LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Programmieren mit statistischer Software

Eva Endres, M.Sc.

Institut für Statistik
Ludwig-Maximilians-Universität München

Navigation

Global Environment I

- Definition von Objekten im Global Environment
- Operationen in der Kommandozeile betreffen die Objekte im Global Environment

```
> x <- 1
> ls()
[1] "x"
```

- Der Benutzer kann auf alles zugreifen, was er selbst im Global Environment definiert hat; zusätzlich auf alles, was in geladenen Paketen vorhanden ist

```
> search()
 [1] ".GlobalEnv"        "package:knitr"     "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

- Daher kann die Funktion `identity()` verwendet werden

```
> identity(1)
[1] 1
```

Global Environment II

- Identity liegt im Paket `base`

```
> find("identity")
[1] "package:base"
```

Search Path I

- Definition von `identity` im Global Environment

```
> identity <- 17
> # oder
> identity <- function(x){
+   return(18)
+ }
```

- `identity` existiert jetzt zweimal

```
> find("identity")
[1] ".GlobalEnv"   "package:base"
```

- Der erste Eintrag von `identity` im Suchpfad wird standardmäßig verwendet

```
> identity
function(x){
  return(18)
}
> # oder
> identity(10)
[1] 18
```

Search Path II

- Zugriff auf die Funktion `identity` aus dem Paket `base` mit `::`

```
> base::identity(10)
[1] 10
```

- Löschen des `identity`-Objekts löscht dieses im Global Environment

```
> rm(identity)
> ls()
[1] "x"
```

- Das Entfernen eines Objekts innerhalb eines Paketes ist nicht erlaubt

```
> # rm(base::identity) ist unzul\"assig
```

Packages

- Objekte in einem Paket haben auch ein Environment (hier: `base`)

```
> identity

function (x)
x
<bytecode: 0x00000000007971420>
<environment: namespace:base>
```

Namespaces I

- Im Global Environment sind alle Objekte sichtbar
- In Paketen müssen nicht alle Objekte für den Benutzer sichtbar sein

```
> library("HSAUR2")
```

```
Loading required package: tools
```

- Speichern der vollständigen Umgebung eines Pakets (Namespace)

```
> ns <- getNamespace("HSAUR2")
```

- Liste aller exportierten, für den Benutzer sichtbaren Objekte

```
> getNamespaceExports(ns)
```

```
[1] "HSAURtable"
```

- Auf exportierte Objekte kann direkt zugegriffen werden

```
> HSAURtable
```

```
function (object, ...)
UseMethod("HSAURtable")
<environment: namespace:HSAUR2>
```

Namespaces II

- Liste aller im Namespace definierten Objekte

```
> ls(envir = ns)
 [1] "caption"                "chkS"
 [3] "cpRoutsave"             "exename"
 [5] "extRact"                "extractBibtex"
 [7] "gattach"                "HSAURcite"
 [9] "HSAURtable"             "HSAURtable.data.frame"
[11] "HSAURtable.table"       "isep"
[13] "isi2bibtex"             "pkgs"
[15] "pkgversions"            "pkgyears"
[17] "prettyS"                "readBibtex"
[19] "Rwelcome"               "toBibtex.HSAURcitation"
[21] "toBibtex.txtBibtex"     "toLatex.dftab"
[23] "toLatex.tabtab"
```

Namespaces III

- Zugriff auf nicht exportierte Objekte mit `:::`

```
> HSAUR2:::exename
function ()
{
    tversion <- paste(version$major, "0", substr(version$minor,
        1, 1), substr(version$minor, 3, 3), sep = "")
    return(paste("rw", tversion, ".exe", sep = ""))
}
<environment: namespace:HSAUR2>
> # HSAUR2::exename ist nicht m\"oglich
```

- Hinweis: Nicht exportierte Objekte sollten nicht verwendet werden!

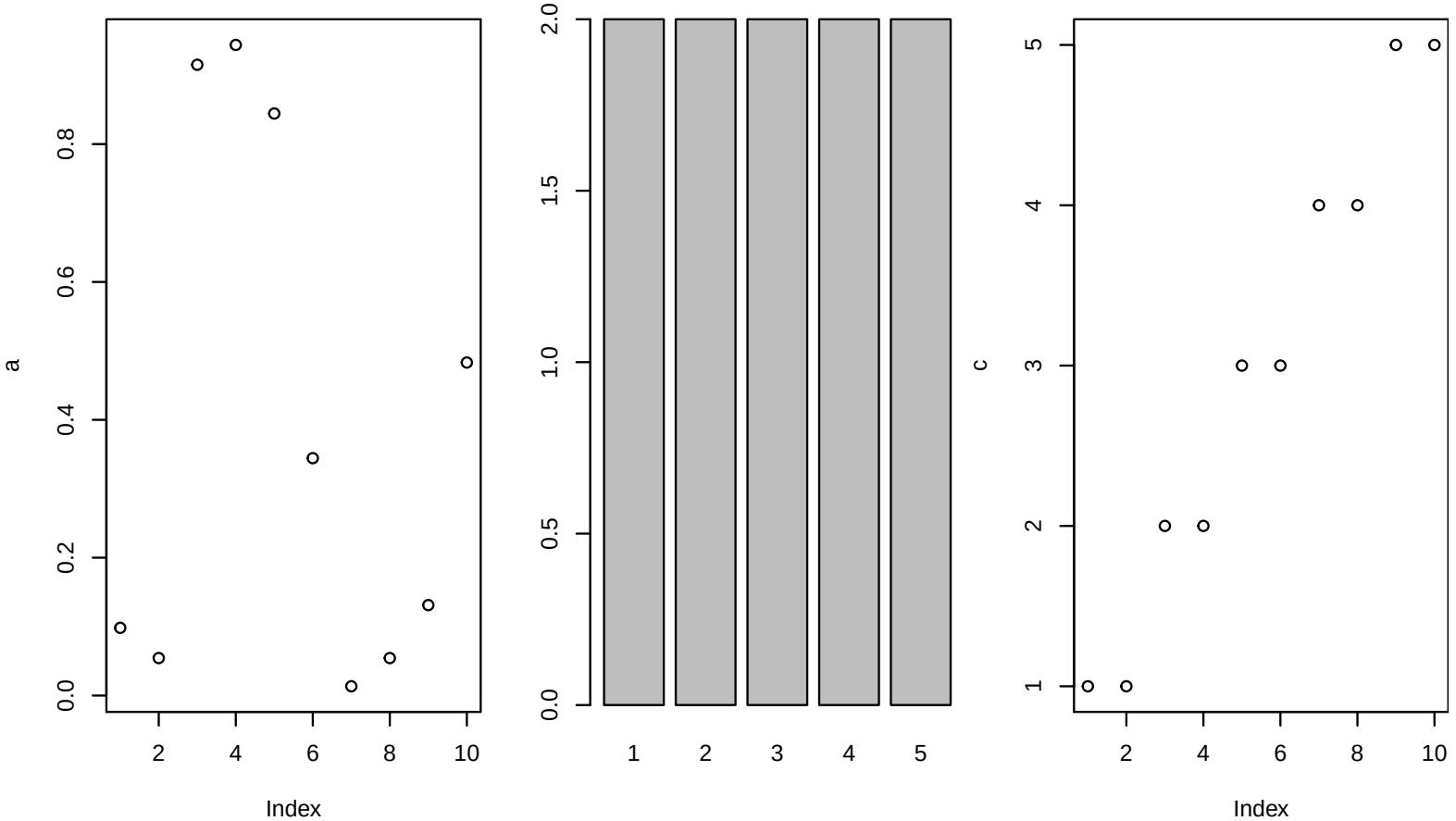
Generic functions (S3) I

```
> a <- runif(10)
> b <- gl(5, 2)
> c <- sort(rep(1:5,2))
```

- Generische Funktionen liefern unterschiedliche Resultate je nach Klasse der Objekte, auf die sie angewendet werden

```
> par(mfrow = c(1, 3), mar = c(4, 4, 1, 0))
> plot(a)
> plot(b)
> plot(c)
```

Generic functions (S3) II



Generic functions (S3) III

- Klasse des Objekts (relevant für generische Funktion)

```
> class(a)
[1] "numeric"
> class(b)
[1] "factor"
> class(c)
[1] "integer"
```

- Weitere Eigenschaften der Objekte (nicht relevant für generische Funktion)
  - R internal type or storage mode

```
> typeof(a)
[1] "double"
> typeof(b)
[1] "integer"
> typeof(c)
[1] "integer"
```

Generic functions (S3) IV

- type or storage mode (übergeordnet, basierend auf `typeof()`)

```
> mode(a)
[1] "numeric"
> mode(b)
[1] "numeric"
> mode(c)
[1] "numeric"
```

- Generische Funktionen erkennt man an `UseMethod("...")`

```
> plot
function (x, y, ...)
UseMethod("plot")
<bytecode: 0x0000000007386150>
<environment: namespace:graphics>
```

Generic functions (S3) V

- Anzeigen aller verfügbaren plot Methoden

```
> methods("plot")
 [1] plot.acf*           plot.data.frame*    plot.decomposed.ts*
 [4] plot.default        plot.dendrogram*    plot.density*
 [7] plot.ecdf           plot.factor*        plot.formula*
[10] plot.function       plot.hclust*        plot.histogram*
[13] plot.HoltWinters*   plot.isoreg*        plot.lm*
[16] plot.medpolish*     plot.mlm*           plot.ppr*
[19] plot.prcomp*        plot.princomp*      plot.profile.nls*
[22] plot.raster*        plot.spec*          plot.stepfun
[25] plot.stl*           plot.table*         plot.ts
[28] plot.tskernel*      plot.TukeyHSD*
see '?methods' for accessing help and source code
```

  - Exakter Name der jeweiligen Funktion für die jeweilige Klasse.
  - Namensschema für S3-Funktionen: `function.class`
  - Wenn keine der Klassen zutrifft, wird `plot.default` verwendet.
  - Nicht sichtbare Funktionen sind mit * gekennzeichnet.

Generic functions (S3) VI

- Öffnen der Hilfe zur entsprechenden Funktion der jeweiligen Klasse

```
> ?plot.factor
> ?plot.default
```

- Anzeigen des Quellcodes für S3-Funktionen

```
> getS3method("plot", "factor")
> getAnywhere("plot.factor")
> getAnywhere("plot.ecdf")
```

- Anzeigen des Quellcodes, falls Paket bekannt

```
> stats::plot.ecdf # wenn sichtbar
> graphics:::plot.factor # wenn nicht sichtbar
```

Print Objects (S3) I

- Auch die Funktion `print` ist eine generische Funktion

```
> x
[1] 1
> print(x)
[1] 1
> print
function (x, ...)
UseMethod("print")
<bytecode: 0x000000000808c678>
<environment: namespace:base>
```

- Verfügbare print-Methoden

```
> methods("print")
```

Generic functions (S4) I

- Das Paket `methods` stellt das S4 Objekt-System zur Verfügung; es ist immer automatisch geladen

```
> search()
 [1] ".GlobalEnv"        "package:HSAUR2"    "package:tools"
 [4] "package:knitr"     "package:stats"     "package:graphics"
 [7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:methods"   "Autoloads"         "package:base"
```

- Einige statistische Funktionen basierend auf dem S4-System sind im Paket `stats4` enthalten

```
> library("stats4")
> ns_stats4 <- getNamespace("stats4")
> ls(envir=ns_stats4)
 [1] "AIC"     "BIC"     "coef"    "confint" "logLik"  "mle"     "nobs"
 [8] "plot"    "profile" "summary" "update"  "vcov"
```

Generic functions (S4) II

- S4-Funktionen sehen etwas anders aus. Indikator ist `standardGeneric("...")`

```
> show(coef)
standardGeneric for "coef" defined from package "stats"

function (object, ...)
standardGeneric("coef")
<environment: 0x000000000ade94f8>
Methods may be defined for arguments: object
Use  showMethods("coef")  for currently available ones.

> # oder nur
> coef
```

- Verfügbaren Methoden für `coef`, analog zu `methods("...")`

```
> showMethods("coef")
Function: coef (package stats)
object="ANY"
object="mle"
object="summary.mle"
```

Generic functions (S4) III

- Quellcode anzeigen, analog zu `getS3method("...")`

```
> getMethod("coef", "mle")
Method Definition:

function (object, ...)
{
    .local <- function (object)
    object@fullcoef
    .local(object, ...)
}
<bytecode: 0x000000000ae35af0>
<environment: namespace:stats4>

Signatures:
        object
target  "mle"
defined "mle"
```

(das erste Argument ist der Name der Funktion, das zweite Argument die Klasse des Objekts)

Function Browsing I

- Schrittweise Ausführung einer Funktion mit `debug`
- Beispiel mit Funktion `jitter`

```
> debug(jitter)
> jitter(1:10)
```

- Beenden des `debug`-Modus

```
> undebug(jitter)
```

- Funktioniert auch für S3-Funktionen, aber nicht für S4

Browsing after an error I

- Debugging nur, falls ein Fehler auftritt
- Beispiel mit Funktion `jitter`

```
> options(error = recover)
> jitter(letters[1:3])
```

- Beenden des debug-Modus

```
> options(error = NULL)
```

- Alternativ: Aufruf von `traceback` nach einem Fehler

```
> jitter(letters[1:3])
> traceback()
```