Homework_02

Sunday Okechukwu

2022-09-20

1. Create a new function which takes a single number as input. Use logicals and conditionals as well as the operator %% as required.

a. Design your function. Write out in words in your .Rmd file, the logic of the steps your function will need to accomplish to convert the input number into the specified output.

Answer

Let x be any number ()

1. If x is divisible by both 3 and 5, then output is "buzzfizz"
2. However, if x is divisible by 3, then output is "buzz"
3. If x is divisible by 5, output is "fizz"
4. If the above conditions fail, then output is x

b. Write R code (Not a function) in a code chunk to implement and test the logic of each of your steps individually on a variable x. Test it with x having values of of 3, 5, 15, and 16.

```
#1. If x is divisible by both 3 and 5, then output is "buzzfizz"

x = 3

if( x %% 3 == 0 & x %% 5 == 0) {
  message <- "buzzfizz"

} else if (x %% 3 == 0) {
  message <- "buzz"
} else if (x %% 5 == 0) {
  message <- "fizz"
} else {
  message <- x
}
message
```

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

```
x = 5

if( x %% 3 == 0 & x %% 5 == 0) {
  message <- "buzzfizz"
```

```
} else if (x %% 3 == 0) {
  message <- "buzz"
} else if (x %% 5 == 0) {
  message <- "fizz"
} else {
  message <- x
}
message
```

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

```
x = 15

if( x %% 3 == 0 & x %% 5 == 0) {
  message <- "buzzfizz"

} else if (x %% 3 == 0) {
  message <- "buzz"
} else if (x %% 5 == 0) {
  message <- "fizz"
} else {
  message <- x
}
message
```

```
## [1] "buzzfizz"
```

```
x = 16

if( x %% 3 == 0 & x %% 5 == 0) {
  message <- "buzzfizz"

} else if (x %% 3 == 0) {
  message <- "buzz"
} else if (x %% 5 == 0) {
  message <- "fizz"
} else {
  message <- x
}
message
```

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

c. Once the individual steps are working, create a new code chunk and copy the code using variable x to the new code chunk and turn it into function with the input argument of x. Make sure x has a default argument

```
buzzfizz <- function(x = 4){
  if( x %% 3 == 0 & x %% 5 == 0) {
  message <- "buzzfizz"
```

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

buzzfizz()
```

```
## [1] 4
```

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

```
buzzfizz(3)
```

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

```
buzzfizz(5)
```

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

```
buzzfizz(15)
```

```
## [1] "buzzfizz"
```

```
buzzfizz(2)
```

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

e. Copy and paste your last function into a new code chunk. Update your function to include error checking.

- Include checks to ensure the input is both numeric and a single value, not a vector.
- Test it on inputs cat, and c(1,5).
- 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.

```
buzzfizz <- function(x = 4){
 stopifnot(length(x) == 1,
                is.numeric(x),
                is.vector(x))

  if( x %% 3 == 0 & x %% 5 == 0) {
  message <- "buzzfizz"

} else if (x %% 3 == 0) {
```

```
  message <- "buzz"
} else if (x %% 5 == 0) {
  message <- "fizz"
} else {
  message <- x
}
  return(message)
}
```

```
buzzfizz("cat")
```

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

```
buzzfizz(c(1,5))
```

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

f. Copy and paste your function with error checking into a new code chunk. Complete your function by inserting and completing Roxygen2 comments in the code chunk, above the function, to document the function.

- Include the following elements: title, description, usage, arguments (the params), and return value.
- Include three examples.

```
#' Title: Write a buzzfizz() function
#'
#' `Description`
#'
#' This function takes a number as input, if the input number is divisible by five, it returns "fizz".I
#'
#' `Usage`
#' buzzfizz(x)
#'
#' @param x An input number x
#'
#' @return If the input number is divisible by five, it returns "fizz".If it's divisible by three, it r
#' @export
#'
#' @examples #
#' buzzfizz(2)
#' buzzfizz(3)
#' buzzfizz(15)
#'
buzzfizz <- function(x = 4){
 stopifnot(length(x) == 1,
                is.numeric(x),
                is.vector(x))

  if( x %% 3 == 0 & x %% 5 == 0) {
  message <- "buzzfizz"
```

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

h. Create a new chunk in your original .Rmd file.

- Use the following line of code but replace the my_path with your relative path to the R script file (including the file name) – cat(readr::read_file("my_path"))

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

```
##
## #' Title: Write a buzzfizz() function
## #'
## #' Description
## #'
## #' This function takes a number as input, if the input number is divisible by five, it returns "fizz
## #'
## #' `Usage`
## #' buzzfizz(x)
## #'
## #' @param x An input number x
## #'
## #' @return If the input number is divisible by five, it returns "fizz".If it's divisible by three, i
## #' @export
## #'
## #' @examples #
## #' buzzfizz_s(2)
## #' buzzfizz_s(3)
## #' buzzfizz_s(15)
## #'
## buzzfizz_s <- function(x = 4){
##   stopifnot(length(x) == 1,
##             is.numeric(x),
##             is.vector(x))
##
##   if( x %% 3 == 0 & x %% 5 == 0) {
##     message <- "buzzfizz"
##
##   } else if (x %% 3 == 0) {
##     message <- "buzz"
##   } else if (x %% 5 == 0) {
##     message <- "fizz"
##   } else {
##     message <- x
##   }
```

```
##   return(message)
## }
```

- Create a new code chunk in your original ,Rmd and source the buzzfizz_s() function.
- Enter code in that chunk to run the function with the values 35, 18, 45, and -1

```
source("./R/buzzfizz_s.R")

buzzfizz(35)
```

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

```
buzzfizz(18)
```

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

```
buzzfizz(45)
```

```
## [1] "buzzfizz"
```

```
buzzfizz(-1)
```

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

2. Write a function called newcut() that uses the Base R function cut() to simplify this set of nested if-else statements?

a. Show the output for inputs: 31, 40, 60, 75, and 90.

```
newcut <- function(x) {
categories <- cut(x, breaks = c(-Inf, 32, 45, 60, 80, Inf),
                  labels = c("freezing", "cold", "cool","nice", "hot"))
return (data.frame(x, categories))

}

newcut(c(31, 45, 60, 80, 90))
```

```
##    x categories
## 1 31   freezing
## 2 45       cold
## 3 60       cool
## 4 80       nice
## 5 90        hot
```

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

- Call this function cutleft().

```
cutleft <- function(x) {
  categories <- cut(x, breaks = c(-Inf, 32, 45, 60, 80, Inf),
labels = c("freezing", "cold", "cool","nice", "hot"), right = FALSE)
return (data.frame(x, categories))

}

cutleft(c(31, 45, 60, 80, 90))
```

```
##    x categories
## 1 31   freezing
## 2 45       cool
## 3 60       nice
## 4 80        hot
## 5 90        hot
```

- Test both newcut() and cutleft() on the vector with values 31, 45, 60, 80, 90
- Compare the outputs.

Answer

By default, the argument right is set to TRUE, so the intervals are opened on the left and closed on the right (x, y]. However, if I set right = FALSE, the intervals will be closed on the left and open on the right [x, y)

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

Answer

While the code with conditionals always works, it does suffer from pattern repetition. When temp has many values, the conditional statements are repeated many times.And when writing code you don't want to repeat yourself a.ka. DRY.

3 Using a Forward Pipe 1. Create a code chunk and set the random number seed to 1 using set.seed(1).

- In the same code chunk, use a forward pipe, %>% or |>, to do the following steps without intermediate variables: – Use sample() to sample from the vector 1:10 1000 times with replacement – Calculate the mean of the resulting sampled vector, then – Exponentiate that mean.

```
set.seed(1)
sample(1:10, 1000, replace = TRUE) %>%
  mean() %>%
  exp() ->
  result
```

```
## Error in sample(1:10, 1000, replace = TRUE) %>% mean() %>% exp(): could not find function "%>%"
```

```
result
```

```
## Error in eval(expr, envir, enclos): object 'result' not found
```

4 Calculate a Proportion * Set the seed to 1 and select a random sample of 100 normally distributed values with mean 10 and variance of 3. * Calculate the proportion greater than 12.

```
set.seed(1)
x <- sample(rnorm(100, mean = 10, sd = sqrt(3)))
length(x[x > 12]) / length(x)
```

```
## [1] 0.12
```

5 Logical Comparisons and Subsetting

```
x <- c(TRUE, FALSE, TRUE, TRUE)
y <- c(FALSE, FALSE, TRUE, FALSE)
z <- NA

x & y
```

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

```
x & z
```

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

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

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

```
x | y
```

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

```
y | z
```

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

```
x[y]
```

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

```
y[x]
```

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

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

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