# L08 – Klausurvorbereitung – Lösung

31. Mai 2021

## Contents



```
library(tidyverse)
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
library(rlang)
##
## Attaching package: 'rlang'
## The following objects are masked from 'package:purrr':
##
##     %@%, as_function, flatten, flatten_chr, flatten_dbl, flatten_int,
##     flatten_lgl, flatten_raw, invoke, list_along, modify, prepend,
##     splice
```

# 1 Typen

Erzeuge eine Liste `objs`, sodass folgender Code eine möglichst große Zahl ergibt.

```
objs %>% sapply(typeof) %>% unique() %>% length()
```

```
objs <- list(
  NULL,
  T,
  0L,
  0,
  1i,
  "",
  raw(), # Diese Funktion muss man nicht kennen. Existenz des Typen raw schon.
  list(),
  rlang::env(),
  function() 0,
  sum,  # Man sollte wissen, dass es 3 unterschiedliche Funktionentypen gibt.
  `[`,  # Die  Typen von sum und `[` muss man nicht wissen.
  rlang::expr(a),
  rlang::expr(!T),
  stats4::mle(function(x) x^2, list(x=1)), # Muss man nicht kennen.
  expression(1+1)) # Muss man nicht kennen.

objs %>% sapply(typeof)
##  [1] "NULL"        "logical"     "integer"     "double"      "complex"
##  [6] "character"   "raw"         "list"        "environment" "closure"
## [11] "builtin"     "special"     "symbol"      "language"    "S4"
## [16] "expression"
objs %>% sapply(typeof) %>% unique() %>% length()
## [1] 16
```

# 2 Basics

Nutze keine Schleife (`for`, `while`, `repeat`) und keine `*apply`-Funktionen in dieser Aufgabe.

Erzeuge folgende Funktionen, die Vektoren der angegebenen Muster ausgeben.

```
va <- function(n) 2^(0:n)
va(5)
## [1]  1  2  4  8 16 32
va(10)
##  [1]    1    2    4    8   16   32   64  128  256  512 1024

vb <- function(n) (n:-n)[-(n+1)]
vb(3)
## [1]  3  2  1 -1 -2 -3
vb(6)
##  [1]  6  5  4  3  2  1 -1 -2 -3 -4 -5 -6

vc <- function(n) rep(1:5, times = ceiling(n/5))[1:n]
vc(3)
## [1] 1 2 3
vc(13)
##  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3
```

```r
vc(18)
##  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3

vd <- function(n) rep(1:n,1:n)
vd(3)
## [1] 1 2 2 3 3 3
vd(5)
##  [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
```

## 3 Zentrumsmatrix

Nutze keine Schleife (`for`, `while`, `repeat`) und keine `*apply`-Funktionen in dieser Aufgabe.

Erstelle eine Funktion `center_mat(n)`, wobei wir `n` als positive ganze Zahl $n \in \mathbb{N}$ annehmen. Die Funktion gibt eine $(2n+1) \times (2n+1)$-Matrix zurück, die den "Feld-Abstand" zum mittleren Eintrag enthält, siehe Ausgaben.

```r
center_mat <- function(n) {
  A <- matrix(abs(n:(-n)), nrow=2*n+1, ncol=2*n+1)
  A + t(A)
}
center_mat(0)
##      [,1]
## [1,]    0
center_mat(1)
##      [,1] [,2] [,3]
## [1,]    2    1    2
## [2,]    1    0    1
## [3,]    2    1    2
center_mat(2)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    4    3    2    3    4
## [2,]    3    2    1    2    3
## [3,]    2    1    0    1    2
## [4,]    3    2    1    2    3
## [5,]    4    3    2    3    4
center_mat(5)
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,]   10    9    8    7    6    5    6    7    8     9    10
##  [2,]    9    8    7    6    5    4    5    6    7     8     9
##  [3,]    8    7    6    5    4    3    4    5    6     7     8
##  [4,]    7    6    5    4    3    2    3    4    5     6     7
##  [5,]    6    5    4    3    2    1    2    3    4     5     6
##  [6,]    5    4    3    2    1    0    1    2    3     4     5
##  [7,]    6    5    4    3    2    1    2    3    4     5     6
##  [8,]    7    6    5    4    3    2    3    4    5     6     7
##  [9,]    8    7    6    5    4    3    4    5    6     7     8
## [10,]    9    8    7    6    5    4    5    6    7     8     9
## [11,]   10    9    8    7    6    5    6    7    8     9    10
```

## 4 Funktionsformen

**a)** Erzeuge eine Funktion, mit der wie unten `NA`-Einträge von atomaren Vektoren ersetzt werden können.

```r
`na<-` <- function(x, value) {
  x[is.na(x)] <- value
  x
}
x <- c(NA, 1, 2, NA, NA , 3)
na(x) <- c(10, 20, 30)
x
## [1] 10  1  2 20 30  3
y <- c(1, 2, NA, NA)
na(y) <- 100
y
## [1]   1   2 100 100
```

**b)** Schreibe eine Infixfunktion `%+%`, die die Summe berechnet und die Summanden in Attributen `left` und `right` speichert.

```r
`%+%` <- function(x, y) {
  res <- x + y
  attr(res, "left") <- x
  attr(res, "right") <- y
  res
}
1 %+% 2
## [1] 3
## attr(,"left")
## [1] 1
## attr(,"right")
## [1] 2
1:5 %+% 10
## [1] 11 12 13 14 15
## attr(,"left")
## [1] 1 2 3 4 5
## attr(,"right")
## [1] 10
```

# 5 Präfixform

Schreibe `(1+2):3*4` in eine äquivalente Form, in der nur Funktionsaufrufe in Präfixform vorkommen!

```r
(1+2):3*4
## [1] 12
`*`(`:`(`(`(`+`(1, 2)), 3), 4)
## [1] 12
```

# 6 Simulation

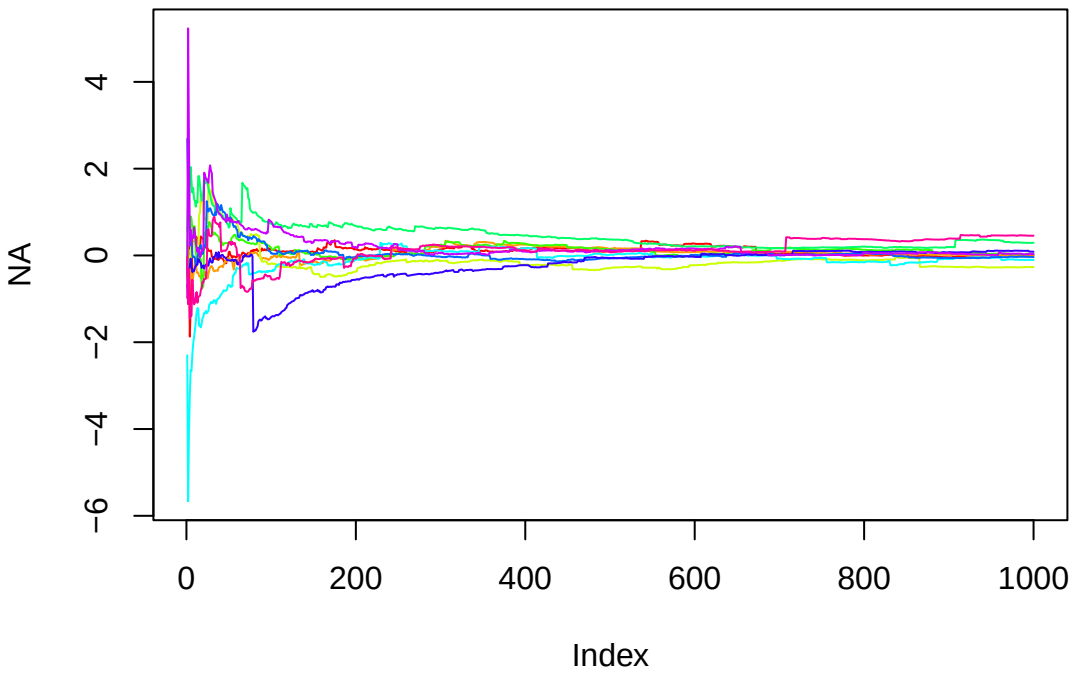
**a)** Nutze keine Schleife (`for`, `while`, `repeat`) in dieser Teilaufgabe. `*apply`-Funktionen sind erlaubt.

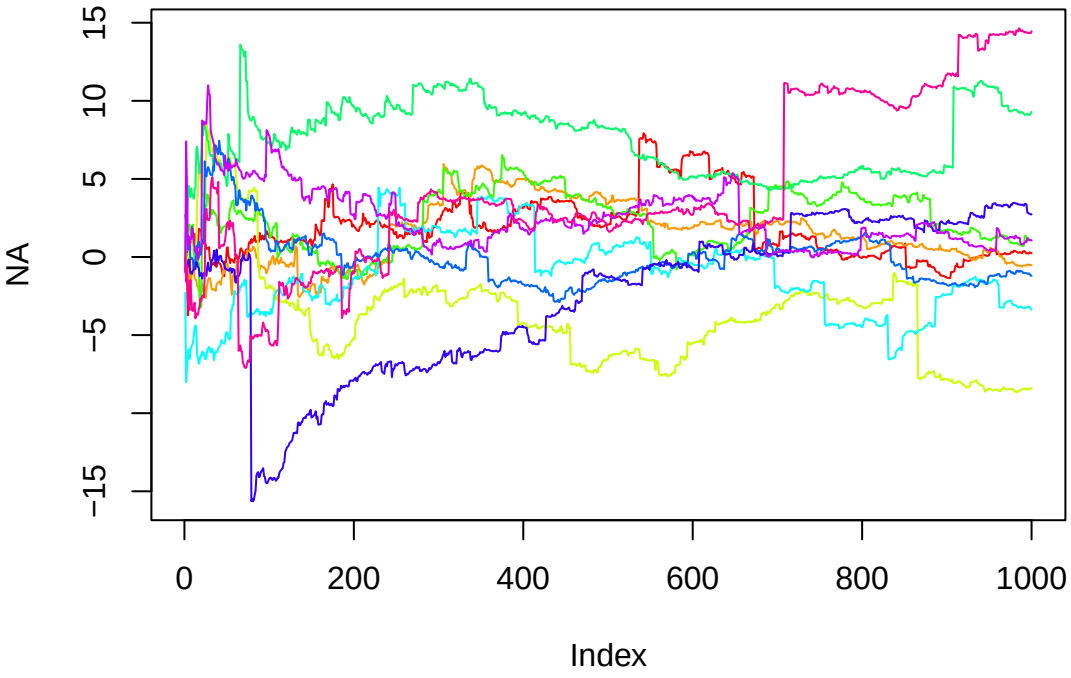
Ziehe mittels `rt(n, df=1.5)` $n = 1000$ $t_{1.5}$-verteilte Zufallszahlen $x_1, x_2, \ldots, x_n$. Berechne daraus die partiellen arithmetischen Mittel $a_k = \frac{1}{k}\sum_{i=1}^{k} x_i$ und normierten Summen $s_k = \frac{1}{\sqrt{k}}\sum_{i=1}^{k} x_i$. Wiederhole dies $m = 10$-mal, um $a_k^{(j)}$ und $s_k^{(j)}$ für $k = 1, \ldots, n$ und $j = 1, \ldots, m$ zu erhalten.

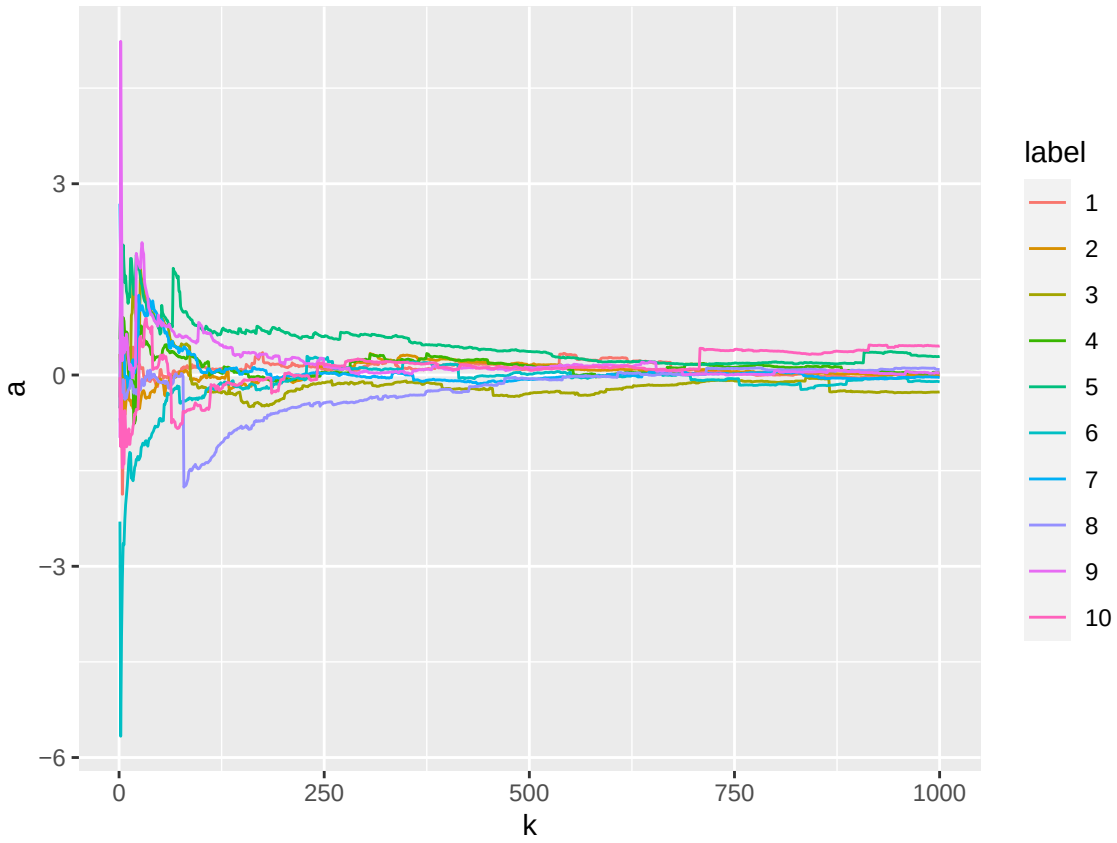
**b)** Erstelle zwei Plots, jeweils mit $m$ Kurven $k \mapsto a_k^{(j)}$ bzw $k \mapsto s_k^{(j)}$ für $j = 1, \ldots, m$.

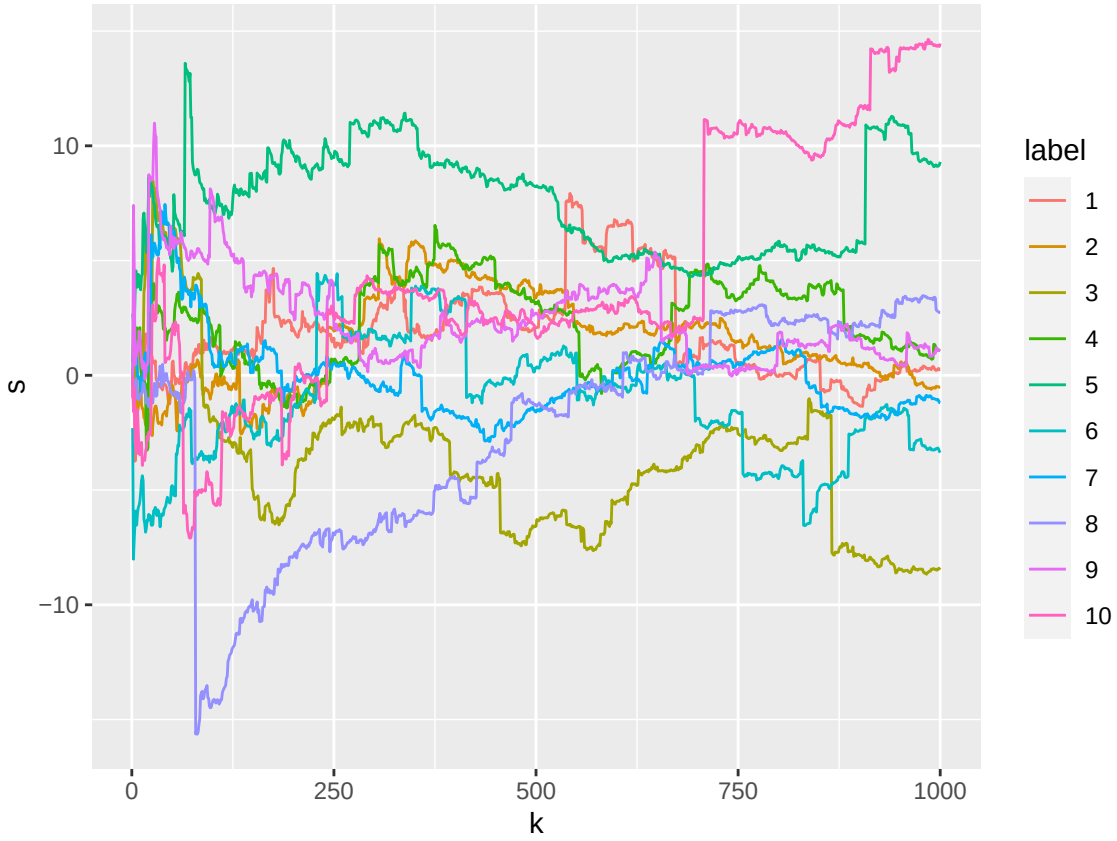
**Hintergrund:** Für eine $t_{1.5}$-verteilte Zufallsvariable $X$ gilt $\mathbf{E}[X] = 0$, $\mathbf{E}[|X|] < \infty$ und $\mathbf{E}[X^2] = \infty$. Für eine Folge unabhängiger Zufallsvariablen dieser Verteilung gilt das Gesetz der großen Zahlen ($a_k \to 0$ fast sicher), aber der zentrale Grenzwertsatz gilt nicht ($s_k \not\to \mathcal{N}(0, \sigma^2)$ in Verteilung).

```r
set.seed(1)
n <- 1000
m <- 10
x <- matrix(rt(n*m, df=1.5), nrow=m)
sums <- apply(x, 1, cumsum)
a <- sums / (1:n)
s <- sums / sqrt(1:n)

# alternative
set.seed(1)
n <- 1000
m <- 10
res <- replicate(m, {
  x <- rt(n, df=1.5)
  sums <- cumsum(x)
  a <- sums / (1:n)
  s <- sums / sqrt(1:n)
  c(a, s)
})
a <- res[1:n, ]
s <- res[(n+1):(2*n), ]


# base graphics plots
plot(NA, xlim=c(1, n), ylim=range(a))
for (i in 1:m) lines(a[,i], col=rainbow(m)[i])
plot(NA, xlim=c(1, n), ylim=range(s))
for (i in 1:m) lines(s[,i], col=rainbow(m)[i])

# ggplots
library(tidyverse)
dim(a) <- NULL
dim(s) <- NULL
tibble(
  k = rep(1:n, times = m),
  label = factor(rep(1:m, each = n)),
  a = a, s = s) -> tb
tb %>%
  ggplot(aes(x = k, y = a, color = label)) +
  geom_line()
tb %>%
  ggplot(aes(x = k, y = s, color = label)) +
  geom_line()
```


NA
4
2
0
−2
−4
−6
0
200
400
600
800
1000
Index


NA
15
10
5
0
−5
−15
0
200
400
600
800
1000
Index

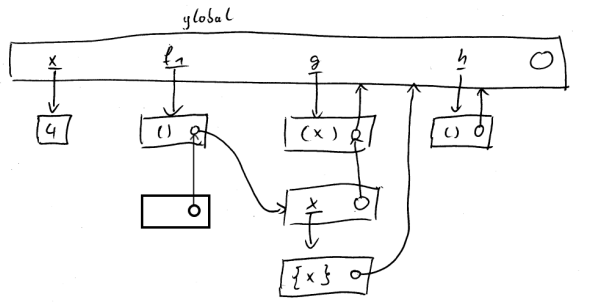

label
1
2
3
4
5
6
7
8
9
10
a
3
0
−3
−6
0
250
500
750
1000
k




# 7 Umgebungsdiagramm

Zeichne ein Umgebungsdiagramm, das den Programmzustand an der Markierung `#2` darstellt beim Aufruf von `f1()` an der Stelle `#1`!

```
x <- 1
g <- function(x) {
  function() {
    #2
    x
  }
}
h <- function() {
  function() x
}
x <- 2
f1 <- g(x)
x <- 4
f1() #1
## [1] 4
```




# 8 Strings

Nutze keine Schleife (`for`, `while`, `repeat`) und keine `*apply`-Funktionen in dieser Aufgabe.

Benutze `stringr::str_sub()` und `stringr::str_c()`, um folgende `character`-Vektoren zu erzeugen.

```
str_c(letters, LETTERS, collapse="")
## [1] "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ"
str_c(letters, collapse="X")
## [1] "aXbXcXdXeXfXgXhXiXjXkXlXmXnXoXpXqXrXsXtXuXvXwXxXyXz"
```

```
let <- str_c(letters, collapse="")
str_sub(let, (0:5)*5+1, (0:5)*5+5)
## [1] "abcde" "fghij" "klmno" "pqrst" "uvwxy" "z"
```

# 9 RegEx

**a)** Setze `pattern` auf einen `character`-Vektor der Länge 1, sodass mit `str_subset(s, pattern)` genau die Elemente eines `character`-Vektors `s` ausgewählt werden, die Wörter enthalten, die mit `en` enden. Der Code muss für beliebige `character`-Vektoren funktionieren, nicht nur für `example_strings` im Beispiel unten.

```
example_string <- c(
  "ente ente",
  "nichts sagend",
  "nichts sagen",
  "guten morgen",
  "guten tag")
pattern <- "en\\b"
str_subset(example_string, pattern)
## [1] "nichts sagen" "guten morgen" "guten tag"
```

**b)** Setze `pattern` und `replacement` jeweils auf einen `character`-Vektor der Länge 1, sodass mit `str_replace_all(s, pattern, replacement)` in `character`-Vektoren `s` direkt vor und nach Wörtern, die mit `#` markiert sind, einen Untenstrich `_` geschrieben wird. Der Code muss für beliebige `character`-Vektoren funktionieren, nicht nur für `example_strings` im Beispiel unten.

```
example_string <- "Guten #Tag! #Heute #ist Samstag. Wie geht #es Ihnen?"
pattern <- "#\\b([:alpha:]+)\\b"
replacement <- "_\\1_"
str_replace_all(example_string, pattern, replacement)
## [1] "Guten _Tag_! _Heute_ _ist_ Samstag. Wie geht _es_ Ihnen?"
```

# 10 S3-Dots

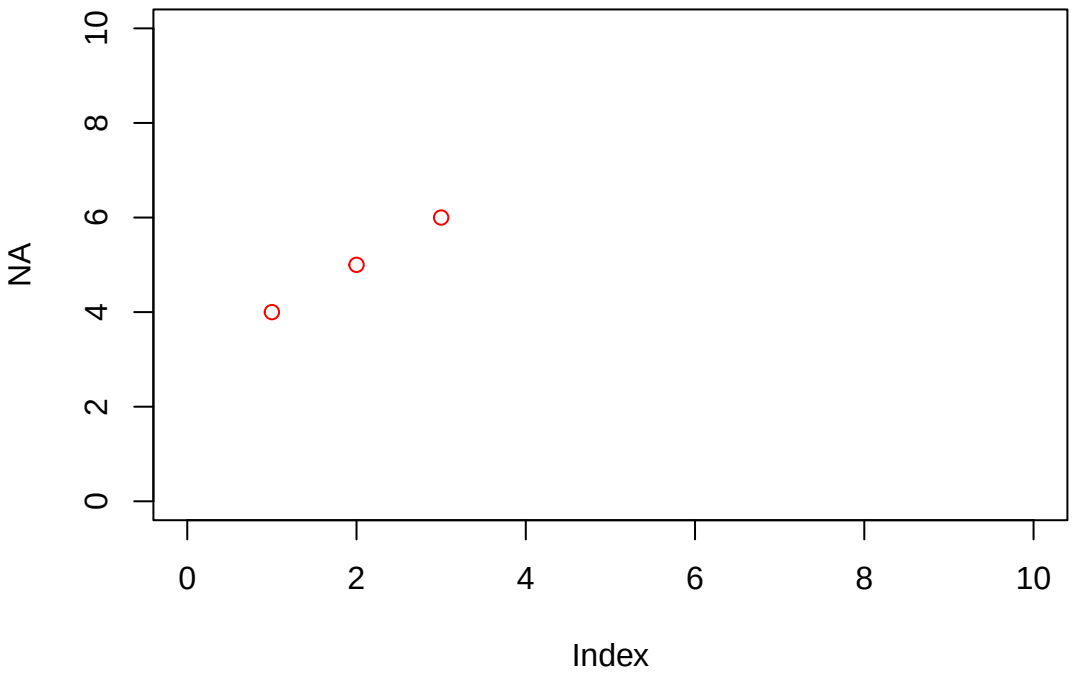
Ein S3-Objekt der Klasse `Dots` ist eine $(n \times 2)$-Matrix, die $n$ Punkte in $\mathbb{R}^2$ repräsentiert und ein Attribut `color` mit einem Farbwert hat.

**a)** Schreibe eine `print()`-Methode, die ausgibt, wie viele Punkte ein `Dots`-Objekt enthält und in welcher Farbe. Falls die Farbe einen Namen hat (siehe `colors()`) soll dieser ausgegeben werden, ansonsten der Hex-Code (siehe `col2hex()`). Schreibe dazu zuerst eine Funktion `color_name()`, die für einen Farbwert, wenn möglich einen Namen ausgibt und sonst den Hex-Code.

```
col2hex <- function(col) {
  rgb <- col2rgb(col)
  rgb(rgb["red", ], rgb["green", ], rgb["blue", ], max = 255)
}
color_name <- function(col) {
  col_hex <- col2hex(col)
  known_cols_hex <- col2hex(colors())
  i <- which(known_cols_hex == col_hex)
  if (length(i) == 0) return(col_hex)
  colors()[i][1]
}
print.Dots <- function(obj) {
  col <- attr(obj, "color")
  n <- nrow(obj)
  cat(n, color_name(col), if(n==1) "dot\n" else "dots\n")
}
structure(matrix(1:2, ncol=2), class="Dots", color=1)
## 1 black dot
structure(matrix(1:4, ncol=2), class="Dots", color=2)
## 2 #DF536B dots
obj <- structure(matrix(1:6, ncol=2), class="Dots", color="#FF0000")
obj
## 3 red dots
```

**b)** Schreibe eine Plot-Methode für `Dots`, die die Punkte in einen vorhanden Plot mit der entsprechenden Farbe einzeichnet.

```
plot.Dots <- function(obj) {
  points(obj, col = attr(obj, "color"))
}
obj
## 3 red dots
plot(NA, xlim=c(0, 10), ylim=c(0, 10))
plot(obj)
```




## 11 dplyr, tidyr

Wir betrachten den Datensatz `diamonds`, siehe `?ggplot2::diamonds`.

```
library(tidyverse)
data <- diamonds
data
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

Nutze `tidyverse`-Funktionen, um folgende Fragen zu beantworten.

**a)** Wie viele Diamanten gibt es im Datensatz pro `cut`-Qualität?

```
data %>% count(cut)
## # A tibble: 5 x 2
##   cut           n
##   <ord>     <int>
## 1 Fair       1610
## 2 Good       4906
## 3 Very Good 12082
## 4 Premium   13791
## 5 Ideal     21551
```

**b)** Was ist der Durchschnittspreis für jede `clarity`-Kategorie? Sortiere, sodass die teuerste oben ist.

```
data %>%
  group_by(clarity) %>%
  summarise(mean_price = mean(price)) %>%
  arrange(desc(mean_price))
## # A tibble: 8 x 2
##   clarity mean_price
##   <ord>        <dbl>
## 1 SI2          5063.
## 2 SI1          3996.
## 3 VS2          3925.
## 4 I1           3924.
## 5 VS1          3839.
## 6 VVS2         3284.
## 7 IF           2865.
## 8 VVS1         2523.
```

**c)** Wähle die Diamanten mit bestem `cut`, bester `color` und bester `clarity` aus und sortiere aufsteigend nach Preis

```
data %>%
  filter(cut == "Ideal", color == "D", clarity == "IF") %>%
  arrange(price)
## # A tibble: 28 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.27 Ideal D     IF       62.4    56   893  4.15  4.12  2.58
##  2  0.31 Ideal D     IF       61.1    56  1251  4.39  4.42  2.69
##  3  0.31 Ideal D     IF       61.1    56  1310  4.42  4.39  2.69
##  4  0.31 Ideal D     IF       60.5    57  1917  4.39  4.41  2.66
##  5  0.34 Ideal D     IF       62.1    57  2287  4.46  4.52  2.79
##  6  0.34 Ideal D     IF       59.8    57  2287  4.57  4.59  2.74
##  7  0.34 Ideal D     IF       62.1    57  2346  4.52  4.46  2.79
##  8  0.34 Ideal D     IF       59.8    57  2346  4.59  4.57  2.74
##  9  0.51 Ideal D     IF       62      56  3446  5.14  5.18  3.2
## 10  0.51 Ideal D     IF       62.1    55  3446  5.12  5.13  3.19
## # ... with 18 more rows
```

**d)** Berechne für jeden Diamanten das Volumen eines Quaders ($xyz$), der den Diamanten enthalten kann. Füge dies als Spalte `quader_vol` hinzu.

```
data %>% mutate(quader_vol = x*y*z)
## # A tibble: 53,940 x 11
##    carat cut       color clarity depth table price     x     y     z quader_vol
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>      <dbl>
```

```
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43       38.2
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31       34.5
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31       38.1
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63       46.7
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75       51.9
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48       38.7
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47       38.8
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53       42.3
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49       36.4
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39       38.7
## # ... with 53,930 more rows
```

**e)** Füge die Spalten `color` und `clarity` zu einer Spalte `color_clarity` zusammen, sodass die beiden Werte durch ein - getrennt sind.

```
data %>% unite("color_clarity", color, clarity, sep="-")
## # A tibble: 53,940 x 9
##    carat cut       color_clarity depth table price     x     y     z
##    <dbl> <ord>     <chr>         <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E-SI2          61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E-SI1          59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E-VS1          56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I-VS2          62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J-SI2          63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J-VVS2         62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I-VVS1         62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H-SI1          61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E-VS2          65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H-VS1          59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

**f)** Wähle nur Diamanten aus, die idealen `cut` haben. Berechne dann für jede `color-clarity`-Kombination den Durchschnittswert des Preises pro `carat` als Spalte `mean_price_per_carat`. Entferne alle Spalten bis auf `color`, `clarity`, und `mean_price_per_carat`. Verändere die Tabelle so, dass die Zeilen die Farbe angeben, die (weiteren) Spalten die `clarity` und die Einträge `mean_price_per_carat`.

```
data %>%
  filter(cut == "Ideal") %>%
  group_by(color, clarity) %>%
  summarise(mean_price_per_carat = mean(price / carat)) %>%
  pivot_wider(names_from=clarity, values_from=mean_price_per_carat)
## `summarise()` has grouped output by 'color'. You can override using the `.groups` argument.
## # A tibble: 7 x 9
## # Groups:   color [7]
##   color    I1   SI2   SI1   VS2   VS1  VVS2  VVS1    IF
##   <ord> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 D     2898. 3453. 3399. 3521. 3969. 5024. 4861. 9034.
## 2 E     3123. 3820. 3588. 3415. 3547. 4045. 4037. 4975.
## 3 F     3285. 3895. 3901. 3956. 4324. 4512. 4182. 4011.
## 4 G     3346. 3841. 3642. 4372. 4487. 4531. 4042. 3767.
## 5 H     3563. 4250. 4135. 3903. 3844. 3425. 3235. 3401.
## 6 I     3013. 4509. 4063. 3935. 3760. 3321. 2955. 2722.
## 7 J     3969. 4148. 3837. 3758. 3691. 3515. 2613. 2775.
```

# 12 Ausdrücke

Schreibe eine Funktion `brace_yourself(x)`, die den übergebenen Ausdruck mit geschweiften Klammern umschlossen wieder als Ausdruck zurückgibt.

```
library(rlang)
brace_yourself <- function(x) {
  arg <- enexpr(x)
  expr({!!arg})
}
brace_yourself(1+2)
## {
##     1 + 2
## }
brace_yourself(you == me)
## {
##     you == me
## }
```