("Everyone confused")
  }
print(our_goals)
```

Will this code ever print `"Everyone confused"`? Why or why not?

Change this the *if-else* statement so that `"Everyone confused"` prints when the variable `shot` is something other than `"right"`, `"left"`, or `"missed"`.

Q2.1.5

```
small_numbers <-  seq(0, 1, by=.001)
they_are_small <-  FALSE
if (small_numbers < 1.1) {
  they_are_small <- TRUE
}
```

Does this code test whether `all` the numbers are less than 1.1? If not, how would you change the code to do this? Note that you **must keep the if statement**.

Q2.1.6

```
student_id <- 1234
set.seed(student_id)
ages <- runif(100, min=0, max=100)
```

Set the seed to your student ID. Return a vector of strings `"working"` if `ages < 66.33` and `"retired"` if `ages >= 66.33` using the `ifelse()` command.

Then return a vector of logicals: `TRUE` if `ages < 66.33` and `FALSE` if `ages >= 66.33` using only a logical statement.

Q2.1.7

```
data_obs = c(6, 7, 8, 9, -3000, 5, 1, 7, -4000)
```

Change the negative numbers in `data_obs` to `NA` in a **for loop**.

Then, in a new line of code, change the negative numbers in `data_obs` to `NA` **without** any loop.

Q2.1.8

```
two_dice <- sample(1:6, 2, replace=TRUE)
```

Consider a game of dice in which two dice are rolled and a score is determined from the roll.

The score is determined as follows: If the sum of the two dice is even, the score equals the sum of the dots (pips) multiplied by the smallest number of the two dice. For example, if you roll a 3 and a 5 then sum, 8, is even. The score is 3 * (3+5)=24.

If the sum of the two dice is odd, your score is the number of dots (pips) multiplied by -3. For example, if you roll a 2 and a 5 then sum, 7, is odd. The score is -3 * (2+5)=-21.

Write an **if-else statement** that calculates your score based on the values of the dice.

Q2.1.9

Generate a random draw from `rnorm` and call this `my_num`. Then write an **if-else statement** that does the following:

If `my_num` is less than 0, `my_num` is replaced by the absolute value of `my_num`. If **not** (i.e. `my_num` is **not** less than 0), then add 1000 to `my_num`.

Q2.1.10

Create the vector `c(3, 5, 7, 9, 11, 13, 15, 17)` using a **for loop**. Start with `double_plus <- numeric()` and fill the vector using the for loop.

Q2.1.11

Create the vector `c(3, 5, 9, 17, 33, 65, 129, 257, 513, 1025)` by adding something to each previous element in a **for loop**. You can start with the vector `c(3)`

Q2.1.12

Write a **for loop** that performs the following operations on your variable `numeric_vec` of an arbitrary size.

Even elements should be subtracted and odd elements should be added. For instance, the first element of the vector should be subtracted from the second element. The third element of the vector should be added from the result of the previous step. The fourth element of the vector should be subtracted. And so on, until the end of the vector.

For example, if

```
numeric_vec <- 1:4
```

the loop would perform the following operations:

```
final_sum <- 0
final_sum <- final_sum + numeric_vec[1]
final_sum <- final_sum - numeric_vec[2]
final_sum <- final_sum + numeric_vec[3]
final_sum <- final_sum - numeric_vec[4]
```

Note that you can use a **if statement** within your for loop, but you don't have to if you're clever with math.

Q2.1.13

Write a **while loop** in which you roll three dice until the outcome is either 6-6-6 or 1-2-3.

Q2.1.14

Consider the following game of dice: A player rolls 3 dice. If all dice come up different the player wins 1 euro, otherwise he loses 3 euro.

Combine a *while loop* and *if else statement* to play this game until you either lose 1000 Euros, play 300 rounds, or win 20 Euros. Also `print` every dice roll number within the loop, e.g. `"Dice roll 49"`

Q2.1.15

```
names <- c("Clara", "Bence", "Enrico", "Leonhard", "Michael")
for (hannes in names) {
  print(hannes)
}
```

What is the variable `hannes` in this code? Why is this a bad variable name? Change this code so that it has better variables names.

Q2.1.16

`unique()` extracts the unique elements of a vector. We tried to write a code that would do the same and return the unique values in variable `special` but our code does not work. Try this code for some numeric vector `these_numbers`. Can you fix the code?

```
special <- numeric()
for (i in 1:length(these_numbers) ) {
  if (sum(these_numbers[1:i] == these_numbers[i]) == 1) {
    special <- c(special, i)
  }
}
```

Q2.1.17

```
integers <- 2:1000
primes <- c()
for (i in integers) {
  is_prime <- TRUE
  for (j in 2:(i-1)) {
    if (i %% j == 0) {
      is_prime <- FALSE
    }
  }
  if (is_prime) {
    primes <- c(primes, i)
  }
}
print(primes)
```

We created this code to print prime numbers between 1 and 1000. However there are two problems with this code:

The code returns an error for `integers <- 1:1000`. Why?

The code does not return the prime number 2. Why?

Q2.1.18

```
verhulst <- function(x, r, k) r*x*(k-x)/k
rabbits <- c(.001)
rate <- 2
capacity <- 1000
for (time in 2:50) rabbits[time] <- verhulst(rabbits[time-1], rate, capacity)
plot(rabbits, type='l', xlab='time', bty='n')
```

The code above simulates a Verhulst equation that describes the growth of rabbits over time (where the variable `rabbits` represents 1000 rabbits). Observe that in the code above we created our own **function** in one line. We can also create a **for loop** in one line. Notice the lack of brackets.

Rewrite the **function** to give default rate and capacity parameters with more descriptive variable names.

Q2.1.19

Using an explicit *for loop*, make a three dimensional `array` of 5 by 5 by 100. Each last dimension should contain a 5 by 5 matrix generated from random exponential numbers with rate given by last dimension index. That is, 25 numbers are placed into `array[, , 30]` drawn from an exponential distribution with rate = 30. See `?rexp`.

Q2.1.20

Take your previously generated 3 dimensional array in the last problem. Calculate the maximum value in each 5 by 5 matrix using an **implicit loop** in one line, see `?apply`.

Q2.1.21

```
t_tests <- replicate(100, t.test(rnorm(100, mean = 0.1)), simplify = FALSE)
```

Within the list `t_tests` you will find a complex list structure. Can you extract the 100 p.values and put them in a vector?

Hint: Use `sapply` with a in-line function `function(test) test\$p.value`.

Q2.1.22

```
marty_mcfly <- function(my_list) {
  print("Doc! Doc!")
  rm(list = my_list, pos = 1)
}
```

What happens when we call `marty_mcfly('marty_mcfly')`?

Q2.1.23

```
set.seed(1)
which_number <- sample(1:4000,1)
which_option <- which_number %% 4 + 1
division_result <- switch(which_option,
                          print("Divisible by 4"),
                          print("Remainder: 1"),
                          print("Remainder: 2"),
                          print("Remainder: 3"),
                          print("Remainder: 4"))
```

Try different random seeds in the first line. Which print statement in the **switch** function is never executed? Why?

Q2.1.24

```
set.seed(1)
caught_fish <- sample(c("large_trout", "small_trout", "carp", "perch",
```

```
                          "unknown", "shoe"),1)
print(caught_fish)
what_do <- switch(caught_fish,
                          large_trout = "eat",
                          small_trout = "return",
                          carp = ,
                          perch = "return",
                          shoe = )
print(what_do)
```

Read the help file for the function `switch`.

What is the variable `what_do` when `caught_fish` is `"large_trout"`? Why?

What is the variable `what_do` when `caught_fish` is `"unknown"`? Why?

What is the variable `what_do` when `caught_fish` is `"carp"`? Why?

What is the variable `what_do` when `caught_fish` is `"shoe"`? Why?

Q2.1.25

```
count <- function(number) {
  if (mode(number) != "numeric") {
    stop("`number` must be numeric." )
  }
  if (length(number) > 1) {
    stop("`number` must be of length 1" )
  }
  if (number < 0) {
    while (number > -100) {
      print(number)
      number <- number - 1
    }
  } else {
    while (number < 100) {
      if (number == 42) {
        print("I don't know the meaning of the universe")
        break
      }
      print(number)
      number <- number + 1
    }
  }
    return(invisible(number))
}
```

Describe what this `count` function does. You do not need to describe the function line by line, but you should describe what it prints in different scenarios.

Q2.1.26

Use the `count` function in the previous question.

Make the `count` function return a warning `"Will not start counting from 42."` when 42 is entered. All other behavior of the function should remain the same.

Then make the `count` function return an error if `number` is not an integer. See the help file for `round`.

Q2.1.27

Use the `count` function in the previous question. What happens when you run the following lines? How would `debug` be useful when writing your own functions? See the help file of `debug`.

```
count(matrix(c(1,2,3,4,ncol=2,nrow=2)))
debug(count)
count(matrix(c(1,2,3,4,ncol=2,nrow=2)))
undebug(count)
```

Q2.1.28

Write a function that returns `"Possibly sorted!"` when the minimum of a vector is in the first place and the maximum is in the last place. Otherwise the function returns `FALSE`. So `c(2, 4, 3, 2, 8)` gives `"Possibly sorted!"` and `c(3, 1, 6)` gives `FALSE`. Check whether `c(1, 4, 2, 8, 8)` works. It should return `"Possibly sorted!"`. If not, improve your function.

Q2.1.29

```
grass <- "green"
colorit <- function(color_me, grass_me) {
  grass_me <- grass
  color_me <- "blue"
  grass <- "blue"
  colorful_items = c(color_me, grass_me)
  return(colorful_items)
}
sky <- "grey"
ground <- "brown"
these_items <- colorit(sky, ground)
sky2 <- these_items[1]
ground2 <- these_items[2]
```

What are the colors of `grass`, `sky`, `sky2`, `ground`, and `ground2` after running this code? Why? Remember the discussion about **global** and **local** variables within and outside **functions**.

Q2.1.30

Save the `colorit` function without the preceding `grass <- "green"` line in a separate .R file then clear `colorit` and `grass` from your workspace with `rm(list=c("colorit","grass"))`. Now use `source` to load your new file to place `colorit` back in your workspace. Does the function work? Why or why not?

Make the function work when using `source` (in the same way as before) by changing one line in the function.

Q2.1.31

Write a function that gives the first `n` Fibonacci numbers. See https://en.wikipedia.org/wiki/Fibonacci_number

Q2.1.32

Write a function `nthroot()` that calculates, by iteration, the nth root of a number using the Newton's method (https://en.wikipedia.org/wiki/Newton%27s_method). The function should take as arguments, `number`, `n`, and a starting value `start`. The default starting value should be 1.

Hint: https://en.wikipedia.org/wiki/Newton%27s_method#Square_root