LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Programmieren mit statistischer Software

Eva Endres, M.Sc.

Institut für Statistik
Ludwig-Maximilians-Universität München

Funktionen

Definition von Funktionen I

- Eine Funktion
  - hat einen Namen
  - benötigt Argumente (Input): arg_1, arg_2, ... → formale Argumente
  - führt (mit den Argumenten) Berechnungen durch und
  - gibt ein Ergebnis zurück (Rückgabewert)

```
> # name <- function(arg_1, arg_2, ...) {
> #   statements
> # }
```

- Funktionsaufruf

```
> # name(expr_1, expr_2)
```

Definition von Funktionen II

- Beispiel: Funktion, die den Index der Beobachtung zurückgibt, die am stärksten vom Median abweicht
  - Name der Funktion: `which_maxdev`
  - Input: x
  - Rückgabewert: standardmäßig die letzte Zeile der Funktion, ansonsten benötigt man `return()`

```
> # ?which.max
>
> which_maxdev <- function(x) {
+   mdn <- median(x)
+   devs <- abs(x - mdn)
+   which.max(devs)
+ }
```

Definition von Funktionen III

- Funktionsaufrufe

```
> data("Forbes2000", package = "HSAUR2")
>
> # summary(Forbes2000$sales)
> which_maxdev(Forbes2000$sales)
[1] 10
> # Forbes2000$sales[which_maxdev(Forbes2000$sales)]
>
> # summary(Forbes2000$marketvalue)
> which_maxdev(Forbes2000$marketvalue)
[1] 2
> # Forbes2000$marketvalue[which_maxdev(Forbes2000$marketvalue)]
```

Definition von Funktionen IV

- Ausgabe der letzten Zeile

```
> which_maxdev <- function(x) {
+   mdn <- median(x)
+   devs <- abs(x - mdn)
+   which.max(devs)
+   print("Hallo")
+ }
>
> which_maxdev(Forbes2000$sales)
[1] "Hallo"
```

- Ausgabe dessen, was mit `return()` explizit zurückgegeben wird (der Rest der Funktion wird nicht mehr ausgeführt)

Definition von Funktionen V

```
> which_maxdev <- function(x) {
+  mdn <- median(x)
+  devs <- abs(x - mdn)
+  return(which.max(devs))
+  print("Hallo")
+ }
>
> which_maxdev(Forbes2000$sales)
[1] 10
```

Das . . . Argument I

```
> which_maxdev <- function(x) {
+   mdn <- median(x)
+   devs <- abs(x - mdn)
+   which.max(devs)
+ }
>
> which_maxdev(Forbes2000$profits)
integer(0)
```

- Alle Funktionen innerhalb der selbst geschriebenen Funktion werden, soweit nicht anders angegeben, mit ihren Default-Argumenten verwendet

```
> median(Forbes2000$profits)
[1] NA
> args(median)
function (x, na.rm = FALSE, ...)
NULL
```

Das ... Argument II

- Argument `na.rm` (Erweiterung der Funktion `which_maxdev` mit dem `...` Argument)

```
> which_maxdev <- function(x, ...) {
+   mdn <- median(x, ...)
+   devs <- abs(x - mdn)
+   which.max(devs)
+ }
>
> which_maxdev(Forbes2000$profits, na.rm = TRUE)
[1] 364
```

Benannte Argumente und Standardeinstellungen I

- `na.rm=TRUE` als Default (Standardeinstellung)

```
> which_maxdev <- function(x, na.rm = TRUE) {
+   mdn <- median(x, na.rm = na.rm)
+   devs <- abs(x - mdn)
+   which.max(devs)
+ }
>
> which_maxdev(Forbes2000$profits)
[1] 364
```

- Äquivalente Aufrufe der Funktion `which_maxdev`
  - Bei Verwendung der Namen der Argumente kann die Reihenfolge beliebig variiert werden.
  - Ohne Verwendung der Namen der Argumente muss die Reihenfolge aus der Definition der Funktion eingehalten werden.

Benannte Argumente und Standardeinstellungen II

- Ohne Namen der Argumente

```
> which_maxdev(Forbes2000$sales)
[1] 10
> which_maxdev(Forbes2000$sales, FALSE)
[1] 10
> # which_maxdev(FALSE, Forbes2000$sales)   # funktioniert nicht
```

- Mit Namen der Argumente

```
> which_maxdev(x = Forbes2000$sales, na.rm = FALSE)
[1] 10
> which_maxdev(na.rm = FALSE, x = Forbes2000$sales)
[1] 10
> which_maxdev(na.rm = FALSE, Forbes2000$sales)
[1] 10
> # teilweise Benennung funktioniert auch
```

Argumentenmatching I

- 3-stufiger Prozess
  1. Exaktes Matching basierend auf den exakten Namen der Argumente

```
> which_maxdev(Forbes2000$sales, na.rm = FALSE)
[1] 10
> # Jeder Argument-Name darf nur einmal verwendet werden
> # which_maxdev(na.rm=Forbes2000$sales, na.rm = FALSE)
> # funktioniert nicht
```

  2. Partielles Matching basierend auf den (Anfangsbuchstaben der) Namen der Argumente

```
> # hier: n statt na.rm
> which_maxdev(Forbes2000$sales, n = FALSE)
[1] 10
> # which_maxdev(Forbes2000$sales, n = FALSE, na = FALSE)
> # funktioniert nicht
```

Argumentenmatching II

3. Matching basierend auf der Position der Argumente

```
> which_maxdev(n = FALSE, Forbes2000$sales)
[1] 10
> # Unmatched Arguments -> error
> which_maxdev(Forbes2000$sales, foo = 1)
Error in which_maxdev(Forbes2000$sales, foo = 1): unbenutztes
Argument (foo = 1)
```

Argumentenprüfung I

- Falsche Eingabe-Argumente müssen aufgefangen werden. Es sollte eine hilfreiche Fehlermeldung zurückgegeben werden.
- Fehlerhafte Verwendung der Funktion `which_maxdev`

```
> which_maxdev(Forbes2000$name)
> # Error in x - mdn : non-numeric argument to binary operator
> # In addition: Warning message:
> # In mean.default(sort(x, partial = half + 0L:1L)[half + 0L:1L]) :
> #   argument is not numeric or logical: returning NA
> which_maxdev(as.matrix(Forbes2000[, c("sales", "marketvalue")]))
> # [1] 2002
> which_maxdev(Forbes2000$sales, na.rm = "yes")
> # Error in if (na.rm) x <- x[!is.na(x)] else
> # if (any(is.na(x))) return(x[FALSE][NA]) :
> # argument is not interpretable as logical
```

Argumentenprüfung II

- Überprüfung der Eingabe mit `stopifnot()`

```
> which_maxdev <- function(x, na.rm = TRUE) {
+   stopifnot(is.numeric(x))
+   stopifnot(is.vector(x))
+   stopifnot(is.logical(na.rm))

+   mdn <- median(x, na.rm = na.rm)
+   devs <- abs(x - mdn)
+   which.max(devs)
+ }
>
> which_maxdev(Forbes2000$name)
Error: is.numeric(x) is not TRUE
> which_maxdev(as.matrix(Forbes2000[, c("sales", "marketvalue")]))
Error: is.vector(x) is not TRUE
> which_maxdev(Forbes2000$sales, na.rm = "yes")
Error: is.logical(na.rm) is not TRUE
```

Argumentenprüfung III

- Ausblick: Anwendung einer Funktion, die für einen Vektor definiert ist, auf eine Matrix

```
> # mit apply()
> apply(as.matrix(Forbes2000[, c("sales", "marketvalue")]), 2,
+       which_maxdev)
```

```
      sales marketvalue
         10           2
```

Rückgabewert I

- Jegliches R-Objekt kann von einer Funktion zurückgegeben werden.
- Default: Wert des letzten Ausdrucks (der letzten Zeile) oder mit `return()` → die Funktion wird dann verlassen

```
> which_maxdev <- function(x, na.rm = TRUE) {
+   stopifnot(is.numeric(x) & is.vector(x))
+   stopifnot(is.logical(na.rm))

+   mdn <- median(x, na.rm = na.rm)
+   devs <- abs(x - mdn)
+   return(which.max(devs))
+ }
```

- kann direkt ausgegeben oder zugewiesen werden

```
> which_maxdev(Forbes2000$sales)
[1] 10
> res <- which_maxdev(Forbes2000$sales)
> res
[1] 10
```

Rückgabewert II

- mit `invisible()` → Objekt nur nach Zuweisung sichtbar

```
> which_maxdev <- function(x, na.rm = TRUE) {
+   stopifnot(is.numeric(x) & is.vector(x))
+   stopifnot(is.logical(na.rm))

+   mdn <- median(x, na.rm = na.rm)
+   devs <- abs(x - mdn)
+   invisible(which.max(devs))
+ }
```

- Ergebnis wird nicht ausgegeben, kann aber zugewiesen werden

```
> which_maxdev(Forbes2000$sales)
> res <- which_maxdev(Forbes2000$sales)
> res
[1] 10
```

Funktionsbereich I

- Regeln, um den Wert eines Symbols zu finden
- 3 Klassen von Symbolen:
  1. Formale Parameter - Argumente der Funktion - hier `x`
  2. Lokale Variablen - werden innerhalb der Funktion definiert - hier `y`
  3. Freie Variablen - der Rest - hier `z` hierzu wird das Environment der Funktion durchsucht, dann der Umgebung, usw., bis zum Globalen Environment
- 1. und 2. werden auch gebundene Variablen genannt. Sie existieren nur innerhalb des Environments der Funktion und nicht außerhalb.

```
> f <- function(x) {
+   y <- 2 * x
+   cat("x =", x, "\n")
+   cat("y =", y, "\n")
+   cat("z =", z, "\n")
+ }
```

Funktionsbereich II

```
> # Fehlermeldung, da z nicht definiert ist
> f(2)

x = 2
y = 4

Error in cat("z =", z, "\n"): Objekt 'z' nicht gefunden

> # Definiton von z
> z <- 42
> f(2)

x = 2
y = 4
z = 42

> # Warnung: Die Verwendung freier Variablen in einer Funktion
> # ist sehr fehleranfaellig
> # und sollte deshalb vermieden werden.
```