HW 2: Pipes, Functions, Logicals

Yunting

1/26/2020

Questions

1. (RDS Exercise 19.4.3) Implement a fizzbuzz() function which takes a single number as input.

Use logicals and conditionals as well as the operator %%
- If the number is divisible by three, return "fizz".
- If it's divisible by five, return "buzz".
- If it's divisible by three and five, return "fizzbuzz".
- Otherwise, return the input number.

```
fizzbuzz <- function(y){
  if(y%%5==0 & y%%3==0){return("fizzbuzz")}
  else if(y%%5==0){
    return("buzz")}
    else if (y%%3==0)
    {return("fizz")}
  else{return(y)}

}
```

a. Design your function. Write out (in words in a text block) the steps the function needs to accomplish

to convert the input into an output. For the above question, we need to follow:
1. set the number divisible 3 and 5 to "fizzbuzz" (return)
2. set the number divisible 3 to "fizz"(return)
3. set the number divisible 5 to "buzz"(return)
4. use else if function successfully.
5. make sure {} in the right position.
6. %% == 0, so we can set the divisor.

b. Show your output for the following inputs: 3, 5, 15, 2.

```
fizzbuzz <- function(y){
  if(y%%5==0 & y%%3==0){return("fizzbuzz")}
  else if(y%%5==0){
    return("buzz")}
    else if (y%%3==0)
    {return("fizz")}
  else{return(y)}
}

fizzbuzz(3)
```

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

```
fizzbuzz(5)
```

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

```
fizzbuzz(15)
```

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

```
fizzbuzz(2)
```

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

*text book

```
fizzbuzz <- function(x) {
  # these two lines check that x is a valid input
  stopifnot(length(x) == 1)
  stopifnot(is.numeric(x))
  if (!(x %% 3) && !(x %% 5)) {
    "fizzbuzz"
  } else if (!(x %% 3)) {
    "fizz"
  } else if (!(x %% 5)) {
    "buzz"
  } else {
    # ensure that the function returns a character vector
    as.character(x)
  }
}

fizzbuzz(3)
```

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

```
fizzbuzz(5)
```

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

```r
fizzbuzz(15)
```

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

```r
fizzbuzz(2)
```

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

c. Update your function to include error checking to ensure the input is both numeric and a single value.Test it on cat, and c(1,5). Remember in the code chunk where you run the function, set your code chunk parameter for error to be TRUE, e.g. {r, error=TRUE}, so it will knit with the error.

```r
fizzbuzz <- function(y){
stopifnot(length(y)==1)
  stopifnot(is.numeric(y))
  if(y%%5==0 & y%%3==0){return("fizzbuzz")}
  else if(y%%5==0){
    return("buzz")}
    else if (y%%3==0)
    {return("fizz")}
  else{return(y)}
}

fizzbuzz(cat)
```

```
## Error in fizzbuzz(cat): is.numeric(y) is not TRUE
```

```r
fizzbuzz(c(1,5))
```

```
## Error in fizzbuzz(c(1, 5)): length(y) == 1 is not TRUE
```

d. Complete your function by using comments in the code chunk, above the function, to document the function. Include the following elements: title, description, useage, arguments, and return value.

```r
# Implement a fizzbuzz() function which takes a single number as input
# y: is a single number as input
# If the number is divisible by three and five, @return "fizzbuzz"
# If the number is divisible by three, @return "fizz"
# If the number is divisible by five, @return "buzz"

fizzbuzz <- function(y){
stopifnot(length(y)==1)
  stopifnot(is.numeric(y))
  if(y%%5==0 & y%%3==0){return("fizzbuzz")}
```

```
  else if(y%%5==0){
    return("buzz")}
    else if (y%%3==0)
    {return("fizz")}
  else{return(y)}
}
```

2. (RDS Exercise 19.4.4) Write a function that uses the function cut() to simplify this set of nested if-else statements? Consider using -Inf and Inf. Note, this will also output the levels of the factors.

a. Show the output for inputs: 31, 30, 10, -10.

```
show_temp <- function(temp){seq(-10,50,by = 5)
cut(temp, c(-Inf,0,10,20,30,Inf),right = TRUE,
labels = c("freezing", "cold", "cool", "warm", "hot"))}

show_temp(31)
```

```
## [1] hot
## Levels: freezing cold cool warm hot
```

```
show_temp(30)
```

```
## [1] warm
## Levels: freezing cold cool warm hot
```

```
show_temp(10)
```

```
## [1] cold
## Levels: freezing cold cool warm hot
```

```
show_temp(-10)
```

```
## [1] freezing
## Levels: freezing cold cool warm hot
```

b. Look at help for cut(). Change the call to cut() to handle < instead of <= in the comparisons.

```
show_temp <- function(temp){seq(-10,50,by = 5)
cut(temp, c(-Inf,0,10,20,30,Inf),right = FALSE,
labels = c("freezing", "cold", "cool", "warm", "hot"))}
```

What is the other chief advantage of cut() for this problem? (Hint: what happens if you have many values in temp?)

Two advantages of using cut is that it works on vectors, whereas if only works on a single value, and that to change comparisons I only needed to change the argument to right, but I would have had to change four operators in the if expression.

3. Using piping %>%, sample from the vector 1:10 1000 times with replacement, calculate the resulting sampled vector's mean, then exponentiate that mean.

```
set.seed(123)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(magrittr)
library(knitr)
sample(c(1:10),1000,TRUE) %>% mean() %>%exp()
```

```
## [1] 298.2703
```

4. EXTRA CREDIT (2 Pts) Write a function of an input integer n to create and print the specified matrix with column sums shown at the bottom and select the specified values out of it as below in a.,b., and c. Print the output from the function with inputs n = 5 and then with n = 10.

Hint. Look at help for the following functions you may find useful: seq(), outer(), colSums(),rbind(),print(),which() (with the arr.ind argument set to TRUE), nrow(), and as_tibble()

Print the output from the function with inputs n = 5 and then with n = 10.

```
Matrix_A <- function(n){
  matrix_a <- matrix(data = NA, nrow = n, ncol = n)
  seq_a <- seq(1, n, by = 1)

  for(i in 1:n) {
```

```
    matrix_a[i,] <- 1/(seq_a+(i-1))
  }
  colsum_hw <- colSums(matrix_a)
  matr_colsum <- rbind(matrix_a, colsum_hw)

  return(matr_colsum)
  }

Matrix_A(5)
```

Calculate the sums of each column

```
##               [,1]      [,2]      [,3]      [,4]      [,5]
##          1.0000000 0.5000000 0.3333333 0.2500000 0.2000000
##          0.5000000 0.3333333 0.2500000 0.2000000 0.1666667
##          0.3333333 0.2500000 0.2000000 0.1666667 0.1428571
##          0.2500000 0.2000000 0.1666667 0.1428571 0.1250000
##          0.2000000 0.1666667 0.1428571 0.1250000 0.1111111
## colsum_hw 2.2833333 1.4500000 1.0928571 0.8845238 0.7456349
```

```
Matrix_A(10)
```

```
##               [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
##          1.0000000 0.50000000 0.33333333 0.25000000 0.20000000 0.16666667
##          0.5000000 0.33333333 0.25000000 0.20000000 0.16666667 0.14285714
##          0.3333333 0.25000000 0.20000000 0.16666667 0.14285714 0.12500000
##          0.2500000 0.20000000 0.16666667 0.14285714 0.12500000 0.11111111
##          0.2000000 0.16666667 0.14285714 0.12500000 0.11111111 0.10000000
##          0.1666667 0.14285714 0.12500000 0.11111111 0.10000000 0.09090909
##          0.1428571 0.12500000 0.11111111 0.10000000 0.09090909 0.08333333
##          0.1250000 0.11111111 0.10000000 0.09090909 0.08333333 0.07692308
##          0.1111111 0.10000000 0.09090909 0.08333333 0.07692308 0.07142857
##          0.1000000 0.09090909 0.08333333 0.07692308 0.07142857 0.06666667
## colsum_hw 2.9289683 2.01987734 1.60321068 1.34680042 1.16822899 1.03489566
##                [,7]       [,8]       [,9]      [,10]
##          0.14285714 0.12500000 0.11111111 0.10000000
##          0.12500000 0.11111111 0.10000000 0.09090909
##          0.11111111 0.10000000 0.09090909 0.08333333
##          0.10000000 0.09090909 0.08333333 0.07692308
##          0.09090909 0.08333333 0.07692308 0.07142857
##          0.08333333 0.07692308 0.07142857 0.06666667
##          0.07692308 0.07142857 0.06666667 0.06250000
##          0.07142857 0.06666667 0.06250000 0.05882353
##          0.06666667 0.06250000 0.05882353 0.05555556
##          0.06250000 0.05882353 0.05555556 0.05263158
## colsum_hw 0.93072899 0.84669538 0.77725094 0.71877140
```

Find all entries in the matrix A between 0.11 and 0.15.

```
Matrix_B <- function(n){
  matrix_a <- matrix(data = NA, nrow = n, ncol = n)
  seq_a <- seq(1, n, by = 1)

  for(i in 1:n) {
    matrix_a[i,] <- 1/(seq_a+(i-1))
  }

  find_entry <- which(matrix_a < 0.15 & matrix_a > 0.11, arr.ind = T)
  my_tibble <- as_tibble(cbind(find_entry, matrix_a[find_entry]))

  return(my_tibble)
  }

Matrix_B(5)
```

Print the tibble.

```
## Warning: `as_tibble.matrix()` requires a matrix with column names or a `.name_repair` argument. Using
## This warning is displayed once per session.
```

```
## # A tibble: 6 x 3
##     row   col    V3
##   <dbl> <dbl> <dbl>
## 1     5     3 0.143
## 2     4     4 0.143
## 3     5     4 0.125
## 4     3     5 0.143
## 5     4     5 0.125
## 6     5     5 0.111
```