Name: Patrick Norton

PERMID: 4149092

Date: December 3, 2023

Function 1 - fun_sequence - :

Function Description

The following sequence takes an integer 'n' as input and prints the first 15 values of a sequence generated by a specific algorithm. The sequence starts with 'n' and follows the rules: a) If the current number is even, the next number is the current number divided by two. b) If the current number is odd, the next number is the current number times 3 plus 1. c) If the computed number is equal to 1, all subsequent numbers in the sequence are set to 1.

```
PERMID <- "4149092" #Type your PERMID with the quotation marks
PERMID <- as.numeric(gsub("\\D", "", PERMID)) #Don't touch
set.seed(PERMID) #Don't touch

fun_sequence <- function(n) {
  sequence <- numeric(15)

  for (i in 1:15) {
    sequence[i] <- n

    if (n %% 2 == 0) {
      n <- n / 2
    } else {
      n <- 3 * n + 1
    }

    if (n == 1) {
      for (j in (i + 1):15) {
        sequence[j] <- 1
      }
      break
    }
  }

  cat("##", sequence, "\n")
}
```

Testing fun_sequence: Test 1: n = 2

Testing to see if '2' will output to 1 and remain at 1

```
# Test 1
fun_sequence(2)
```

```
## ## 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Testing fun_sequence: Test 2: n = 12

The values are dividing by 2 until an odd number is present, then following the description above, 3 will be tripled and added by 1 and the process repeats until 2 is divided by 2 equaling 1.

```r
# Test 2
fun_sequence(12)
```

```
## ## 12 6 3 10 5 16 8 4 2 1 1 1 1 1 1
```

Testing fun_sequence: Test 3: n = 27

This test should never reach the value 1 because the values are getting exponentially bigger.

```r
# Test 3
fun_sequence(27)
```

```
## ## 27 82 41 124 62 31 94 47 142 71 214 107 322 161 484
```

Function 2 - prev_letter - :

Function Description

This R function takes a string as input and returns a new string in which each letter is replaced by its predecessor in the alphabet. The function prints a sentence indicating the input and output for clarity. In addition, it also converts the input string to uppercase to ensure uniform processing, iterates through each character in the input string, and determines its predecessor in the alphabet. If the letter is 'A,' it wraps around to 'Z.' Moreover, it maintains non-letter characters as they are and finally prints a sentence indicating the input and output for the provided string.

```r
prev_letter <- function(input_str) {
  input_str <- toupper(input_str)  # Convert input to uppercase
  output_str <- ""

  for (char in strsplit(input_str, NULL)[[1]]) {
    if (char %in% LETTERS) {
      prev_char <- ifelse(char == "A", "Z", intToUtf8(utf8ToInt(char) - 1))
      output_str <- paste(output_str, prev_char, sep = "")
    } else {
      output_str <- paste(output_str, char, sep = "")
    }
  }

  cat("The input is:", input_str, "\n")
  cat("The output is:", output_str, "\n\n")

  return(output_str)
}
```

Testing prev_letter: Test 1: "ABCD"

The letter 'A' wrapped around to 'Z' and started the alphabet over.

```r
# Test 1
prev_letter("ABCD")
```

```
## The input is: ABCD
## The output is: ZABC
```

```
## [1] "ZABC"
```

Testing prev_letter: Test 2: "GOOD"

Each letter was replaced by the preceding letter in the alphabet and is still in accordance to the pattern of the word "GOOD".

```
# Test 2
prev_letter("GOOD")
```

```
## The input is: GOOD
## The output is: FNNC
```

```
## [1] "FNNC"
```

Testing prev_letter: Test 3: "BBBBBB"

Every letter should have returned to 'A' because 'A' precedes 'B' in the alphabet.

```
# Test 3
prev_letter("BBBBBB")
```

```
## The input is: BBBBBB
## The output is: AAAAAA
```

```
## [1] "AAAAAA"
```

Function 3 - simulator - :

Function Description

The **simulator** function is designed to take three inputs: the number of faces on the die **num_faces**, the number of draws to simulate **num_draws**, and the probabilities of each face showing up **probabilities**. The function then performs die rolls based on the given probabilities and returns a tibble with four columns: **value**, **n**, **share**, and **probability**.

```
library(tidyr)
library(tidyverse)
library(ggplot2)

simulator <- function(num_faces, num_draws, probabilities) {
  if (num_faces <= 0 || num_draws <= 0 || sum(probabilities) != 1) {
    stop("Invalid input")
  }

  die_rolls <- sample(1:num_faces, num_draws, replace = TRUE, prob = probabilities)

  result_table <- table(die_rolls)

  share_column <- prop.table(result_table)
  probability_column <- probabilities

  result_tibble <- tibble(
    value = as.numeric(names(result_table)),
    n = as.numeric(result_table),
    share = share_column,
    probability = probability_column
  )
```

```
  print(result_tibble)

  barplot_data <- result_tibble %>%
    mutate(value = as.factor(value)) %>%
    gather(key = "type", value = "percentage", probability, share)
}
```

Testing simulator: 1000 draws for n = 5

The function is tested with 10,000 draws for a 5-faced die with custom probabilities.

```
num_faces <- 5
num_draws <- 10000
probabilities <- c(0.1, 0.2, 0.3, 0.2, 0.2)

simulator_result <- simulator(num_faces, num_draws, probabilities)
```

```
## # A tibble: 5 x 4
##   value     n share       probability
##   <dbl> <dbl> <table[1d]>       <dbl>
## 1     1   998 0.0998              0.1
## 2     2  1976 0.1976              0.2
## 3     3  3001 0.3001              0.3
## 4     4  2021 0.2021              0.2
## 5     5  2004 0.2004              0.2
```

Testing simulator: Barplot

A barplot is generated based on the results obtained from the simulation. The **barplot_data** tibble is prepared using the **gather** function and is then used to create a barplot using **ggplot2**. This barplot is in line with the results of the tibble as the shares of the face value is nearly identical with its probability. Note that the y axis title "Proportion" refers to the likelihood of both the probability and the share of the face value.

```
ggplot(simulator_result, aes(x = value, y = percentage, fill = type, color)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Comparison of Probability and Share for Each Face",
       x = "Die Face",
       y = "Proportion",
       fill = "Type") +
  theme_classic()
```


Comparison of Probability and Share for Each Face
Proportion
0.3
0.2
0.1
0.0
1
2
3
4
5
Die Face
Type
probability
share