Einführung in R

Clemens Brunner

18.-19.10.2019

Vektoren

Erstellen von Vektoren

Der atomare (einfachste) Datentyp in R ist der Vektor. Ein Vektor besteht aus einem oder mehreren Elementen. Demnach ist also auch eine einfache Zahl in Wirklichkeit ein Vektor (mit einem Element). Ein Vektor mit mehreren Elementen wird mit der Funktion `c` erzeugt (steht für combine oder concatenate).

```
y <- c(1, 2, 3.1415, -100)
y
```

```
[1]    1.0000    2.0000    3.1415 -100.0000
```

Die Länge eines Vektors (also die Anzahl der Elemente des Vektors) kann mit der Funktion `length` ausgegeben werden.

```
length(y)
```

```
[1] 4
```

```
a <- 6
length(a)
```

```
[1] 1
```

Mit `c` können dementsprechend auch Vektoren beliebiger Längen miteinander kombiniert werden - es werden alle Elemente zu einem Vektor zusammengefügt.

```
c(666, y, 666, c(23, 24))
```

```
[1]  666.0000    1.0000    2.0000    3.1415 -100.0000  666.0000   23.0000   24.0000
```

Typen von Vektoren

Ein Vektor besteht aus Elementen desselben Typs (man nennt Vektoren daher homogene Datentypen). Bis jetzt haben wir schon numerische Vektoren kennengelernt, das sind Vektoren deren Elemente Zahlen sind. Es gibt aber auch Vektoren, die aus logischen Elementen oder aus Zeichenketten/Buchstaben bestehen. Man unterscheidet in R daher grob zwischen folgenden Typen:

- Numerische Vektoren
- Logische Vektoren
- Zeichenkettenvektoren

Numerische Vektoren

Numerische Vektoren sind vom Typ `numeric` und beinhalten ausschließlich Zahlen.

```
c(2, 13, 15, 17)
```

```
[1]  2 13 15 17
```

Mit der Funktion `class` können Sie den Typ (die Klasse) eines Objektes (welches man als Argument an die Funktion übergibt) bestimmen:

```r
class(2)
```

```
[1] "numeric"
```

```r
z <- c(1.11, 2.33)
class(z)
```

```
[1] "numeric"
```

Logische Vektoren

Logische Vektoren bestehen aus den Werten `TRUE` oder `FALSE`. Sie sind vom Typ `logical`.

```r
class(TRUE)
```

```
[1] "logical"
```

```r
class(c(FALSE, FALSE, TRUE))
```

```
[1] "logical"
```

Es ist übrigens möglich, die Werte `TRUE` und `FALSE` mit `T` und `F` abzukürzen (in der Praxis sollte man auf diese Abkürzungen aufgrund der schlechteren Lesbarkeit aber verzichten).

Logische Vektoren entstehen unter anderem durch Vergleiche:

```r
x <- c(0.5, 55, -10, 6)
class(x)
```

```
[1] "numeric"
```

```r
x < 1
```

```
[1]  TRUE FALSE  TRUE FALSE
```

```r
class(x < 1)
```

```
[1] "logical"
```

In R gibt es folgende Vergleichsoperatoren: `>`, `>=`, `<`, `<=`, `==` (ist gleich) und `!=` (ist ungleich). Vergleiche können mit `|` (oder) und `&` (und) verknüpft und mit `!` negiert werden. Gruppierungen durch Klammersetzung ist ebenfalls möglich.

```r
(3 > 5) & (4 == 4)
```

```
[1] FALSE
```

```r
(TRUE == TRUE) | (TRUE == FALSE)
```

```
[1] TRUE
```

```r
((111 >= 111) | !(TRUE)) & ((4 + 1) == 5)
```

```
[1] TRUE
```

Achtung: Der Gleichheitsoperator lautet `==` (also zwei Gleichheitszeichen)!

Zeichenkettenvektoren

Zeichenkettenvektoren bestehen aus einzelnen Zeichenketten (`character`), welche in Anführungszeichen gesetzt werden. Es können sowohl einfache `'` als auch doppelte `"` Anführungszeichen verwendet werden. Eine Zeichenkette kann aus Buchstaben, Ziffern und Sonderzeichen bestehen.

```r
s <- c("What's", 'your', "name?")
s
```

```
[1] "What's" "your"   "name?"
```

```
class(s)
```

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

```
class("Hello!")
```

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

Die Funktion `length` gibt wie erwartet die Länge des Vektors aus und nicht die Anzahl der Zeichen einer Zeichenkette. Dafür kann man die Funktion `nchar` verwenden.

```
length(c('Hello', 'world!'))
```

```
[1] 2
```

```
nchar(c('Hello', 'world!'))
```

```
[1] 5 6
```

Zwang (Coercion)

Wie bereits erwähnt sind Vektoren homogene Datentypen, d.h. sie enthalten nur Elemente desselben Typs. Wenn man einen Vektor mit Elementen unterschiedlichen Typs erstellt, wird der Vektor automatisch in einen Typ gezwungen, welcher alle Elemente abbilden kann. Wenn man also Zahlen mit Zeichenketten mischt, werden alle Elemente in Zeichenketten "gezwungen" (da Zeichenketten im Allgemeinen nicht als Zahlen dargestellt werden können).

```
x <- c(1, 2.14, "5", 6)
x
```

```
[1] "1"    "2.14" "5"    "6"
```

```
class(x)
```

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

Rechnen kann man mit dem Vektor im obigen Beispiel nicht mehr, da die Elemente nun Zeichenketten und keine Zahlen mehr sind.

Man kann mit folgenden Funktionen auch explizit eine Umwandlung in einen gewünschten Typ erzwingen:

- `as.numeric`
- `as.integer`
- `as.logical`
- `as.character`

Folgendes Beispiel wandelt einen Zeichenketten-Vektor in einen numerischen Vektor um (dies funktioniert, weil alle Zeichenketten als Zahlen interpretiert werden können):

```
as.numeric(c("1", "2.12", "66"))
```

```
[1]  1.00  2.12 66.00
```

Wenn das nicht geht, wird eine Warnung ausgegeben und `NA` (steht für "Not Available", also fehlender Wert) für das Element, welches sich nicht umwandeln lässt, angenommen:

```
as.numeric(c("1", "2.12", "X"))
```

```
Warning: NAs introduced by coercion
```

```
[1] 1.00 2.12   NA
```

Rechnen mit Vektoren

Mit numerischen Vektoren kann man Rechenoperationen durchführen - diese werden stets *elementweise* angewendet.

```
y
```

```
[1]    1.0000    2.0000    3.1415 -100.0000
```

```
y * 100 + 2
```

```
[1]   102.00   202.00   316.15 -9998.00
```

Wie wir bereits wissen, gibt es die üblichen Operatoren `+`, `-`, `*`, und `/` für die Addition, Subtraktion, Multiplikation und Division. Das Zeichen ^ oder `**` steht für "hoch" (berechnet also die Potenz von einer Basis zum Exponenten). Der Operator `%/%` berechnet die ganzzahlige Division und `%%` berechnet den Rest. Weitere praktische Funktionen sind `abs` für den Betrag und `sqrt` für die Quadratwurzel einer Zahl. Die Funktionen `log` bzw. `exp` berechnen den (natürlichen) Logarithmus bzw. die Exponentialfunktion.

Recycling

Wenn zwei Vektoren unterschiedlich lang sind, dann wiederholt R die Werte des kürzeren Vektors, sodass es gleich viele Elemente gibt wie beim längeren Vektor. Man bezeichnet dies als Recycling. Dies ist z.B. auch schon der Fall, wenn man einen Vektor mit vier Elementen mit einem Skalar (Vektor mit einem Element) multipliziert, wie im folgenden Beispiel.

```
c(1, 2, 3, 4) * 2
```

```
[1] 2 4 6 8
```

Der skalare Vektor 2 wird automatisch auf den Vektor `c(2, 2, 2, 2)` erweitert, daher entspricht die obige Operation eigentlich folgender elementweiser Berechnung:

```
c(1, 2, 3, 4) * c(2, 2, 2, 2)
```

```
[1] 2 4 6 8
```

Weiteres Beispiel:

```
c(1, 2, 3, 4) + c(0, 10)
```

```
[1]  1 12  3 14
```

```
c(1, 2, 3, 4) + c(0, 10, 0, 10)
```

```
[1]  1 12  3 14
```

Wenn sich das Recycling nicht genau ausgeht, d.h. wenn die Länge des längeren Vektors kein ganzzahliges Vielfaches des kürzeren Vektors ist, dann funktioniert das Recycling zwar genauso, aber es wird eine Warnung ausgegeben:

```
c(1, 2, 3, 4) + c(0, 10, 8)
```

```
Warning in c(1, 2, 3, 4) + c(0, 10, 8): longer object length is not a multiple of shorter object
length
```

```
[1]  1 12 11  4
```

```
c(1, 2, 3, 4) + c(0, 10, 8, 0)
```

```
[1]  1 12 11  4
```

Erstellen von Zahlenfolgen

Vektoren mit definierten Zahlenfolgen erstellt man mit `:` oder mit `seq`. Bei ersterem ist die Schrittweite immer 1, bei letzterem kann diese angepasst werden.

```
1:20
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
pi:10
```

```
[1] 3.141593 4.141593 5.141593 6.141593 7.141593 8.141593 9.141593
```

```
9:2
```

```
[1] 9 8 7 6 5 4 3 2
```

```
seq(1, 20)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
seq(20, 1)
```

```
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

```
seq(0, 10, by=0.5)  # Schrittweite
```

```
 [1]  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0
[20]  9.5 10.0
```

```
seq(10, 0, by=-0.5)
```

```
 [1] 10.0  9.5  9.0  8.5  8.0  7.5  7.0  6.5  6.0  5.5  5.0  4.5  4.0  3.5  3.0  2.5  2.0  1.5  1.0
[20]  0.5  0.0
```

```
seq(0, 20, 2)  # gerade Zahlen
```

```
 [1]  0  2  4  6  8 10 12 14 16 18 20
```

```
seq(1, 20, 2)  # ungerade Zahlen
```

```
 [1]  1  3  5  7  9 11 13 15 17 19
```

```
seq(1, 3, length=10)  # Gesamtlänge des Ergebnisses
```

```
 [1] 1.000000 1.222222 1.444444 1.666667 1.888889 2.111111 2.333333 2.555556 2.777778 3.000000
```

Die Funktion `rep` wiederholt Werte:

```
rep(0, 90)
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[49] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Hier kann man auch die Bedeutung der Werte in eckigen Klammern erkennen, die vor jeder Ausgabezeile stehen: sie geben den Index des ersten Elements der Zeile an (also im Beispiel `[1]` für die erste Zeile und `[49]` für die zweite Zeile).

```
rep(c(0, 1, 2), times=10)
```

```
 [1] 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
```

```
rep(c(0, 1, 2), each=10)
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

```r
rep(c(0, 1, 2), times=c(10, 10, 10))  # gleichwertig mit each=10
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

Indizieren

Vektoren können indiziert werden, d.h. einzelne Elemente können herausgegriffen werden. Im Gegensatz zu vielen anderen Programmiersprachen beginnt R mit 1 zu zählen (d.h. das erste Element entspricht dem Index 1). Man verwendet dazu eckige Klammern, innerhalb derer die gewünschten Elemente angegeben werden.

```r
x <- seq(10, 110, 10)
x
```

```
 [1]  10  20  30  40  50  60  70  80  90 100 110
```

```r
x[4]  # 4. Element
```

```
[1] 40
```

```r
x[1:5]  # Elemente 1-5
```

```
[1] 10 20 30 40 50
```

```r
x[c(1, 4, 8)]  # Elemente 1, 4 und 8
```

```
[1] 10 40 80
```

Negative Indizes bedeuten "alle Elemente *außer* die angegebenen".

```r
x[c(-1, -10)]
```

```
[1]  20  30  40  50  60  70  80  90 110
```

```r
x[-c(1, 10)]
```

```
[1]  20  30  40  50  60  70  80  90 110
```

Man kann auch mit logischen Vektoren indizieren. Dazu erstellt man zuerst einen logischen Vektor und verwendet diesen dann als Index (dies kann direkt in einem Schritt gemacht werden). Dabei werden jene Elemente herausgegriffen, für die der Indexvektor `TRUE` ist.

```r
x
```

```
 [1]  10  20  30  40  50  60  70  80  90 100 110
```

```r
x > 40
```

```
 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```r
x[x > 40]
```

```
[1]  50  60  70  80  90 100 110
```

Idealerweise ist der logische Indexvektor gleich lang wie der zu indizierende Vektor. Falls der Indexvektor kürzer ist, findet wieder Recycling statt.

Benannte Vektoren

Vektoren können auch Elemente mit Namen enthalten. So kann man auch mit den Elementnamen anstatt mit der Position indizieren.

```r
vect <- c(a=11, b=2, c=NA)
vect
```

```
 a  b  c
11  2 NA
```

```
vect[2]
```

```
b
2
```

```
vect["b"]
```

```
b
2
```

Die Funktion `names` gibt die Namen der Elemente zurück.

```
names(vect)
```

```
[1] "a" "b" "c"
```

Mit dieser Funktion kann man die Elementnamen eines Vektors auch nachträglich setzen:

```
x <- 1:3
names(x)
```

```
NULL
```

```
names(x) <- c("test", "value", "x")
x
```

```
 test value     x
    1     2     3
```

Fehlende Werte

In R können fehlende Werte mit dem speziellen Wert `NA` (not available) codiert werden.

```
vect <- c(15, 1.12, NA, 12, NA, 33.22)
vect
```

```
[1] 15.00  1.12    NA 12.00    NA 33.22
```

Mit der Funktion `is.na` können die fehlenden Werte bestimmt werden. So kann man einfach alle Werte aus einem Vektor extrahieren, die nicht `NA` sind.

```
is.na(vect)  # alle fehlenden Werte
```

```
[1] FALSE FALSE  TRUE FALSE  TRUE FALSE
```

Mit dem `!` kann man einen logischen Vektor invertieren, d.h. alle `TRUE` werden `FALSE` und umgekehrt.

```
!is.na(vect)  # alle nicht fehlenden Werte
```

```
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
```

```
vect[!is.na(vect)]
```

```
[1] 15.00  1.12 12.00 33.22
```

Es gibt noch einen weiteren speziellen Typ für nicht definierte Ausdrücke namens `NaN` (not a number). Dieser Wert ist automatisch `NA`; umgekehrt ist aber `NA` nicht automatisch `NaN`.

```
0 / 0
```

```
[1] NaN
```

```
Inf - Inf  # Inf bedeutet unendlich
```

```
[1] NaN
```

```
is.na(0 / 0)
```

```
[1] TRUE
```

```
is.nan(0 / 0)
```

```
[1] TRUE
```

```
is.nan(NA)
```

```
[1] FALSE
```

Übungen

Übung 1

Berechnen Sie die Fläche sowie den Umfang eines Kreises mit dem Radius 5 (erzeugen Sie dafür eine Variable `r`). Speichern Sie die Ergebnisse in den Variablen `a` bzw. `u` ab. Wie lauten die Werte der beiden Variablen?

Übung 2

Erstellen Sie einen Vektor `x` mit den Elementen 4, 18, -7, 16, 4, 29, 8 und -44. Erstellen Sie danach einen Vektor `y`, welcher die quadrierten Elemente aus `x` enthält (nutzen Sie dazu die Eigenschaft, dass R Rechenoperationen elementweise durchführt). Zum Schluss erstellen Sie einen Vektor `z`, indem Sie `x` und `y` aneinanderhängen. Mit welcher Funktion können Sie die Anzahl der Elemente in `z` bestimmen?

Übung 3

Gegeben sei folgender Vektor:

```
x <- c(44, 23, -56, 98, 99, 32, 45, 19, 22)
```

Welche Elemente aus `x` sind gerade? Welche Elemente sind ungerade? Erstellen Sie zwei entsprechende logische Vektoren und geben Sie diese am Bildschirm aus. Geben Sie außerdem die entsprechenden Elemente von `x` aus.

Hinweis: Gerade Zahlen ergeben einen Rest von 0 wenn man sie durch 2 dividiert, ungerade einen Rest von 1. Verwenden Sie daher den Operator `%%` für den Rest einer Division.

Übung 4

Erstellen Sie folgende Vektoren:

- Einen Vektor mit den ganzen Zahlen von 15 bis 40.
- Einen Vektor mit den absteigenden Zahlen von 80 bis 60 in Dreierschritten.
- Einen Vektor bestehend aus 77 Zahlen zwischen 14 und 39.

Übung 5

Erstellen Sie einen Zeichenketten-Vektor mit folgenden Einträgen: zuerst 15 mal `"Placebo"`, dann 15 mal `"Group 1"` und schließlich 15 mal `"Group 2"`.

Übung 6

Erstellen Sie einen Vektor `k` mit den geraden Zahlen von 0 bis 40. Geben Sie dann durch Indizieren die folgenden Elemente dieses Vektors am Bildschirm aus:

- Alle Elemente bis auf das 8. und 9. Element.
- Die ersten fünf Elemente.
- Die Elemente 2, 5 und 26 (fällt Ihnen hier etwas auf?).
- Alle Elemente, die größer als 11 sind.

Übung 7

Erstellen Sie folgenden Vektor:

```
t <- c(10, 20, NA, 30, 40)
```

Berechnen Sie dann mit der Funktion `mean` den Mittelwert von `t`. Was bewirkt der fehlende Wert `NA`? Sehen Sie in der Hilfe zum Befehl `mean` nach, wie Sie fehlende Werte bei der Berechnung ignorieren können und führen Sie diese Berechnung durch.

Übung 8

Gegeben seien folgende Standardabweichungen von verschiedenen Variablen:

```
std <- c(1, 2.22, 11.3, 7.8, 3.4, 6)
```

Wie können Sie daraus die Varianzen berechnen?