HW 2

Enter your name and EID here:

Jongho Yoo - jy23294

You will submit this homework assignment as a pdf file on Gradescope.

For all questions, include the R commands/functions that you used to find your answer (show R chunk). Answers without supporting code will not receive credit. Write full sentences to describe your findings.

The goal of this assignment is to encode your name (or any other message) using a *cipher* function: We want to replace each letter of a given character vector with the letter of the alphabet that is *k* positions after it in the alphabet. For example, if the letter was `a` and `k = 3`, we would replace it with `d`. We will also want it to loop around, so if the letter was `y` and `k = 3`, we'd replace it with `b`. For example, with `k = 3`, the word `dog` would become `grj`. Let's take it step by step.

Question 1: (2 pts)

Type the word `letters` into the R chunk below. What does this predefined object in R contain? What is this object's data type/class? How many elements does it contain? *Include base R commands used to answer all three questions.*

```
# Code to display letters object, then find its class and length
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
class(letters)
```

```
## [1] "character"
```

```
length(letters)
```

```
## [1] 26
```

This object contains all the lowercase letters of the English alphabet. This object is a character class. It contains 26 elements.

Question 2: (2 pts)

First, here is the code to split a word into a vector containing each letter.

```
test <- unlist(strsplit("test", split = ""))
# Note: the function strsplit() returns a list, use unlist() to return a vector
```

Remember that `A %in% B` returns a vector of the positions of matches of an object A in an object B (Worksheet 2):

```
letters %in% test
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE
```

How many elements are false in the resulting logical vector? *Include base R commands used to answer this question (recall: TRUE is equivalent to the value 1 and FALSE is 0.*

```
# Sum function to find # of true and false values
sum(letters %in% test)
```

```
## [1] 3
```

23 of the elements in the resulting logical vector are false. Since 3 is true, and there are 26 elements, the 23 remaining are false.

Question 3: (2 pts)

Another function that will be useful is `which()`: it takes a logical vector as an input and returns the indices/positions that are TRUE. For example, run the following code:

```
# Note: T is shorthand for TRUE, F for FALSE
which(c(F,T,F,T,F,T))
```

```
## [1] 2 4 6
```

The output tells you that elements in position 2, 4, and 6 are true. Now, use the `which` function, along with `%in%` and `letters`, to find which positions in the alphabet the letters in the name `layla` occupy (saved as an object called `name_v` below). Would that combination of functions alone work to encode a name? Why/Why not?

```
# Define name as a vector
name_v <- unlist(strsplit("layla", split = ""))

# Find which positions in "letters" the letters of "layla" occupies
which(letters %in% name_v)
```

```
## [1]  1 12 25
```

Positions: 1, 12, 25. It does not work alone to encode a name because it will not return the positions of duplicating numbers.

Question 4: (2 pts)

How can we avoid this? For example, we can test each letter one at a time in their correct order! One approach would be to use a *for loop*. Write a for loop that goes through each element of the character vector `name_v` (i.e., each letter in `c("l","a","y","la")`) one at a time, finds its position in the alphabet, and saves each position in a vector called `positions`. Confirm that the positions are correct by using `positions` as an index to find the corresponding letters in the object `letters`.

For example, the name ***ali*** *would give you the positions 1,12, and 9. You can grab the letters in those positions by doing* `letters[c(1, 12, 9)]`.

```
# Initialize the vector positions
positions <- c()

# for loop to iterate through each letter in name_v
for(i in name_v) {
  # find the position of each letter in name_v in the alphabet, and save it into
  # the positions vector
  positions <- c(positions, which(letters %in% i))
}

# call positions vector to find the corresponding letters in the alphabet
positions
```

```
## [1] 12  1 25 12  1
```

Question 5: (2 pts)

Let's encode the name `layla`! Shift all the `positions` by 1 and index `letters` to obtain the encoded name. Is the encoded name a real name?

```
# your code goes below (make sure to edit comment)
name <- unlist(strsplit("layla", split = ""))

for (i in name) {
  positions <- which(letters %in% i) + 1
  print(letters[positions])
}
```

```
## [1] "m"
## [1] "b"
## [1] "z"
## [1] "m"
## [1] "b"
```

No. The resulting name was mbzmb, which is not a real name.

Question 6: (2 pts)

Now, what if you would like to get the positions in your name? Or any other name? We would have to repeat questions 2-5... Instead let's write a function to 1) split a name as a vector (i.e., a character vector whose elements contain single letters), 2) initialize the positions, and 3) report each position in a vector `positions` with a for loop for each new name we would like to encode. The function should take a `name` (for example, "layla") as the input and return the alphabetical positions each of those letters occupy. Call the function `get_position`. Once you have defined it, test it out with "layla". Did you get all positions?

```
# define function
get_position <- function (name_input) {
  name <- unlist(strsplit(name_input, split = "")) # encode name
  positions <- c() # initialize vector
  for (i in name) { # for loop to iterate through each element in the name object
    positions <- c(positions, which(letters %in% i)) # save position of letter
    # into positions vector
  }
  positions # display resulting positions of each letter
}

# call function to return alphabetical positions of each letter
get_position("layla")
```

```
## [1] 12  1 25 12  1
```

Yes, I got all positions

Question 7: (2 pts)

What happens when we shift the positions past `z`, the 26th and final letter of the alphabet? Shifting the positions in `layla` up by `k = 2` should give `ncanc`, but since there is no 27th element of letters, it will return `NA` instead of `a`. Try it in the code chunk below.

```
letters[get_position(name_v) + 2]
```

```
## [1] "n" "c" NA  "n" "c"
```

```
# returns NA instead of a :(
```

How do we make it loop around so that `z` shifted up 1 becomes `a`? In other words, how can we make 27 become 1, 28 become 2, 29 becomes 3, etc.? We will use a mathematical operator called modulo `%%` (which tells you the remainder when you divide one number by another). Try running the code below, `27 %% 26` (pronounced "27 modulo 26"). It returns 1, the remainder when the number on the left (27) is divided by the number on the right (26).

```
27 %% 26
```

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

We just need our shifted positions *modulo* 26. You can do this with `(positions + k) %% 26`. One last minor issue: `26 %% 26` is 0 (or any multiple of 26 %% 26 is 0) but we want it to return 26 (i.e., the letter `z`). We can fix this issue by using `ifelse` for example. Test if `positions + k %% 26` is 0: if it is, use 26, if it is not use `positions + k %% 26`. Use your function `get_position()` and the fix with modulo `%%` in `ifelse()` to encode the word `layla` by shifting every letter `k = 2` positions forward correctly. Is the encoded name a real name?

```
# define function with name and k as inputs
get_position <- function (name_input, k) {
  name <- unlist(strsplit(name_input, split = "")) # encode name
  positions <- c() # initialize vector
  for (i in name) { # for loop to iterate through each element in name object
    positions <- c(positions, which(letters %in% i)) # save position of each letter
    # into the vector
  }
  newPositions <- ifelse((positions + k) %% 26 == 0, 26, (positions + k) %% 26) # ifelse
  # statement to loop back to a when exceeding z
  letters[newPositions] # index letters object with positions of each letter
  # to get corresponding name
}

get_position("layla", 2)
```

```
## [1] "n" "c" "a" "n" "c"
```

No. The resulting name was "ncanc" which is not a real name.

Question 8: (2 pts)

Putting it all together: Write a function that incorporates all the work you have done to achieve the encoding task. Name the function `cipher`. This function should take two arguments: a `name` (a string) and how many positions to shift (`k`). Fill in the code below with what you have been using above. Check your code with `layla` shifted by 2 positions and test your code with your own name with the shift of your choice! Is the encoded name a real name?

```
# define function with name and k as inputs
cipher <- function(name, k) {
  name <- unlist(strsplit(name, split = "")) # encode name
  positions <- c() # initialize vector
  for (i in name) { # for loop to iterate through each element in name object
    positions <- c(positions, which(letters %in% i)) # save position of each letter
    # into the vector
  }
  newPositions <- ifelse((positions + k) %% 26 == 0, 26, (positions + k) %% 26) # ifelse
  # statement to loop back to a when exceeding z
```

```
  letters[newPositions] # index letters object with positions of each letter
  # to get corresponding name
}

# check
cipher("layla", 2)
```

```
## [1] "n" "c" "a" "n" "c"
```

```
# test your name!
cipher("steven", 20)
```

```
## [1] "m" "n" "y" "p" "y" "h"
```

No, I the encoded name is not a real name.

Question 9: (2 pts)

A less guided question... You were given the code `oldp`. Can you decipher the code and find the name hidden behind it?

```
# for loop to test k values 1:26
for (i in 1:26) {
  print(i) # print k value corresponding to each name
  print(cipher("oldp", i)) # call function to print each name
}
```

```
## [1] 1
## [1] "p" "m" "e" "q"
## [1] 2
## [1] "q" "n" "f" "r"
## [1] 3
## [1] "r" "o" "g" "s"
## [1] 4
## [1] "s" "p" "h" "t"
## [1] 5
## [1] "t" "q" "i" "u"
## [1] 6
## [1] "u" "r" "j" "v"
## [1] 7
## [1] "v" "s" "k" "w"
## [1] 8
## [1] "w" "t" "l" "x"
## [1] 9
## [1] "x" "u" "m" "y"
## [1] 10
## [1] "y" "v" "n" "z"
## [1] 11
## [1] "z" "w" "o" "a"
```

```
## [1] 12
## [1] "a" "x" "p" "b"
## [1] 13
## [1] "b" "y" "q" "c"
## [1] 14
## [1] "c" "z" "r" "d"
## [1] 15
## [1] "d" "a" "s" "e"
## [1] 16
## [1] "e" "b" "t" "f"
## [1] 17
## [1] "f" "c" "u" "g"
## [1] 18
## [1] "g" "d" "v" "h"
## [1] 19
## [1] "h" "e" "w" "i"
## [1] 20
## [1] "i" "f" "x" "j"
## [1] 21
## [1] "j" "g" "y" "k"
## [1] 22
## [1] "k" "h" "z" "l"
## [1] 23
## [1] "l" "i" "a" "m"
## [1] 24
## [1] "m" "j" "b" "n"
## [1] 25
## [1] "n" "k" "c" "o"
## [1] 26
## [1] "o" "l" "d" "p"
```

The hidden name is Liam.

Formatting: (2 pts)

Comment your code, write full sentences, and knit your file!

```
##                                                                                                                                  sysn
##                                                                                                                                "Darw
##                                                                                                                                 relo
##                                                                                                                                "21.3
##                                                                                                                                 vers
## "Darwin Kernel Version 21.3.0: Wed Jan  5 21:37:58 PST 2022; root:xnu-8019.80.24~20/RELEASE_ARM64_T8
##                                                                                                                               noden
##                                                                                                                 "Stevens-MacBook-Pro-2.lo
##                                                                                                                                  mach
##                                                                                                                                 "arm
##                                                                                                                                   lo
```

```
##                                                                              "re
##                                                                                 "
##                                                                           "steven
##                                                                        effective_
##                                                                           "steven
```