EUROPA-FACHBUCHREIHE
Programmierung für die
IT-Ausbildung

R für IT-Berufe

Eine verständliche Einführung

```
Inhalt <- function(Nr)
{
  if (Nr == 1) {
      print("Kapitel 1: Einfuehrung in R")
  } else if (Nr == 2) {
      print("Kapitel 2: Objekte")
  } else if (Nr == 3) {
      print("Kapitel 3: Elementare Programmierung")
  } else if (Nr == 4) {
      print("Kapitel 4: Praxisnahe Anwendungen")
  } else if (Nr == 5) {
      print("Kapitel 5: Aufgabenpool")
  } else {
      print("Kapitel 6: Lernsituationen")
  }
}
for (i in (1:6)) {
    Inhalt (i)
}
```

```
[1] "Kapitel 1: Einfuehrung in R"
[1] "Kapitel 2: Objekte"
[1] "Kapitel 3: Elementare Programmierung"
[1] "Kapitel 4: Praxisnahe Anwendungen"
[1] "Kapitel 5: Aufgabenpool"
[1] "Kapitel 6: Lernsituationen"
```

1. Auflage

VERLAG EUROPA-LEHRMITTEL · Nourney, Vollmer GmbH & Co. KG
Düsselberger Str. 23 · 42781 Haan-Gruiten

Europa-Nr.: 84422L (4-Jahres-Lizenz) · 84422V (Jahreslizenz)

Verfasser:
Dirk Hardy, 46049 Oberhausen

1. Auflage 2024

Druck 5 4 3 2 1

Alle Drucke derselben Auflage sind parallel einsetzbar, da sie bis auf die Korrektur von Druckfehlern identisch sind.

ISBN 978-3-8085-8442-2 (4-Jahres-Lizenz)
ISBN 978-3-8085-8443-9 (Jahreslizenz)

www.europa-lehrmittel.de
Satz: Typework Layoutsatz & Grafik GmbH, 86153 Augsburg
Umschlag: braunwerbeagentur, 42477 Radevormwald
Umschlagfotos: AI Farm, imageteam – beide adobestock.com

Vorbemerkung

Die Programmiersprache ***R*** (1992 entstanden) wurde für mathematische und statistische Berechnungen entwickelt. Sie ist eine Weiterentwicklung der Programmiersprache ***S*** (1976 konzipiert), die für statistische Berechnungen und Grafiken entworfen wurde. Im Gegensatz zu ***S*** ist ***R*** eine freie Programmiersprache (freie Software) und kostenfrei für alle Benutzer anwendbar. Die Entwicklungs- und Laufzeitumgebung von ***R*** unterstützt nicht nur statistische Berechnungen, sondern bietet auch deren grafische Darstellung. Die Programmiersprache ***R*** ist auf dem TIOBE-Index (ein monatlich aktualisiertes Ranking von Programmiersprachen) unter der ersten 20 der beliebtesten und meist gebrauchten Programmiersprachen zu finden.

Aufbau des Moduls

Das vorliegende **Modul 1** möchte die Sprache ***R*** möglichst anschaulich, praxis- und unterrichtsnah vermitteln. Damit verfolgt dieses Buch einen **praktischen Ansatz.** Es ist die Ansicht des Autors, dass gerade in der schulischen Ausbildung der Zugang zu den komplexen Themen der Programmierung verstärkt durch anschauliche und praktische Umsetzung vorbereitet werden muss. Anschließend können allgemeine und komplexe Aspekte der Programmierung oder auch der Softwareentwicklung besser verstanden und umgesetzt werden.

Das Modul ist in **drei Teile** getrennt. Der **erste Teil** des Moduls (Kapitel 1–4) dient als **Informationsteil** und bietet eine **systematische Einführung in Programmierung mit der Sprache *R*.**

Der **zweite Teil** des Moduls (Kapitel 5) ist eine **Sammlung von Übungsaufgaben.** Nach der Erarbeitung der entsprechenden Kenntnisse aus dem Informationsteil können die Aufgaben aus diesem Teil zur weiteren Auseinandersetzung mit den Themen dienen und durch verschiedene Schwierigkeitsgrade auch die Differenzierung im Unterricht ermöglichen.

Der **dritte Teil** des Moduls (Kapitel 6) beinhaltet **Lernsituationen** basierend auf dem aktuellen Rahmenlehrplan für die IT-Berufe (speziell Daten- und Prozessanalyse). Lernsituationen konkretisieren sich aus den Lernfeldern und sollen im Idealfall vollständige Handlungen darstellen (Planen, Durchführen, Kontrollieren). Aus diesem Grund werden die Lernsituationen so angelegt, dass neben einer Planungsphase nicht nur die Durchführung (Implementation des Programms) im Blickpunkt steht, sondern auch geeignete Testverfahren zur Kontrolle des Programms bzw. des Entwicklungsprozesses in die Betrachtung einbezogen werden. Die Lernsituationen können aber auch als **Projektideen** verstanden werden.

Das Buch ist für Ausbildungsgänge im IT-Bereich konzipiert, die mit Datenanalyse arbeiten. Das sind vor allem die ***Fachinformatiker für Daten- und Prozessanalyse.***

Es kann aber von allen IT-Berufen und Interessierten genutzt werden, die einen Exkurs in die Datenanalyse machen möchten.

Für Anregungen und Kritik zu diesem Buch sind wir Ihnen dankbar (gerne auch per E-Mail).

Dirk Hardy

E-Mail: Hardy@DirkHardy.de

Im Frühjahr 2024

Verlag Europa-Lehrmittel

E-Mail: Info@Europa-Lehrmittel.de

Inhaltsverzeichnis

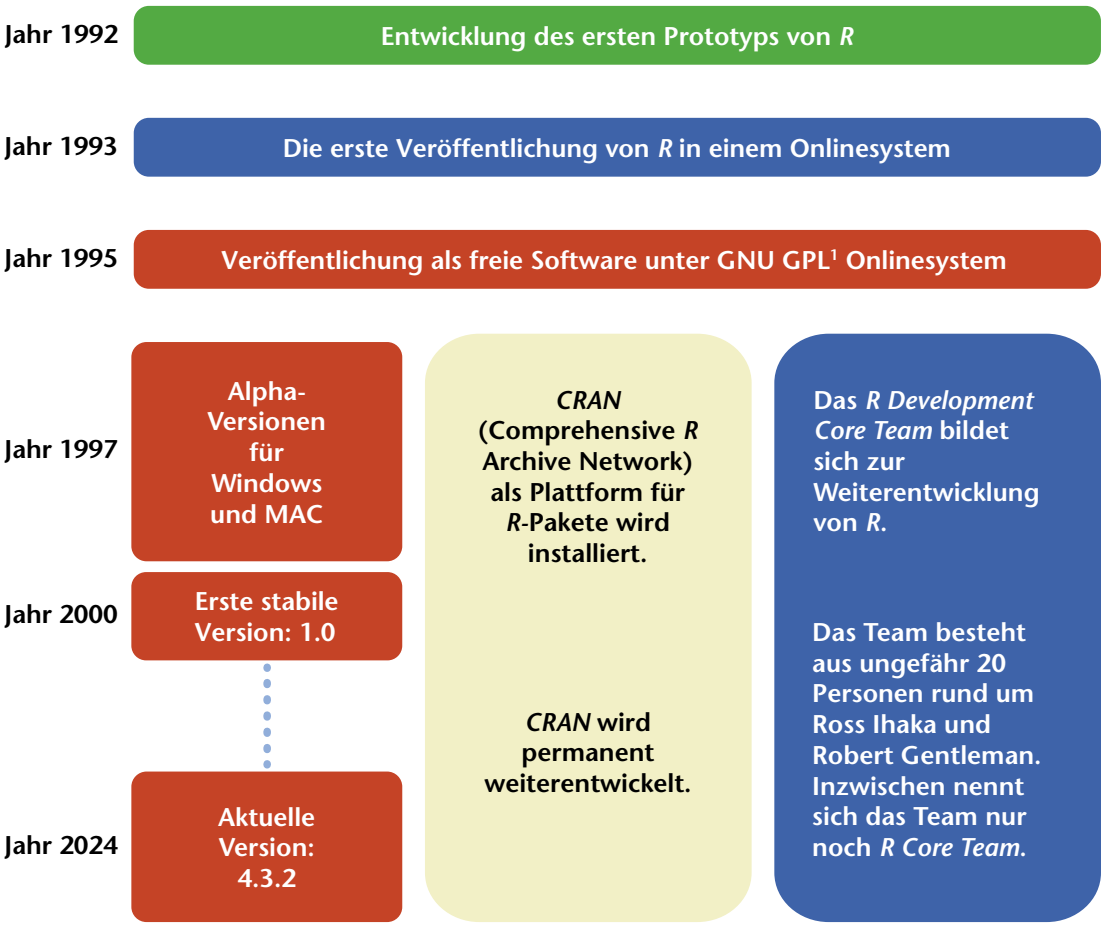
1 Einführung in R

1.1 Entstehung der Sprache R

Anfang den 90-er Jahren entwickelte der neuseeländische Statistiker *Ross Ihaka* zusammen mit dem kanadischen Statistiker *Robert Gentleman* die Programmiersprache ***R***. Beide arbeiteten an der Universität von Auckland in Neuseeland. Der Name R entstand wohl dadurch, dass einerseits die Vornamen beider Statistiker mit einem „R" beginnen und andererseits die Grundlage von ***R*** die Programmiersprache ***S*** war, so dass es nahelag, die Neuentwicklung auch nur mit einem Buchstaben zu benennen.

Im Laufe der Jahre entwickelte sich ***R*** zu einem der mächtigsten Werkzeuge im Bereich des Data Science und auch speziell der Datenanalyse. Das liegt sowohl an vielen eingebauten Funktionen, die der Kern von ***R*** schon bietet, aber auch an einer Vielzahl von Bibliotheken, die frei eingebunden werden können.
Neben Pyhon ist ***R*** damit die wichtigste Sprache in diesem Bereich.

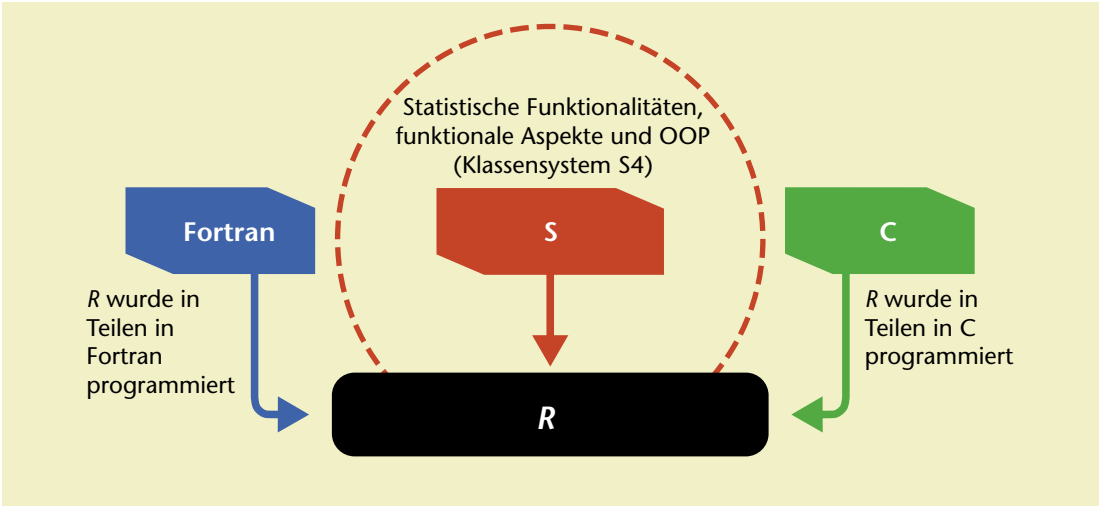
Die folgende Grafik zeigt den zeitlichen Verlauf der ***R***-Entwicklung:



1 Die **GNU** (unixähnliches Betriebssystem als freie Software) **GPL** (General Public License) ist eine Softwarelizenz, die es jedem Nutzer ermöglicht, die Software auszuführen, zu analysieren, zu ändern und zu verbreiten. Ein solche Software wird **freie Software** genannt.

1.2 Einordnung der Sprache R

R ist eine Programmiersprache, die verschiedene Programmier-Paradigma[2] unterstützt. Neben der Objektorientierung (alles, das in R gespeichert wird, ist ein Objekt) hat R auch den funktionalen Ansatz (alles, das in R ausgeführt wird, ist eine Funktion). R wurde mithilfe der Programmiersprachen C und Fortran implementiert und orientierte sich stark an der statistischen Programmiersprache S, die damals nur kommerziell zur Verfügung stand. Mit R als freier Software entstand dann eine kostenfreie Alternative zu S. Die folgende Grafik zeigt den Einfluss verschiedener Programmiersprachen an der Konzeption und Umsetzung der Sprache R.



Neben den zwei Hauptparadigmen der funktionalen und objektorientierten Programmierung ist es in R als Programmiersprache ebenfalls möglich mit Kontrollstrukturen zu arbeiten (strukturierte Programmierung) und im Sinne der prozeduralen Programmierung wiederkehrende Programmabschnitte in Funktionen auszulagern und bei Bedarf aufzurufen.

1.3 Strukturierte, prozedurale, objektorientierte und funktionale Programmierung

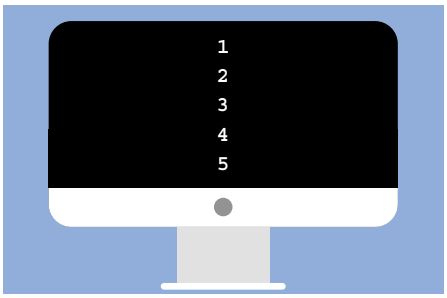
In der Einordnung der Sprache R wurde darauf hingewiesen, welche Programmier-Paradigma R umsetzt. Zum besseren Verständnis werden diese Paradigma anhand von Beispielen kurz erläutert:

Strukturierte Programmierung

Die strukturierte Programmierung zeichnet sich vor allem durch Kontrollstrukturen wie die **Selektion (`IF-ELSE`)** oder die **Iteration (`FOR`, `WHILE`)** aus. Damit erhält ein Programm eine nachvollziehbare Struktur.

Beispiel:

```
FÜR Var := 1 BIS 5 MIT SCHRITTWEITE 1
    SCHREIBE AUF BILDSCHIRM Var
```



Das Beispiel zeigt eine Iteration (Wiederholung) in so genanntem Pseudocode[3]. Dieser Code beschreibt den Ablauf des Programmes, ohne allerdings auf eine spezielle Programmiersprache einzugehen. In dem Beispiel wird eine Variable `Var` so lange um 1 erhöht, bis der Wert 5 erreicht ist. Jeder Wert der Variablen wird dann auf dem Bildschirm ausgegeben.

2 Paradigma kommt aus dem Griechischen und heißt so viel wie Muster oder Vorbild.
3 Pseudocode ist eine Art Sprache, mit der der Ablauf eines Programmes beschrieben wird. Pseudocode zeichnet sich dadurch aus, dass er näher an der natürlichen Sprache als eine Programmiersprache ist. Ein Programm, das in Pseudocode geschrieben ist, kann problemlos in jede Programmiersprache übersetzt werden.

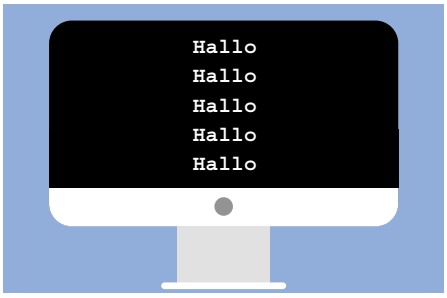
Prozedurale Programmierung

Die prozedurale Programmierung teilt Programme in kleine Einheiten (Prozeduren oder Funktionen), die für bestimmte Aufgaben verantwortlich sind. Sind diese Prozeduren einmal geschrieben und getestet, dann können sie immer wieder benutzt werden – das spart Entwicklungszeit und führt auch zu einer besseren Lesbarkeit des Programms.

Beispiel:

```
PROZEDUR Ausgabe
    SCHREIBE AUF BILDSCHIRM "Hallo"
ENDE

FÜR Var := 1 BIS 5 MIT SCHRITTWEITE 1
    AUFRUF Ausgabe
```



Das Beispiel in Pseudocode zeigt eine Prozedur mit dem Namen **Ausgabe**. Diese Prozedur hat eine Anweisung, die das Wort „Hallo“ auf den Bildschirm schreibt. Die bereits bekannte Wiederholung aus dem Beispiel vorher läuft dann 5-mal und ruft jedes Mal die Prozedur Ausgabe auf. Damit steht 5-mal das Wort „Hallo“ auf dem Bildschirm.

Objektorientierte Programmierung

Die objektorientierte Programmierung möchte Objekte der realen Welt in einem Programm abbilden. Damit sollen Problemstellungen aus beliebigen Bereichen (Geschäftsprozesse, wissenschaftliche Untersuchungen usw.) geeigneter als mit den anderen Programmierparadigmen in Programme umgesetzt werden können.

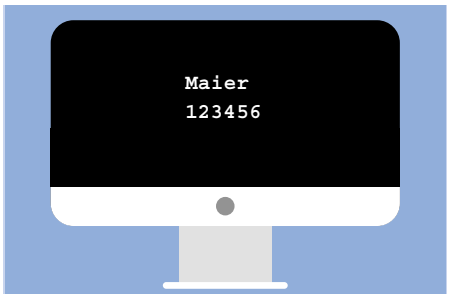
Im Mittelpunkt der objektorientierten Programmierung steht die **Klasse**, aus der dann konkrete Objekte gebildet werden. Diese Objekte haben bestimmte Eigenschaften (Attribute) und so genannte Methoden, mit denen diese Eigenschaften beispielsweise verändert werden können.

Beispiel:

```
KLASSE Kunde
   Name
   Telefon
ENDE

BILDE OBJEKT K1 VON Kunde
K1.Name    := "Maier"
K1.Telefon := "123456"

SCHREIBE AUF BILDSCHIRM K1.Name und K1.Telefon
```



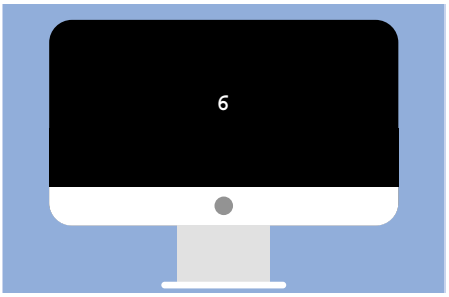
In dem Beispiel wird ein Klasse Kunde definiert. Von dieser Klasse können dann konkrete Objekte wie **K1** (für Kunde 1) gebildet werden. Die Eigenschaften des Objektes (Name, Telefon) können dann mit Werten belegt werden. In diesem Beispiel erhält das Objekt **K1** den Namen „Maier“ und die Telefonnummer „123456“. Anschließend werden Name und Telefon des Objektes auf den Bildschirm geschrieben.

Funktionale Programmierung

In der funktionalen Programmierung basiert alles auf Funktionen. Berechnungen werden mithilfe von Funktionen definiert, die wiederum Funktionen aufrufen oder deren Rückgabewerte nutzen. Funktionen können auch als anonym definiert werden und können damit sehr flexibel eingesetzt werden. Auch Elemente der strukturierten Programmierung wie die Iteration werden durch Funktionen ersetzt – in diesem Fall müssen dann so genannte rekursive Funktionen genutzt werden (das sind Funktionen, die sich selbst aufrufen).

Beispiel:

```
FUNKTION ggT (Übergabewerte x, y)
    FALLS x = y
        RÜCKGABE von: x
    SONST
        FALLS x < y
           RÜCKGABE von: ggT (x, y-x)
        SONST
           RÜCKGABE von: ggT (x-y, y)
ENDE
SCHREIBE AUF BILDSCHIRM ggT (12, 42)
```



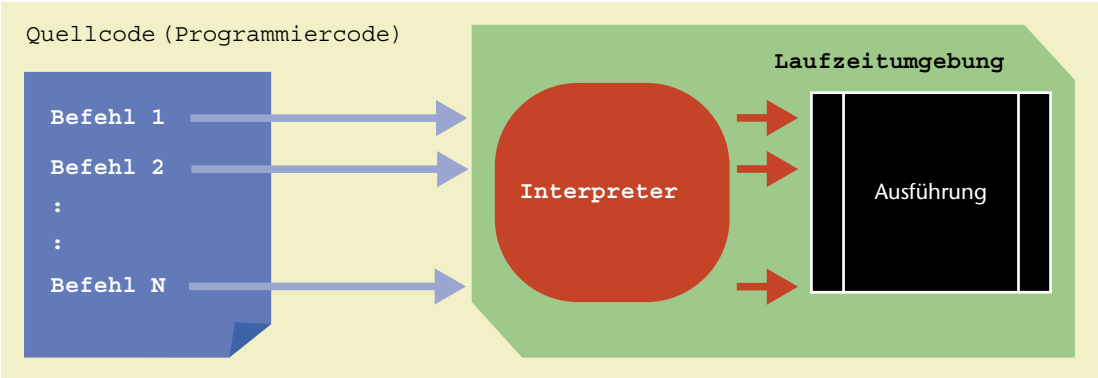
In dem Beispiel wird eine Funktion **ggT** definiert, die den größten gemeinsamen Teiler von zwei Zahlen berechnet. Dabei ruft die Funktion sich selbst auf und übergibt angepasste Zahlen, je nachdem, ob die erste übergebene Zahl größer oder kleiner der zweiten übergebenen Zahl ist. Das entspricht dem euklidischen Algorithmus, der von dem berühmten Mathematiker Euklid stammt (griechischer Mathematiker, 3. Jahrhundert vor Christus). Ein solches Prinzip nennt sich ***Rekursion.***

1.4 Interpretersprache vs. Compilersprache

Die meisten Programmiersprachen lassen sich in die beiden Bereiche Interpreter- oder Compilersprache einteilen. R ist eine Interpretersprache. Zum besseren Verständnis werden beide Möglichkeiten dargestellt.

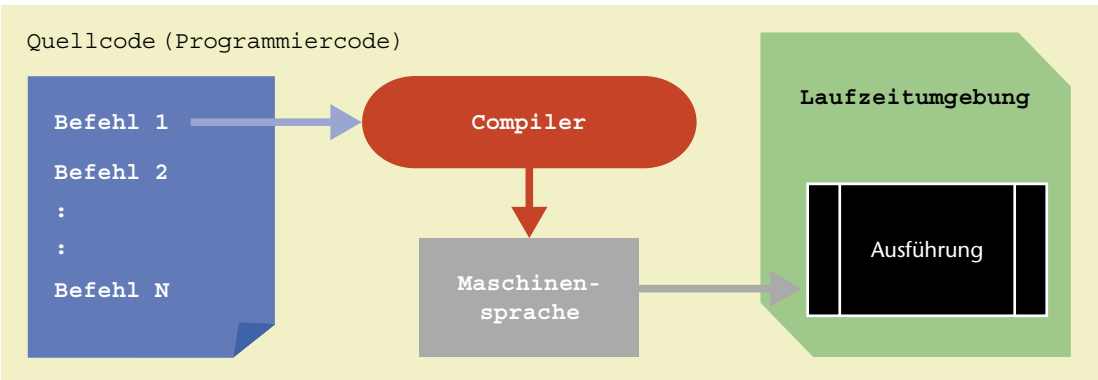
Interpretersprache

Eine Interpretersprache ist definiert als eine Sprache, die von einem Interpreter Zeile für Zeile übersetzt und auf der jeweiligen Plattform ausgeführt wird. Bildlich kann das so dargestellt werden:

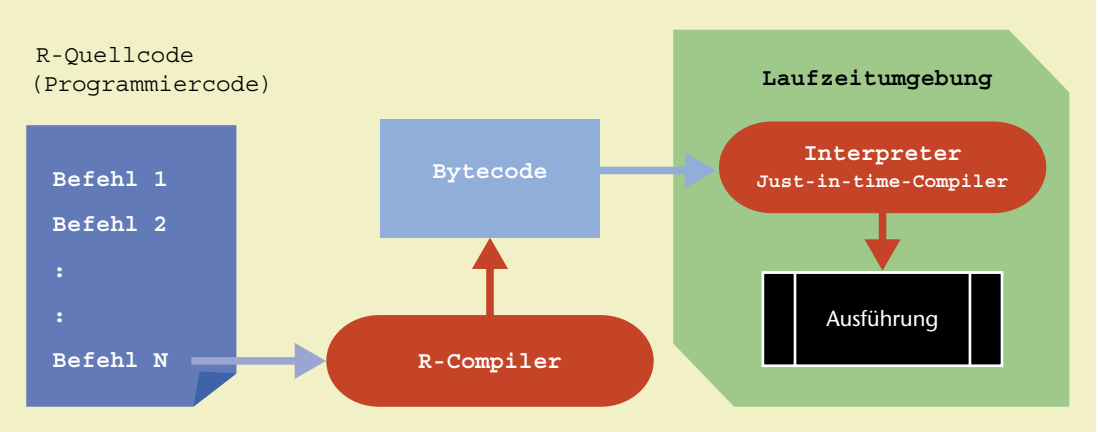


Compilersprache

Bei einer Compilersprache wird der Quellcode zuerst von einem Compiler in Maschinensprache der jeweiligen Plattform übersetzt. Diese Maschinensprachen-Programm kann dann direkt auf dieser Plattform ausgeführt werden. Bildlich kann das so dargestellt werden:



Seit 2010 gibt es in R zusätzlich die Möglichkeit, den R-Quellcode mit einem Compiler in einen Zwischencode (Bytecode) zu übersetzen, der dann in der entsprechenden Laufzeitumgebung interpretiert und ausgeführt wird (durch einen so genannten Just-in-time-Compiler). Das Prinzip findet sich auch bei Programmiersprachen wie Java oder C#, die ebenfalls zuerst in einen Zwischencode (Bytecode, Intermediatecode) übersetzt werden.



Der Vorteil bei dieser Vorgehensweise ist ein sehr kompakter und schnell ausführbarer Zwischencode.

1.5 Pakete der Sprache R

R ist eine Programmiersprache, die vor allem für statistische Berechnungen und deren grafische Aufbereitung eingesetzt werden kann. Die Sprache hat eine sehr umfangreiche Bibliothek von Zusatzpaketen (***CRAN***), die ständig weiterentwickelt werden und kostenfrei installierbar sind. Damit ist R perfekt für alle Bereiche des Data Sciences geeignet (wie die Datenanalyse oder auch das Machine-Learning).

Die folgende Auflistung zeigt einige wichtige R-Pakete mit Beschreibung:

Paket	Beschreibung
Standardpaket (vorinstalliert)	Das Standardpaket von R beinhaltet schon eine Vielzahl von Funktionalitäten zu statistischen Berechnungen und grafischen Möglichkeiten.
Paket *dplyr*	Dieses Paket bietet hervorragende Möglichkeiten der Datenmanipulation und Datenaufbereitung. Dabei wird eine eigene Grammatik verwendet.
Paket *ggplot2*	Sehr umfangreiche und oft genutzte Bibliothek zur Grafikerstellung unter R.
Paket *caret*	Dieses Paket beinhaltet Funktionalitäten zur Klassifizierung und zur Regressionsberechnung im Sinne des Machine-Learnings.
Paket *DBI*	Ermöglicht die Anbindung von R an relationale Datenbanken.
Paket *readxl*	Mit diesem Paket ist der Zugriff auf Excel-Tabellen möglich.
Paket *stringr*	Das Paket dient zur Zeichenkettenbearbeitung (inkl. regulärer Ausdrücke).
Paket *rgl*	Mit diesem Paket sind 3-D-Visualisierungen möglich. Es nutzt OpenGL und ist in C++ implementiert.

1.6 Der R-Interpreter

1.6.1 Einen R-Interpreter online nutzen

Das Besondere an R ist die Möglichkeit, sofort mit der Nutzung zu starten. Da R eine Interpretersprache ist, braucht es keinen grundsätzlichen Aufbau eines Programmes wie in den Sprachen C++ oder Java, sondern die Eingabe eines einzigen Befehls wird sofort interpretiert und ausgeführt. Natürlich können mit R auch größere und komplexe Programme wie mit den anderen modernen Programmiersprachen geschrieben werden. Dazu wird aber in der Regel eine Entwicklungsumge-

bung genutzt, die alle Quellcode-Dateien verwaltet und auch die Übersetzung und Ausführung übernimmt. Eine solche Entwicklungsumgebung wird in dem nächsten Unterkapitel vorgestellt. Allerdings soll nun an erster Stelle die einfache Möglichkeit gezeigt werden, wie R-Befehle (oder auch ein ganzes R-Programm) mit einem Online-Interpreter direkt ausgeführt werden können. Es gibt eine Vielzahl von solchen Interpretern, hier wird der Interpreter der Webseite *„https://www.w3schools.com“* genutzt. Diese Webseite bietet eine Vielzahl von kostenfreien E-Learning-Angeboten und auch verschiedene Online-Compiler und -Interpreter:

Nach dem Aufruf des Online-Interpreters können im linken Bereich R-Befehle oder Ausdrücke (beispielsweise mathematische Berechnungen) eingegeben werden. Nach dem Klick auf „Run“ werden die Eingaben interpretiert (ausgeführt) und im rechten Bereich angezeigt.

1.6.2 Erste Eingaben mit R

Die Möglichkeit dieser Online-Interpreter-Eingabe dient vor allem dem schnellen Testen von einzelnen oder wenigen R-Anweisungen. Die Eingabe kann aber auch als eine Art Taschenrechner-Ersatz dienen. Die folgenden Beispiele zeigen die Verwendung dieser einfachen Option:

Beispiel 1: Addition der Zahlen 10 und 20

Beispiel 2: Berechnung der Quadratwurzel von 9

Diese ersten Beispiele zeigen die Verwendung dieser Eingabemöglichkeit. Wenn das Wissen über die Programmiersprache R weiter fortgeschritten ist, dann ist der Einsatz dieser Möglichkeit deutlich interessanter.

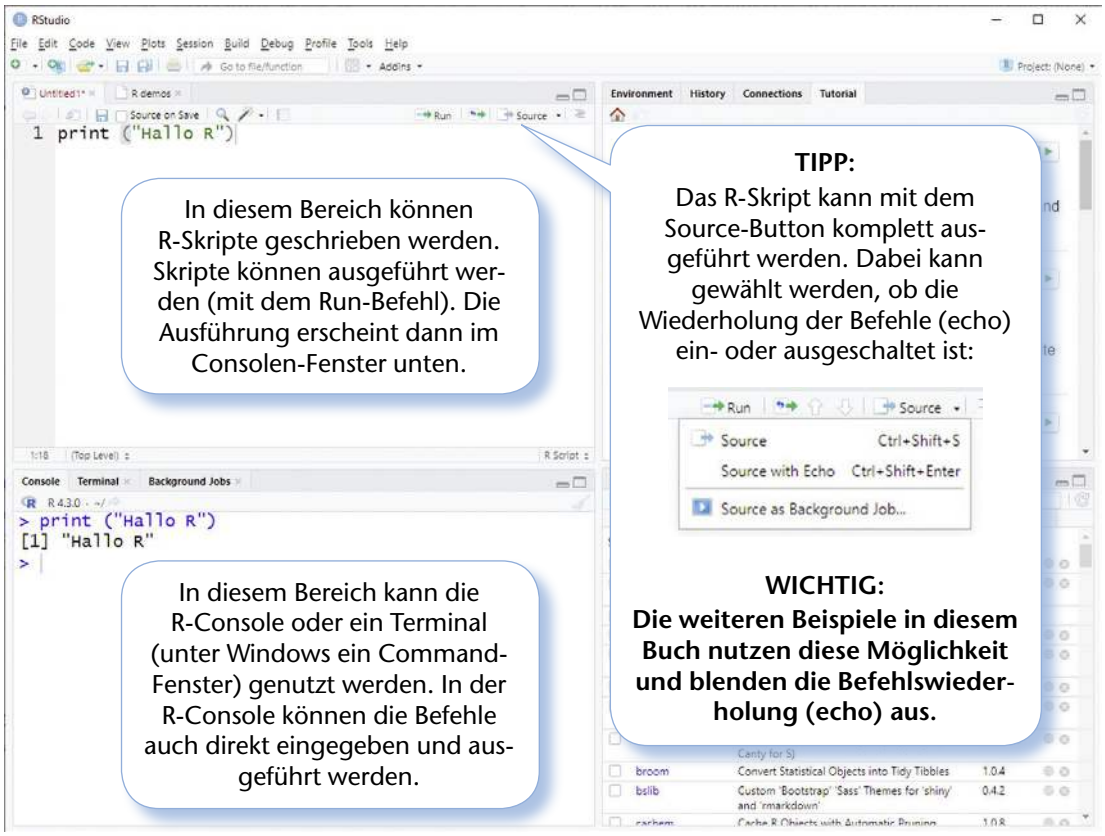
1.7 Einsatz einer Entwicklungsumgebung

1.7.1 RStudio

Die integrierte Entwicklungsumgebung ***RStudio*** ist eine komfortable Umgebung, um R-Programme zu entwickeln. Besonders erfreulich ist der Umstand, dass die Umgebung kostenfrei im Internet bereitsteht. Das ***RStudio*** bietet die Möglichkeiten R-Skripte zu verwalten, Kommandozeilen einzugeben sowie weitere Tools (wie eine Paketverwaltung) zu nutzen.

Installieren von RStudio:

- Die Installationspakete von ***R*** und ***RStudio*** können kostenfrei vom Hersteller der Software heruntergeladen und installiert werden (https://posit.co/downloads/):
- Starten Sie ***RStudio*** nach der Installation:



1.7.2 Das erste R-Programm

Da R als Skriptsprache kein Programmgerüst braucht, um eine erste Anweisung auszuführen, ist das erste R-Programm sehr einfach:

```
print("Das erste R-Programm")
```

Nach dem Starten erscheint das folgende Ausgabe im ***RStudio:***

Console
[1] "Das erste R-Programm"

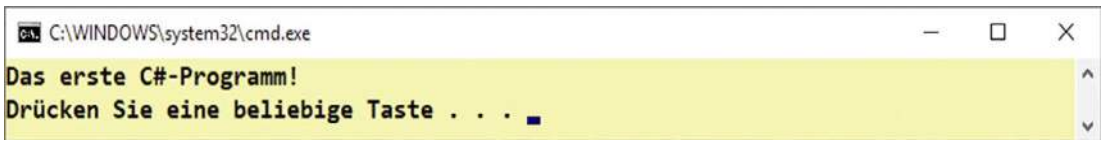
Der **`print`**-Befehl bzw. die **`print`**-Funktion schreibt (wie der Name schon andeutet) einen Inhalt auf den Bildschirm.

Zum Vergleich wird hier das erste Programm in der Programmiersprache C# gegenübergestellt.

```
using System;
using System.Collections.Generic; using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Erstes_Programm
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            System.Console.WriteLine("Das erste C#-Programm!");
        }
    }
}
```

Nach dem Starten erscheint dieses Konsolenfenster:



Es ist erkennbar, dass für das erste C#-Programm deutlich mehr Aufwand betrieben werden muss. In anderen Sprachen wie Java oder C++ ist es ähnlich. Das zeigt die Einfachheit einer Sprache wie R.

1.8 Grundlegende Konventionen in R

1.8.1 Schlüsselworte in R

Ein R-Programm ist nichts Anderes als eine oder mehrere R-Anweisungen, die hintereinander abgearbeitet werden. Dabei gibt es Möglichkeiten, die Reihenfolge diese Abarbeitung zu steuern (mit Kontrollstrukturen wie der Selektion und Iteration) oder Teile der Anweisungen in Funktionen zu definieren. Diese Themen werden im Laufe des Buches ausführlich beschrieben. Die Grundlage aller dieser Möglichkeiten sind die Schlüsselworte der Programmiersprache R. Es sind klar definierte Begriffe, die nur für den vorgesehenen Zweck eingesetzt werden dürfen.

R Schlüsselworte:

`if`	`else`	`repeat`	`while`	`function`	`for`
`next`	`break`	`TRUE`	`FALSE`	`NULL`	`Inf`
`NaN`	`NA`	`NA_integer_`	`NA_real_`	`NA_complex_`	`NA_character_`

Insgesamt hat R 18 Schlüsselworte. Das sind im Vergleich sehr wenige Schlüsselworte. Allerdings lebt die Programmiersprache R auch eher von den unzähligen Funktionen aus den Bibliotheken und dient nicht dazu, Anwendungen mit Oberflächen zu entwickeln.

Die folgende Tabelle zeigt andere Programmiersprachen im Vergleich:

Programmiersprache	Anzahl der Schlüsselworte
Python	ca. 35 Schlüsselworte
C	ca. 40 Schlüsselworte
C#	ca. 80 Schlüsselworte
Java	ca. 50 Schlüsselworte
C++	ca. 80 Schlüsselworte
PHP	ca. 70 Schlüsselworte
Javascript	ca. 45 Schlüsselworte

Hinweis:

In einem R-Programm wird die Groß- und Kleinschreibung beachtet. Die folgenden beiden Anweisungen sind also unterschiedlich:

`if` → `das ist die Einleitung einer Selektion in R`

`IF` → `das ist keine Einleitung einer Selektion, eventuell könnte es als Name für ein Objekt benutzt werden, was aber nicht sinnvoll wäre`

1.8.2 Bezeichner (Namen) in R

Wie in allen Programmiersprachen gibt es in R Namen für Objekte, Konstanten, Funktionen und Klassen. Diese selbst gewählten bzw. definierten Namen unterliegen einer gewissen Konvention, die unbedingt eingehalten werden muss:

- Das erste Zeichen muss ein Buchstabe sein (ein Punkt ist auch erlaubt `'.'`).
- Der Rest kann aus Ziffern, Buchstaben oder einem Unterstrich `'_'` gestaltet werden.
- Der Bezeichner darf nicht mit einem Schlüsselwort von R übereinstimmen.

Beispiel:

- `Zahl_1` gültiger Bezeichner
- `_zahl` kein gültiger Bezeichner (erstes Zeichen ist ein Unterstrich)
- `2mal7` kein gültiger Bezeichner (erstes Zeichen ist eine Ziffer)
- `break` kein gültiger Bezeichner (Schlüsselwort in R)
- `zahl 1` kein gültiger Bezeichner (Leerzeichen nach zahl)
- `zahl.1` gültiger Bezeichner
- `.zahl1` gültiger Bezeichner (***ACHTUNG:*** bei einem vorangestellten Punkt wird das Objekt „unsichtbar gespeichert" und kann nicht im Arbeitsbereich (Workspace) von R angezeigt werden).

Hinweis:

R unterstützt auch Unicode-Zeichen. Damit wären auch folgende Namen gültige Bezeichner:

- `länge` Sonderzeichen sind auch erlaubt
- `ΔΕΙΤα` auch das griechische Alphabet ist erlaubt

Die Benennung von Bezeichnern ist dem Programmierer eigentlich freigestellt. Allerdings hat es sich als sinnvoll erwiesen, dass in einem Programm von Anfang an eine bestimmte Konvention eingehalten wird. Eine dieser Konventionen sieht vor, dass der Bezeichner immer mit einem Kleinbuchstaben anfängt und weitere Ergänzungen mit einem Unterstrich verbunden werden. Weiterhin sollen die Bezeichner aussagekräftig sein. Klassen sollen hingegen mit einem Großbuchstaben eingeleitet werden und keine Unterstriche enthalten – Ergänzungen sollen einfach wieder mit einem Großbuchstaben beginnen (auch bekannt als Camel-Case-Notation). Die folgenden Beispiele verdeutlichen diese Konvention:

Beispiele:

Namen für Objekte:

- `kapital_neu` ein Name, der auf einen Kapitalwert verweist
- `x_koord` ein Name, der auf eine X-Koordinate verweist

Funktionsnamen:

- `auswertung_daten` eine Funktion, die Daten auswertet
- `lösche_wert` eine Funktion, die einen bestimmten Wert löscht

Klassennamen:

- `MeineKlasse` eine eigene Klasse
- `DatenbankTest` eine Klasse, die Datenbanken testet

1.8.3 Aufbau von R-Programmen und Trennzeichen

Ein R-Programm besteht aus einer oder mehreren Anweisungen, die hintereinander (sequenziell) abgearbeitet werden. Die erste Anweisung wird auch zuerst ausgeführt. Die einzelnen Anweisungen stehen in der Regel in einer eigenen Zeile und werden Schritt für Schritt abgearbeitet. Werden Funktionen definiert, so werden diese erst ausgeführt bzw. genutzt, wenn entsprechende Anweisungen im R-Programm vorliegen. Die folgenden Beispiele zeigen deshalb erst einmal die schematische Art der Ausführung. Die einzelnen Bestandteile wie Funktionen werden nur ansatzweise dargestellt, da diese Themen in den fortgeschrittenen Kapiteln ausführlich behandelt werden. Es geht also erst einmal nur um das grundsätzliche Prinzip.

Beispiel 1: Ein erstes R-Programm, welches drei Anweisungen hat.

```
print ("Das")
print ("erste")
print ("R-Programm")
```

Console

```
[1] "Das"
[1] "erste"
[1] "R-Programm"
```

Hinweis:

Die Bedeutung der vorangestellten Klammer [1] wird beim Thema Vektoren behandelt.

Nach dem Starten werden alle drei Anweisungen hintereinander ausgeführt. Die Anweisungen werden dabei durch Zeilenumbrüche (jede Anweisung in einer Zeile) unterteilt.

Beispiel 2: Das erste R-Programm mit drei Anweisungen in einer Zeile.

```
print ("Das") ; print ("erste") ; print ("R-Programm")
```

Console

```
[1] "Das"
[1] "erste"
[1] "R-Programm"
```

Nach dem Starten werden alle drei Anweisungen hintereinander ausgeführt. Die Anweisungen sind nur durch Semikolons getrennt. Für die Leserlichkeit eines Programmes ist das aber nicht zu empfehlen.

Beispiel 3: Ein gültiges R-Programm, das nur eine Funktion definiert.

```
ausgabe <- function()
{
   print ("Eine R-Funktion")
}
```

Definition einer Funktion

Console

Nach dem Starten passiert nichts. Es wurde zwar ein gültiges R-Programm geschrieben, aber es gibt keine Anweisung, die ausgeführt werden kann - die Funktion müsste dazu aufgerufen werden.

Beispiel 4: Ein gültiges R-Programm, das eine Funktion definiert, aufruft und weitere Anweisungen hat.

```
ausgabe <- function()
{
   print ("Eine R-Funktion")
}

print("Aufruf der Funktion:")
ausgabe()
```

Definition einer Funktion

Anweisungen, die hintereinander ausgeführt werden

Nach dem Starten werden zwei Anweisungen hintereinander ausgeführt. Zuerst wird eine Bildschirmausgabe mit print erzeugt und danach die Funktion aufgerufen.

Console

```
[1] "Aufruf der Funktion:"
[1] "Eine R-Funktion"
```

1.8.4 Kommentare

Kommentare werden bei der Übersetzung des R-Programmes ignoriert, sie dienen also nur dem Verständnis des Quellcodes – jeder Programmierer weiß, wie wichtig Kommentare in einem Programm sind, vor allem wenn Monate später der Quellcode modifiziert werden soll und sich niemand mehr erinnert, welche Bedeutungen die Funktionen oder Programmabschnitte hatten.

Ein Kommentar muss mit dem **#**-Zeichen eingeleitet werden. In der Entwicklungsumgebung werden Kommentare in der Regel in grüner Farbe angezeigt.

Beispiel:

```
# Jetzt folgt eine Bildschirmausgabe
print("Hallo")
```

Hinweis:

Das Kommentieren von Quellcode ist wichtig, gerade für die Wiederverwendbarkeit und den übersichtlichen Aufbau des Quellcodes. Es sollte aber nicht übertrieben werden. Selbstverständliche Sachverhalte sollten nicht zusätzlich kommentiert werden. Kurz und präzise ist wichtiger, als ausladende Erklärungen abzugeben. Es kann sinnvoll sein, zu Beginn des Quellcodes einen Kommentarkopf anzulegen, der über den Quellcode bzw. das Programm informiert. Ebenso könnten Programmabschnitte mit einem Kommentar eingeleitet werden.

Beispiel:

```
# Programmname: TESTXY
# Version: 1.0
# Autor: Hardy
# letzte Überarbeitung: 15. Mai
```

2 Objekte

2.1 Objekte

Eine elementare Aufgabe von Programmen ist die Speicherung und Verarbeitung von Daten. Die wenigsten Programme beschränken sich auf reine Bildschirmausgaben. Das eigentliche Programmieren beginnt mit der Möglichkeit, Werte über die Tastatur oder aus Dateien und Datenbanken einzulesen und geeignet zu verarbeiten. Die folgenden Ausführungen vermitteln die Grundlagen für diese Prozesse.

2.1.1 Einfache Objekte in R

In den gängigen Programmiersprachen dienen Variablen dazu Werte zu speichern. Sie sind also Platzhalter mit einem festen Namen, einem festen Speicherplatz und einem Inhalt (Wert). In den meisten Programmiersprachen (z. B. C++) müssen Variablen angelegt (deklariert) werden, bevor sie benutzt werden können. Dabei wird auch festgelegt, welche Art von Werten die Variable speichern kann (beispielsweise ganzzahlige Werte, Fließkommazahlen oder auch Zeichenketten). Man spricht dabei vom Datentyp der Variable. Dieser Datentyp bestimmt ganz genau den Wertebereich und die Art des Wertes einer Variablen. **In R gibt es keine Variablen in diesem Sinne. In R werden Werte (inklusive ihrem Datentyp, deshalb spricht man auch von Objekten) im Speicher angelegt und mit einem Namen verbunden.** Dieser Name kann aber auch im Laufe des Programms auf einen anderen Wert im Speicher verweisen – es gibt also keine feste Bindung des Namens an einen Speicherplatz wie bei der klassischen Variable. **Die Verbindung eines Objektes (Wertes) mit einem Namen geschieht durch eine Zuweisung.** Wird einem Namen während des Programms ein neues Objekt zugewiesen, so ändert sich nicht der Name, sondern er verweist einfach auf ein anderes Objekt im Speicher. Das macht die Verwendung von Namen enorm flexibel. In einer Programmiersprache wie C++ ist das nicht möglich – hier behalten Namen (Variablen) ihren einmal festgelegten Datentyp. Die folgenden Beispiele zeigen, wie Objekte in R angelegt werden können:

Beispiel 1: In einem R-Programm werden drei verschiedene Objekte angelegt. Sie erhalten einen Wert durch eine Zuweisung mit dem Zuweisungsoperator „`<-`“ (dazu später mehr im Kapitel Operatoren)

```
x <- 1
anfangs_kapital <- 1575.75
nach_name <- "Maier"
```

- Dem Objekt **`x`** wird der ganzzahlige Wert 1 zugewiesen. Damit hat dieses Objekt den Datentyp „Ganzzahl“.
- Dem Objekt **`anfangs_kapital`** wird der Fließkommawert **`1575.75`** zugewiesen. Damit hat dieses Objekt den Datentyp „Fließkommazahl“.
- Dem Objekt **`nach_name`** wird die Zeichenkette **`"Maier"`** zugewiesen. Damit hat dieses Objekt den Datentyp „Zeichenkette“ oder „String“.

Hinweise:

- Die Begriffe „Ganzzahl“, „Fließkommazahl“ und „Zeichenkette“ werden im nächsten Unterkapitel (elementare Datentypen) ausführlich behandelt.
- Die Werte **`1`**, **`1575.75`** und **`"Maier"`** werden als ***Literale*** bezeichnet. Literale sind Zeichenfolgen, die in Programmiersprachen in Werte von elementaren Datentypen übersetzt werden.

Beispiel 2: Ein Name erhält in einem R-Programm hintereinander verschiedene Werte (Objekte) zugewiesen.

```
flexibel <- 1
flexibel <- 1575.75
flexibel <- "Maier"

print(flexibel)
```

Nach dem Starten werden die Objekte hintereinander zugewiesen. Am Ende „behält" das Objekt **`flexibel`** den Inhalt „Maier", wie die Ausgabe von **`flexibel`** zeigt.

```
Console
[1] "Maier"
```

2.1.2 Elementare Datentypen

Ganzzahlen (engl. Integer)

Ganzzahlige Werte zeichnen sich dadurch aus, dass sie keine Nachkommstellen haben. Sie entsprechen damit einer Teilmenge der ganzen Zahlen aus der Mathematik. Sie werden beim Übersetzungsvorgang in eine duale Zahl (Binärzahl) übersetzt. Eine duale Zahl besteht nur aus den Ziffern 1 und 0 und ist deshalb perfekt geeignet, um in einem Computer gespeichert zu werden (der Computer kann genau diese Zustände ***1*** = ***„Strom an"*** sowie ***0*** = ***„Strom aus"*** speichern). Ganzzahlige Werte werden in R als Zahlen ohne Nachkommstellen angegeben. Zusätzlich muss am Ende der Zahl ein „L" (für Long Integer = große Ganzzahl) angefügt werden. Ansonsten würde R die Zahl als eine Fließkommazahl ohne Nachkommastellen einstufen.

Beispiel:

```
x <- 10
class(x) # Die Funktion class liefert den Datentyp des Objektes
x <- 10L
class(x)
```

```
Console
[1] "numeric"
[1] "integer"
```

Hinweis:

Der Datentyp „integer" ist 32-Bit groß. Damit lassen sich Werte von **`-2147483647`** bis **`+2147483647`** darstellen.

Nach dem Starten zeigt sich, dass der Datentyp bei der ersten Zuweisung vom Typ **„`numeric`"** (Fließkommazahl) und bei der zweiten Zuweisung von Typ **„`integer`"** (Ganzzahl) ist – das liegt an dem nachgestellten „L".

Fließkommzahlen (engl. Floating point)

Fließkommazahlen (auch Gleitkommazahlen genannt) repräsentieren eine Teilmenge der reellen Zahlen und können auch Nachkommstellen darstellen. Sie wurden für die Computerarithmetik erfunden, damit auch Berechnungen mit Nachkommastellen möglich sind. Der Begriff Fließkomma bedeutet dabei, dass eine reelle Zahl mit endlich vielen Nachkommastellen in die Form einer Darstellung mit einer sogenannten ***Mantisse*** und einem ***Exponenten*** gebracht wird. Dabei verschiebt sich das Komma entsprechend. Das folgende Beispiel zeigt diese Vorgehensweise:

Beispiel:

Gegeben ist die Zahl: $2734{,}129$

$$= 273{,}4129 * 10^{1}$$
$$= 27{,}34129 * 10^{2}$$
$$= 2{,}734129 * 10^{3}$$

Mantisse Exponent

Das Komma „fließt“ immer weiter nach links, desto höher der Exponent wird. Damit der Computer mit diesen Zahlen rechnen kann, wird jede Fließkommazahl zuerst normiert – das bedeutet, dass die Mantisse beispielsweise zwischen 1 und 10 liegen muss. Der Exponent passt sich entsprechend an. Mit diesen normierten Werten kann der Computer dann die so genannte Fließkomma-Arithmetik durchführen – also rechnen. In den Programmiersprachen wird das Komma allerdings durch einen Punkt repräsentiert, da die meisten Programmiersprachen die englische bzw. amerikanische Darstellung implementiert haben.

Beispiel:

```
x <- 10.123
print(x)
```

Console

```
[1] 10.123
```

Hinweis:

Der Datentyp „numeric“ als Fließkommazahl ist 64-Bit groß und vergleichbar mit dem **`double`**-Datentyp aus anderen Programmiersprachen.

Mithilfe eingebauter Konstanten (**`.Machine$integer.max`** und **`.Machine$double.xmax`**) können die Maximalwerte der Datentypen angezeigt werden, wie das folgende Beispiel zeigt.

Beispiel:

```
print(.Machine$integer.max)
print(.Machine$double.xmax)
```

Console

```
[1] 2147483647
[1] 1.797693e+308
```

Hinweis:

R zeigt zwar als Datentyp „numeric“ an, intern wird die Fließkommazahl aber auch mit **`double`** bezeichnet.

Zeichenketten (engl. strings)

Zeichenketten sind einfach eine Reihe von Zeichen, die durch einfache oder doppelte Hochkommata begrenzt werden. Innerhalb dieser Begrenzung sind alle Zeichen des jeweiligen Zeichensatzes erlaubt.

Beispiel:

```
kette <- "Hallo R"
class(kette)
nchar(kette)
```

Console

```
[1] "character"
[1] 7
```

Hinweis:

R zeigt als Datentyp „character“ für eine Zeichenkette an und die Anzahl der Zeichen in einer solchen Zeichenkette kann mit der Funktion **`nchar()`** ermittelt werden.

Wahrheitswerte (engl. boolean)

Ein Wahrheitswert ist der einfachste Datentyp in R. Er kennt nur zwei Zustände (wahr oder falsch). In der Sprache R durch die Schlüsselworte **`TRUE`** bzw. **`FALSE`** ausgedrückt.

Beispiel:

```
wahrheitswert <- FALSE
class(wahrheitswert)
print(wahrheitswert)
wahrheitswert <- TRUE
print(wahrheitswert)
```

```
Console
[1] "logical"
[1] FALSE
[1] TRUE
```

Hinweis:

R zeigt als Datentyp „logical“ für den Wahrheitswert an. Die Zustände sind entweder `TRUE` oder `FALSE`.

2.1.3 Einfache Operationen auf elementaren Datentypen

Die arithmetischen Operatoren (`+,-,/,*`) sind aus dem Mathematikunterricht bekannt. Sie können zum Addieren, Subtrahieren, Dividieren und Multiplizieren verwendet werden. Diese Operationen können natürlich auch mit Werten in R durchgeführt werden. Mithilfe des Zuweisungsoperators „`<-`“ werden dann Werte bzw. Ergebnisse einer Berechnung dem Objekt zugewiesen. Die folgenden Beispiele zeigen die Verwendung dieser Operatoren:

Beispiel 1: Summe und Produkt von zwei Ganzzahlen

```
x <- 10
y <- 20
summe <-  x + y       # summe erhält den Wert 30
produkt <- x * y      # produkt erhält den Wert 200
```

Beispiel 2: Division von Fließkommazahlen

```
x <- 2.4
y <- 1.2
quotient <- x / y     # quotient erhält  den Wert 2.0
```

2.2 Einfache Ein- und Ausgabe in R

Damit eine einfache Interaktion zwischen Benutzer und R-Programm möglich ist, werden nun die Möglichkeiten der Ausgabe auf dem Bildschirm mit **`print`** und das Einlesen über die Tastatur mit **`readline`** dargestellt. **`Print`** und **`readline`** sind Funktionen, die Inhalte entgegennehmen und auf dem Bildschirm darstellen können oder eingelesene Werte über die Tastatur zu Verfügung stellen. Beides sind Funktionen, deren Arbeitsweise noch deutlicher wird, wenn das Kapitel über Funktionen behandelt wurde.

2.2.1 Ausgabe mit print

Die Funktion print erwartet einen Inhalt, den sie auf dem Bildschirm ausgeben kann. Dieser Inhalt wird innerhalb runder Klammern angegeben und kann entweder ein Literal oder ein Objekt sein. Auch das Ergebnis einer Operation (wie die Summe von zwei Ganzzahlen ist erlaubt, da es nichts Anderes als ein Ganzzahl-Literal ist). Sollen allerdings mehr als ein Objekt oder Literal ausgegeben werden, so müssen diese Inhalte mit einer weiteren Funktion erst zusammengefügt werden. Diese Funktion lautet **`paste`** oder **`paste0`**. Die folgenden Beispiele zeigen die Einsatzmöglichkeiten von **`print`** und **`paste`**:

Beispiel 1: Ausgabe von Literalen

```
print(10)
print(12.333)
print("Hallo")
```

```
Console
[1] 10
[1] 12.333
[1] "Hallo"
```