HW_2_Team_GA

2022-09-16

Instructions

- After completing the questions, upload both the .RMD and PDF files to Bb.
- Learning Outcomes:
- Create functions
- Create and source scripts
- Employ piping
- Employ logicals
- Employ conditionals.

1. Implement a fizzbuzz() function

`1. Create a new function which takes a single number as input. Use logicals and conditionals as well as`

- Requirements

– If the input 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(single_number){
  if(single_number%%5==0 & single_number%%3==0)
    return("fizzbuzz")
  else if(single_number%%5==0)
    return("buzz")
  else if (single_number%%3==0)
    return("fizz")
  else
    return(single_number)}
```

`a. Design your function. Write out in words in a text chunk the steps your function will need to accom`

1. use %% == 0, so we can set the divisor.
2. assign the function to take "single_number"
3. make sure all arguments are in {} in the right position.
4. set the number divisible 3 and 5 to retunr "fizzbuzz"
5. set the number divisible 3 to return "fizz"
6. set the number divisible 5 to return "buzz"
7. otherwise, I set to retunr the "single_number" itself

8. use else if function

    b. Write R code in a code chunk to implement your steps using a variable x. Test it with x having values of of 3, 5, 15, and 16.

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

```
fizzbuzz(3)
```

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

```
fizzbuzz(5)
```

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

```
fizzbuzz(15)
```

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

```
fizzbuzz(16)
```

```
## [1] 16
```

c. Once the code is working, then copy it to a new code chunk and turn it into function with input argu

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

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

```
fizzbuzz(3)
```

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

```
fizzbuzz(5)
```

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

```
fizzbuzz(15)
```

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

```
fizzbuzz(2)
```

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

e. Update your function to include error checking

- Ensure the input is both numeric and a single value - not a vector.

- 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.

```
fizzbuzz <- function(x) {
  stopifnot(length(x) == 1) # checking if x is single input, if not error
  stopifnot(is.numeric(x)) # Checking if x is numeric, if not error
 if(x%%5==0 & x%%3==0)
    return("fizzbuzz")
  else if(x%%5==0)
    return("buzz")
  else if (x%%3==0)
    return("fizz")
  else
    return (as.character(x))}

fizzbuzz('cat')
```

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

```
fizzbuzz <- function(x) {
  stopifnot(length(x) == 1) # checking if x is single input, if not error
  stopifnot(is.numeric(x)) # Checking if x is numeric, if not error
 if(x%%5==0 & x%%3==0)
    return("fizzbuzz")
  else if(x%%5==0)
    return("buzz")
  else if (x%%3==0)
    return("fizz")
  else
    return (as.character(x))}

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

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

f. Complete your function by inserting and completing Roxygen comments in the code chunk, above the function, to document the function.Include the following elements: title, description, usage or syntax, arguments (the params), and return value.

```
# Implement a fizzbuzz() function which takes a single number as input
# x: 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(x) {
  stopifnot(length(x) == 1) # checking if x is single input, if not error
  stopifnot(is.numeric(x)) # Checking if x is numeric, if not error
 if(x%%5==0 & x%%3==0)
    return("fizzbuzz")
  else if(x%%5==0)
    return("buzz")
```

```
  else if (x%%3==0)
    return("fizz")
  else
    return (as.character(x))}
```

g. Create a script out of your fizzbuzz() function

• Copy and paste the code from your working function into a new .R file and save in the R directory with the file name fizzbuzz_s.R

• Rename the function to fizzbuzz_s

• Use the following code in a code chunk to show your code

– cat(readr::read_file(“./R/fizzbuzz_s.R”))

– Adjust the relative path as necessary

• Write code in a new code chunk in your original homework file to source the fizzbuzz_s() function

• Run the function in your homework .Rmd file to show the results with the values 35, 18, 45, and -1

```
cat(readr::read_file("./fizzbuzz_s.R"))
```

```
## # Implement a fizzbuzz() function which takes a single number as input
## # x: 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_s <- function(x) {
##   stopifnot(length(x) == 1) # checking if x is single input, if not error
##   stopifnot(is.numeric(x)) # Checking if x is numeric, if not error
##   if(x%%5==0 & x%%3==0)
##     return("fizzbuzz")
##   else if(x%%5==0)
##     return("buzz")
##   else if (x%%3==0)
##     return("fizz")
##   else
##     return(x)
## }
```

```
source("~/Desktop/Desktop - FENTAW's MacBook Air/American_U/R_programming/fizzbuzz_s.R")
fizzbuzz_s(35)
```

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

```
fizzbuzz_s(18)
```

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

```
fizzbuzz_s(45)
```

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

```
fizzbuzz_s(-1)
```

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

2. Create a new cut() function

1. 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.

```
# #temp=c(-Inf,0,10,20,30,Inf)
# if (temp <= 0)
#   return("freezing")
# else (temp <= 10)
#   return("cold")
# else (temp <= 20)
#   return("cool")
# } else if (temp <= 30) {
# "warm"
# } else {
# "hot"}
```

```
temp_type <- 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"))}
```

```
temp_type(31)
```

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

```
temp_type(30)
```

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

```
temp_type(10)
```

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

```
temp_type(-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.

```
?cut()

# cut(x, ...)
#
# ## Default S3 method:
# cut(x, breaks, labels = NULL,
#     include.lowest = FALSE, right = TRUE, dig.lab = 3,
#     ordered_result = FALSE, ...)
# cut divides the range of x into intervals and codes the values in x according to which interval they
temp_type <- 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"))}

temp_type(30)
```

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

c. What is the other chief advantage of the cut() method for this problem? (Hint: what happens if you ha

```
# By using cut, I can works on vectors,and
# To change comparisons I only needed to change the argument to "right"
```

3. Using the Forward Pipe

1. Using the forward pipe %>%,

- Sample from the vector 1:10 1000 times with replacement,
- Calculate the resulting sampled vector's mean, then
- Exponent 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. Calculate a proportion

- Select a random sample of 100 normally distributed values with mean 10 and variance of 3.
- Calculate the proportion greater than 12.

```
library(magrittr)
sd1=sqrt(3)
x<- rnorm(100,mean=10,sd=sd1)
proportion_greater12= mean(x>12) # use the mean function to roll this up to a proportion
proportion_greater12
```

```
## [1] 0.15
```

```
### Note: every time I run the chunk, the proportion output changed because of rnorm.
```

```
### This is the other way I've tried,
sd1=sqrt(3) # sd is a sqrt of variance.
x<- rnorm(100,mean=10,sd=sd1)
pnorm(12,mean=10,sd=sd1, lower.tail = TRUE)
```

```
## [1] 0.8758935
```

5. Logical Comparisons and Subsetting

- Create the values:

– x <- c(TRUE, FALSE, TRUE, TRUE)

– y <- c(FALSE, FALSE, TRUE, FALSE)

– z <- NA

- What are the results of the following:

– x & y

```
x <- c(TRUE, FALSE, TRUE, TRUE)

y <- c(FALSE, FALSE, TRUE, FALSE)

z <- NA

x&y
```

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

– x & z

```
x&z
```

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

– !(x | y)

```
!(x | y)
```

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

– x | y

```
x | y
```

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

– y | z

```
y | z
```

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

– x[y]

```
x[y]
```

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

– y[x]

```
y[x]
```

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

– x[x|y]

```
x[x|y]
```

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