Function 1: fun_sequence

The `fun_sequence` function takes an integer `n` as input and generates a sequence based on a specific algorithm. It prints the first 15 values of the sequence as defined below:

- The first number is the input integer `n`.
- Each subsequent number is determined by the following rules:
  - If the current number is even, the next number is the current number divided by two.
  - If the current number is odd, the next number is the current number multiplied by 3 plus 1
  - If the number computed is equal to 1, all subsequent numbers will be 1.
- The function returns a sequence of the first 15 values based on the algorithm described.

Input:

- `n`: An integer representing the starting point of the sequence.

Output:

- A numeric vector containing the first 15 values of the sequence.

Function

```
fun_sequence <- function(n) {
  sequence <- numeric(15)
  sequence[1] <- n

  for (i in 2:15) {
    if (sequence[i - 1] %% 2 == 0) {
      sequence[i] <- sequence[i - 1] / 2
    } else {
      sequence[i] <- sequence[i - 1] * 3 + 1
    }

    if (sequence[i] == 1) {
      sequence[(i + 1):15] <- rep(1, 15 - i)
      break
    }
  }

  return(sequence)
}
```

Tests

Test 1:

```r
fun_sequence(2)
```

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

Test 2:

```r
fun_sequence(12)
```

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

Test 3:

```r
fun_sequence(27)
```

```
##  [1]  27  82  41 124  62  31  94  47 142  71 214 107 322 161 484
```

Function 2: prev_letter

The `prev_letter` function takes a character string as input and returns another character string where each letter is replaced by its preceding letter in the alphabet.

- For example, if the input is "ABCD", the output will be "ZABC". Similarly, if the input is "GOOD", the output will be "FNNC".

The function prints the input string and its corresponding output string.

Input:

- `input_string`: A character string where each character should be a letter.

Output:

- A character string where each letter is replaced by its preceding letter in the alphabet.

Function

```r
prev_letter <- function(word) {
  alphabet <- c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N",
                "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z")

  prev_alphabet <- c("Z", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L",
                     "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y")

  word <- toupper(word)
```

```
  prev_word <- chartr(paste(alphabet, collapse = ""), paste(prev_alphabet, collapse = ""), word)

  input_sentence <- paste("The input is:", word)
  output_sentence <- paste("The output is:", prev_word)

  cat(input_sentence, "\n")
  cat(output_sentence, "\n")
}
```

Tests

Test 1:

```
prev_letter("ABCD")
```

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

Test 2:

```
prev_letter("GOOD")
```

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

Test 3:

```
prev_letter("BBBBBB")
```

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

Function 3: simulator

The `simulator` function simulates dice rolls with a variable number of faces, allowing customization of the number of faces, number of draws, and probabilities for each face.

- It takes three inputs: `num_faces` (number of faces of the die), `num_draws` (number of draws), and `probabilities` (a numeric vector representing the probabilities of each face).
- The function generates random draws based on the specified probabilities and returns a tibble containing columns: value (face number), n (count of each face), share (proportion of each face), and probability (assigned probability for each face).

Input:

- `num_faces`: An integer specifying the number of faces of the die.
- `num_draws`: An integer indicating the number of draws to simulate.
- `probabilities`: A numeric vector representing the probabilities assigned to each face that must add up to 1. The length of this vector should match `num_faces`.

Output:

- A tibble containing information about the simulated dice rolls, including the count of each face, proportion, and assigned probability.

Function

```
library(tidyverse)
library(knitr)

simulator <- function(num_faces, num_draws, probabilities) {
  if (sum(probabilities) != 1) {
    stop("Probabilities must sum up to 1")
  }

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

  counts <- table(draws)
  shares <- prop.table(counts)

  result <- tibble(
    value = as.integer(names(counts)),
    n = as.integer(counts),
    share = shares,
    probability = probabilities
  )

  print(kable(result, caption = "Counts and Shares for Each Face Value"))

  bar_data <- bind_rows(
    tibble(Type = "Probability", Face_Value = 1:num_faces, Probability = probabilities),
    tibble(Type = "Share", Face_Value = 1:num_faces, Share = result$share)
  )

  ggplot(bar_data, aes(x = factor(Face_Value), y = Face_Value, fill = Type)) +
    geom_bar(stat = "identity", position = "dodge") +
    labs(title = "Comparison of Probability and Share for Each Face",
         x = "Face Value", y = "Probability / Share") +
    theme_minimal()
}
```

Tests

```
simulator(num_faces = 5, num_draws = 10000, probabilities = c(0.1, 0.3, 0.2, 0.15, 0.25))
```

```
##
##
## Table: Counts and Shares for Each Face Value
##
```

```
## | value|    n|  share| probability|
## |-----:|----:|------:|-----------:|
## |     1|  979| 0.0979|        0.10|
## |     2| 2997| 0.2997|        0.30|
## |     3| 2022| 0.2022|        0.20|
## |     4| 1478| 0.1478|        0.15|
## |     5| 2524| 0.2524|        0.25|
```