Workout 3 - R Package Roller

Stat 133, Fall 2018, Prof. Sanchez

Due date: Dec-02

The purpose of this assignment is to create an R package that implements functions for simulating rolling an object (e.g. a coin or a die) multiple times.

Here's a list of resources that will help you complete this assignment:

- **Pack YouR Code**: gastonsanchez.com/packyourcode
- **Example package `"cointoss"`**: github.com/gastonstat/cointoss
- **Package development cheat-sheet**: packages-cheatsheet.pdf
- **R Packages**: r-pkgs.had.co.nz

The goal is to program two classes of objects: the `"device"` to be rolled, and the object `"rolls"` (containing the rolls of the `"device"`).

1) Object `"device"`

Because your package will be used to simulate rolling an object, the first thing to do is writing code that allows you to create such an object. In other words, you will have to create an R object of class `"device"`.

The idea is to write a function `device()` that takes two arguments, `sides` and `prob`, in order to return a `"device"` object. Here's an example of the default call to `device()`:

```
# default call: creates a coin device
fair_coin <- device()
fair_coin
```

```
## object "device"
##
##   side prob
## 1    1  0.5
## 2    2  0.5
```

In order to create an object `"device"` write:

- a constructor function `device()` that creates a *fair* coin by default. This function should have two arguments:
  - `sides`: vector of $k \geq 2$ elements, by default numbers 1 and 2.
  - `prob`: vector of probabilities for each side (all equal to 1/2 by default).

  - arguments `sides` and `prob` must be of the same length.

- an auxiliary function `check_sides()`, called by `device()`, that checks the validity of the argument `sides`. Basically, `sides` must be a vector with more than one element, and it cannot contain duplicated elements.
- an auxiliary function `check_prob()`, called by `device()`, that checks the validity of the argument `prob`. Basically, `prob` must be a numeric vector with valid probability values: more than one lement, containing numbers between 0 and 1, such that their sum equals 1.
- a `"print"` method for `"device"` objects, that displays the class of the object, and a tabular display of the sides and the associated probabilities (see examples below).
- an `is.device()` function to test if a given object `x` is of class `"device"`. Returns `TRUE` if input is of class `"device"`, otherwise `FALSE`.

Here are a couple of examples of various ways to call `device()`:

```
# die with non-standard sides
weird_die <- device(
  sides = c('i', 'ii', 'iii', 'iv'),
  prob = rep(1/4, 4))

weird_die
```

```
## object "device"
##
##   side prob
## 1    i 0.25
## 2   ii 0.25
## 3  iii 0.25
## 4   iv 0.25
```

```
is.device(weird_die)
```

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

```
# create a loaded die
loaded_die <- device(
  sides = 1:6,
  prob = c(0.075, 0.1, 0.125, 0.15, 0.20, 0.35))

loaded_die
```

```
## object "device"
##
##   side  prob
```

```
## 1    1 0.075
## 2    2 0.100
## 3    3 0.125
## 4    4 0.150
## 5    5 0.200
## 6    6 0.350
```

```
is.device(loaded_die)
```

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

```
# bad sides (there must be at least 2-sides)
invalid_device <- device(sides = c('a'))
```

```
## Error in check_sides(sides):
## 'sides' must be a vector of length greater than 1
```

```
# bad sides (duplicated sides)
bad_coin <- device(sides = c('heads', 'heads'))
```

```
## Error in check_sides(sides):
## 'sides' cannot have duplicated elements
```

```
# bad probability values for prob
bad_coin <- device(
  sides = c('a', 'b'),
  prob = c(0.2, 0.1))
```

```
## Error in check_prob(prob):
## elements in 'prob' must add up to 1
```

```
# sides and prob of different lengths
bad_example <- device(
  sides = c('a', 'b', 'c'),
  prob = c(0.2, 0.8))
```

```
## Error in device(sides = c("a", "b", "c"), prob = c(0.2, 0.8)):
## 'sides' and 'prob' have different lengths
```

```
# false device
is.device(c(1, 2, 3))
```

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

---

2) Object "rolls"

To *roll* a "device" you will have to create a roll() function that takes a device and a number times (i.e. number of rolls). The returned output is an object of class "rolls". Here's a basic example for roll():

```
# roll fair die 50 times
fair_die <- device()

set.seed(123)

fair50 <- roll(fair_die, times = 50)
fair50
```

```
## object "rolls"
##
## $rolls
##  [1] 2 1 2 1 1 2 1 1 1 2 1 2 1 1 2 1 2 2 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2
## [36] 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1
```

In order to create a "rolls" object write:

- a constructor function roll(). This function should have two arguments:
  - device: object of class "device".
  - times: number of times to roll the device (default value of 1).
  - the function should stop() if device is not of class "device".
- an auxiliary function check_times(), called by roll(), that checks the validity of the argument times. Basically, times must be a positive integer greater than or equal to 1.
- the output of roll() will be a **list** containing four elements:
  - rolls: vector with outputs of the rolls
  - sides: vector with the sides of the "device" object
  - prob: vector with probabilities for each side of the "device" object
  - total: total number of rolls (i.e. times)
- a "print" method for "rolls" objects—print.rolls()—that displays the class of the object, and the generated rolls.

Here are a couple of examples of various ways to call roll():

```
# roll fair die 50 times
fair_die <- device(sides = 1:6, prob = rep(1/6, 6))

# roll 50 times
set.seed(123)
fair_50rolls <- roll(fair_die, times = 50)
```

```r
# print
fair_50rolls
```

```
## object "rolls"
##
## $rolls
##  [1] 3 6 4 1 1 2 5 1 5 4 1 4 6 5 2 1 3 2 3 1 1 6 5 1 5 6 5 5 3 2 1 1 6 6 2
## [36] 4 6 3 3 3 2 4 4 4 2 2 3 4 3 1
```

```r
# what's in fair50?
names(fair50)
```

```
## [1] "rolls" "sides" "prob"  "total"
```

```r
fair50$rolls
```

```
##  [1] 2 1 2 1 1 2 1 1 1 2 1 2 1 1 2 1 2 2 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2
## [36] 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1
```

```r
fair50$sides
```

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

```r
fair50$prob
```

```
## [1] 0.5 0.5
```

```r
fair50$total
```

```
## [1] 50
```

A less basic example:

```r
# string die
str_die <- device(
  sides = c('a', 'b', 'c', 'd', 'e', 'f'),
  prob = c(0.075, 0.1, 0.125, 0.15, 0.20, 0.35))

# roll 20 times
set.seed(123)
str_rolls <- roll(str_die, times = 20)
names(str_rolls)
```

```
## [1] "rolls" "sides" "prob"  "total"
```

```r
str_rolls
```

```
## object "rolls"
##
```

```
## $rolls
##  [1] "f" "c" "e" "b" "a" "f" "e" "b" "d" "e" "a" "e" "d" "d" "f" "b" "f"
## [18] "f" "f" "a"
```

3) Summary method for object "rolls"

Write a *summary* method—i.e. `summary.rolls()`—for `"rolls"` objects that returns an object `"summary.rolls"`. The output of `summary.rolls()` will be a **list** containing a **data frame** called `freqs` with 3 columns:

- `side`: the sides of the rolled `device`.
- `count`: the frequency (count) of each side of the rolled `device`.
- `prop`: the relative frequency (proportion) of each side of the rolled `device`.

You will also have to write a *print* method for the summary—i.e. `print.summary.rolls()`—such that when a `summary.rolls` is printed, you get an output like the following example:

```
set.seed(123)
fair_50rolls <- roll(fair_die, times = 50)
fair50_sum <- summary(fair_50rolls)

fair50_sum
```

```
## summary "rolls"
##
##   side count prop
## 1    1    11 0.22
## 2    2     8 0.16
## 3    3     9 0.18
## 4    4     8 0.16
## 5    5     7 0.14
## 6    6     7 0.14
```

```
# class
class(fair50_sum)
```

```
## [1] "summary.rolls"
```

```
# what's in the summary
names(fair50_sum)
```

```
## [1] "freqs"
```

```
fair50_sum$freqs
```

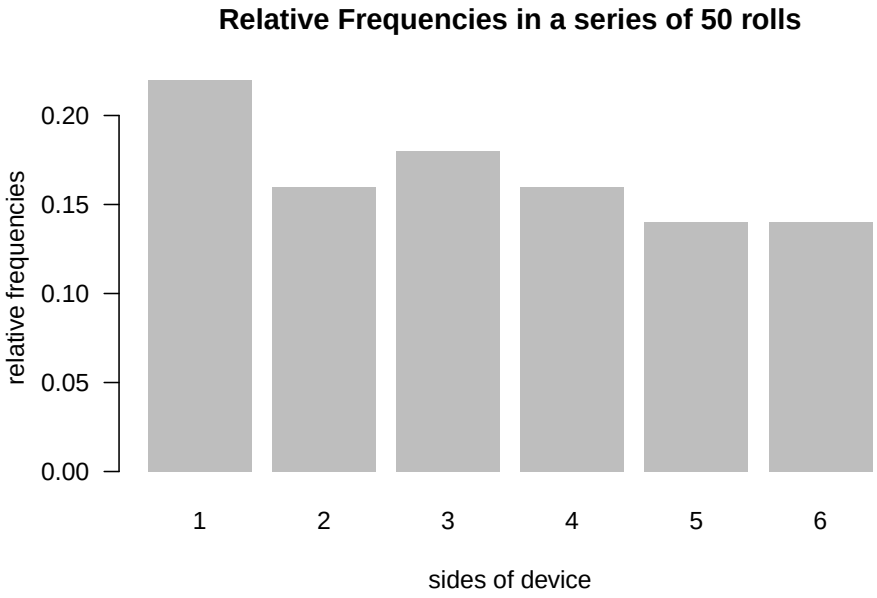
```
##   side count prop
## 1    1    11 0.22
```

```
## 2    2     8 0.16
## 3    3     9 0.18
## 4    4     8 0.16
## 5    5     7 0.14
## 6    6     7 0.14
```

4) Plot methd for object `"rolls"`

Write a *plot* method for `"rolls"` objects—i.e. `plot.rolls()`. You need to graph a barchart of proportions (e.g. relative frequencies of `1`'s, `2`'s, `3`'s, `4`'s, `5`'s, and `6`'s).

```
# plot method
plot(fair_50rolls)
```



5) Additional Methods

Also, write functions for the following methods of an object `"rolls"`:

- an extraction method `"["` to extract the value of a given roll.
- a replacement method `"[<-"` to replace the value of a given roll.
- an addition `"+"` method to add more rolls.

Example

Here's a comprehensive example in which a 8-sided *device* is created, and then it gets rolled 500 times to obtain an object `"roll"` on which we apply the various programmed methods:

```r
# roll fair 8-sided die
set.seed(123)
fair_dev <- device(sides = letters[1:8], prob = rep(1/8, 8))
fair500 <- roll(fair_dev, times = 500)

# summary method
summary(fair500)
```

```
## summary "rolls"
##
##   side count  prop
## 1    a    63 0.126
## 2    b    54 0.108
## 3    c    73 0.146
## 4    d    69 0.138
## 5    e    69 0.138
## 6    f    51 0.102
## 7    g    65 0.130
## 8    h    56 0.112
```

```r
# extracting roll in position 500
fair500[500]
```

```
## [1] "h"
```

```r
# replacing last roll
fair500[500] <- 'a'
fair500[500]
```

```
## [1] "a"
```

```r
summary(fair500)
```

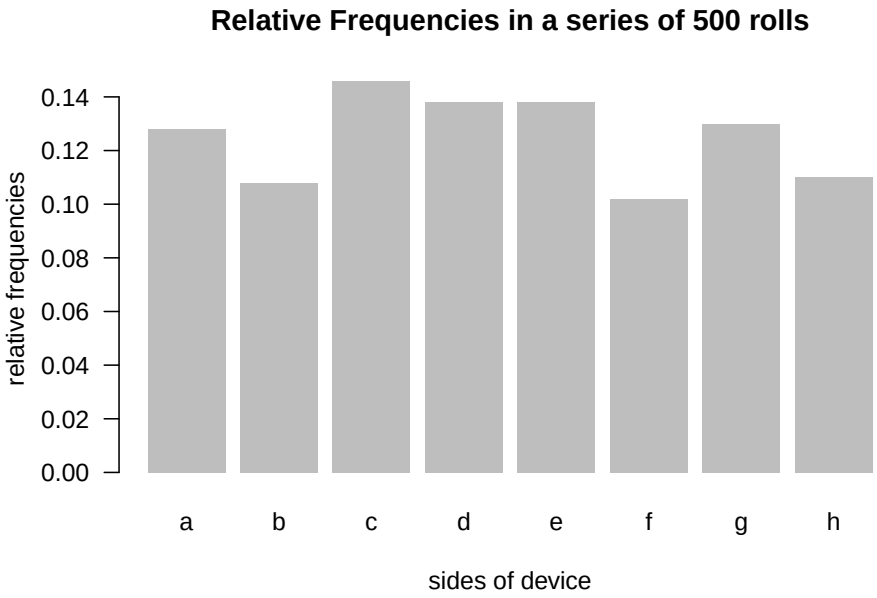
```
## summary "rolls"
##
##   side count  prop
## 1    a    64 0.128
## 2    b    54 0.108
## 3    c    73 0.146
## 4    d    69 0.138
## 5    e    69 0.138
## 6    f    51 0.102
## 7    g    65 0.130
## 8    h    55 0.110
```

```r
# adding 100 rolls
fair600 <- fair500 + 100
```

```
summary(fair600)
```

```
## summary "rolls"
##
##   side count   prop
## 1    a    79 0.1317
## 2    b    69 0.1150
## 3    c    78 0.1300
## 4    d    85 0.1417
## 5    e    78 0.1300
## 6    f    67 0.1117
## 7    g    74 0.1233
## 8    h    70 0.1167
```

```
# plot method
plot(fair500, 500)
```



Package Creation

Carefully check the example package `"cointoss"` to get some hints and inspiration. We expect that you write your own code, with your consistent style (avoid the temptation of copy/plagiarism):

https://github.com/gastonstat/cointoss

Tests: Your package should include tests for your functions `device()` and `roll()`, as well as for the auxiliary functions called by them: e.g. `check_sides()`, `check_prob()`,

`check_times()`.

Vignette: Your package should include an introductory vignette that shows the user how to utilize the various functionalities of your package.

Package Structure

After completion, your package `"roller"` should have the following filestructure:

```
roller/
   .Rbuildignore
   roller.Rproj
   devtools-flow.R
   DESCRIPTION
   NAMESPACE
   README.md
   R/
   man/
   tests/
   vignettes/
   inst/
```

Submission

- Create a folder (i.e. subdirectory) `roller/` in your github classroom repository. This will be the folder of your package.
- Create another folder (i.e. subdirectory) `workout03` in your github classroom repository.
- Use `workout03` to write a report (github document) in which you use your package `"roller"` to show us how to use your package. You can take the content of your package vignette, and add more content, examples, and tutorials.
- The due date is Dec-02 (before midnight). You will have to show your package to your GSI during lab, either Dec-5 or Dec-6. We will only grade committed work pushed to your classroom repo before the deadline.