

# L08 – Klausurvorbereitung

31. Mai 2021

## Contents

1	Typen	2
2	Basics	2
3	Zentrumsmatrix	2
4	Funktionsformen	3
5	Präfixform	3
6	Simulation	3
7	Umgebungsdiagramm	4
8	Strings	4
9	RegEx	4
10	S3-Dots	5
11	dplyr, tidyr	6
12	Ausdrücke	7

```
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()
```

# 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(5)
## [1] 1 2 4 8 16 32
va(10)
## [1] 1 2 4 8 16 32 64 128 256 512 1024

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(2)
## [1] 1 2
vc(13)
## [1] 1 2 3 4 5 1 2 3 4 5 1 2 3
vc(19)
## [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4

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.

```
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 4 3 2 1 0 1
## [2,] 9 8 7 6 5 4 3 2 1 0 1 2
## [3,] 8 7 6 5 4 3 2 1 0 1 2 3
## [4,] 7 6 5 4 3 2 1 0 1 2 3 4
## [5,] 6 5 4 3 2 1 0 1 2 3 4 5
## [6,] 5 4 3 2 1 0 1 2 3 4 5 6
## [7,] 6 5 4 3 2 1 0 1 2 3 4 5
## [8,] 7 6 5 4 3 2 1 0 1 2 3 4
## [9,] 8 7 6 5 4 3 2 1 0 1 2 3
## [10,] 9 8 7 6 5 4 3 2 1 0 1 2
## [11,] 10 9 8 7 6 5 4 3 2 1 0 1
```

## 4 Funktionsformen

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

```
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.

```
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!

## 6 Simulation

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

Ziehe mittels `rt(n, df=1.5)`  $n = 1000$  viele  $t_{1.5}$ -verteilte Zufallszahlen  $x_1, x_2, \dots, x_n$ . 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  $k = 1, \dots, n$  und  $j = 1, \dots, m$  zu erhalten.

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

**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 ( $a_k \rightarrow 0$  fast sicher), aber der zentrale Grenzwertsatz gilt nicht ( $s_k \not\rightarrow \mathcal{N}(0, \sigma^2)$  in Verteilung).

## 7 Umgebungsdiagramm

Zeichne ein Umgebungsdiagramm, das den Programzustand 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.

```
## [1] "aAbBcCdDeEfGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ"
## [1] "aXbXcXdXeXfXgXhXiXjXkXlXmXnXoXpXqXrXsXtXuXvXwXxXyXz"
## [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")
```

```
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 Unterstrich `_` 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?"
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.

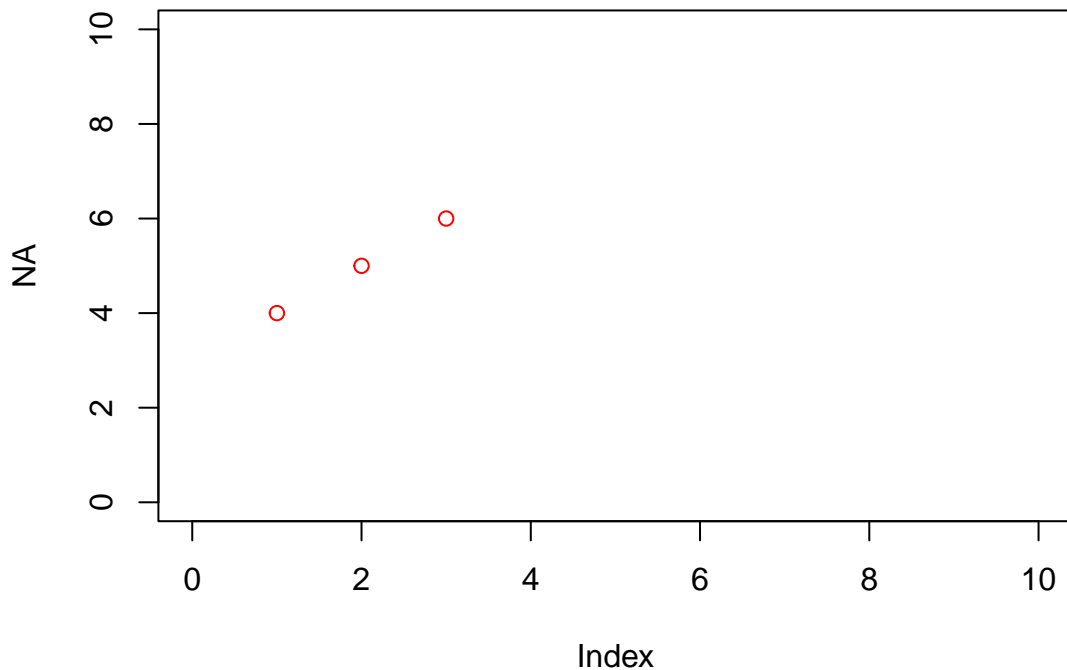
a) Schreibe eine `print()`-Methode, die ausgibt, wie viele Punkte ein `Dots`-Objekt enthält und in welcher Farbe. Falls die Farbe einen Namen hat (siehe `colors()`) soll dieser ausgegeben werden, ansonsten der Hex-Code (siehe `col2hex()`). Schreibe dazu zuerst eine Funktion `color_name()`, die für einen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

```
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 vorhandenen Plot mit der entsprechenden Farbe einzeichnet.

```
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.

- Wie viele Diamanten gibt es im Datensatz pro `cut`-Qualität?
- Was ist der Durchschnittspreis für jede `clarity`-Kategorie? Sortiere, sodass die teuerste oben ist.

- c) Wähle die Diamanten mit bestem `cut`, bester `color` und bester `clarity` aus und sortiere aufsteigend nach Preis
- d) Berechne für jeden Diamanten das Volumen eines Quaders ( $xyz$ ), der den Diamanten enthalten kann. Füge dies als Spalte `quader_vol` hinzu.
- e) Füge die Spalten `color` und `clarity` zu einer Spalte `color_clarity` zusammen, sodass die beiden Werte durch ein `-` getrennt sind.
- 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`.

## 12 Ausdrücke

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

```
brace_yourself(1+2)
## {
##     1 + 2
## }
brace_yourself(you == me)
## {
##     you == me
## }
```