Programmieren mit R für Einsteiger

2. Datentypen / 2.3 Zeichenketten

Berry Boessenkool 2022-02-25 11:40

Zeichenketten Grundlagen

```
"Moin Moin"  # Anführungsstriche:
'guten Tag'  # beide Sorten möglich
```

```
satz <- "Dies ist kein einheitlicher Satz"
class(satz)
## [1] "character"
```

```
nchar(satz) # Anzahl Zeichen
## [1] 32
```

```
tolower(satz)
## [1] "dies ist kein einheitlicher satz"
toupper(satz)
## [1] "DIES IST KEIN EINHEITLICHER SATZ"
```

```
substr(satz, start=7, stop=16) # Leerzeichen zählen mit
## [1] "st kein ei"
```

```
cat(satz) # concatenate and type
## Dies ist kein einheitlicher Satz
```

Zeichenketten zusammenfügen

```
paste("Wort", 1:4) # vektorisiert: 'Wort' 4 mal recycelt
## [1] "Wort 1" "Wort 2" "Wort 3" "Wort 4"
```

```
paste("Wort", 1:4, sep="_/") # Eigene Trennzeichenkette
## [1] "Wort_/1" "Wort_/2" "Wort_/3" "Wort_/4"
```

```
paste0("Wort", 1:4) # Leerer charstring '' als separator
## [1] "Wort1" "Wort2" "Wort3" "Wort4"
```

Mehrere Elemente zu einer einzigen Zeichenkette zusammenfügen:

```
paste0("Wort", 1:4, collapse="-")
## [1] "Wort1-Wort2-Wort3-Wort4"
```

```
toString(c("Diese", 1:5, "Wörter")) # kommagetrennt
## [1] "Diese, 1, 2, 3, 4, 5, Wörter"
```

Zeichenketten auftrennen

```
satz
## [1] "Dies ist kein einheitlicher Satz"
```

```
worte <- strsplit(satz, split=" ")[[1]]
```

`strsplit` gibt eine `list` als Ausgabe. Um das erste Element auszuwählen (das einen Vektor mit 5 Einträgen hat), nutzen wir doppelte eckige Klammern.

```
worte
## [1] "Dies"          "ist"           "kein"
## [4] "einheitlicher" "Satz"
```

Zeichenketten suchen

```
worte
## [1] "Dies"     "ist"       "kein"     "einheitlicher" "Satz"
```

```
match("kein", worte) # Index des ersten gleichen Eintrages
## [1] 3
match("ei", worte)        # nur komplette Übereinstimmungen
## [1] NA
```

```
"kein" %in% worte    # Logischer Wert, ob Eintrag vorkommt
## [1] TRUE
```

```
grep("ei", worte)      # in welchen Elementen 'ei' vorkommt
## [1] 3 4
```

```
grep("ei", worte, value=TRUE)  # Worte, die 'ei' enthalten
## [1] "kein"           "einheitlicher"
```

```
grepl("ei", worte)    # für jedes Wort: ist 'ei' enthalten?
## [1] FALSE FALSE  TRUE  TRUE FALSE
```

Zeichenketten ersetzen

```
worte
## [1] "Dies"     "ist"      "kein"     "einheitlicher" "Satz"
```

```
# Ersetze jeweils den ersten Fund:
sub(pattern="ei", replacement="EI", x=worte)
## [1] "Dies"           "ist"              "kEIn"
## [4] "EInheitlicher" "Satz"
```

```
# Ersetze alle Vorkommnisse (Auftreten) von 'ei'
gsub(pattern="ei", replacement="EI", x=worte)
## [1] "Dies"           "ist"              "kEIn"
## [4] "EInhEItlicher" "Satz"
```

Zusammenfassung

Zeichenketten (Character strings):

- `"zeichen"`, `'kette'`, `nchar`, `tolower`, `toupper`
- `paste(sep,collapse)`, `paste0`, `toString`
- `substr`, `strsplit`
- `match`, `%in%`, `grep`, `grepl`
- `sub`, `gsub`

Zeichenketten: Sonderzeichen Backslash

Ein Backslash signalisiert, dass danach was besonderes kommt.

```
cat("Satz mit\nZeilenumbruch") # \newline
## Satz mit
## Zeilenumbruch
cat("1\t9","1234\t9","12345678\t9", sep="\n") # \tabstop
## 1       9
## 1234    9
## 12345678        9
```

AltGr + ß,
Option + Shift + 7

```
cat("Satz mit \" Symbol") # Anführungsstrich
## Satz mit " Symbol
cat('Satz mit " Symbol') # weniger Tipparbeit :)
## Satz mit " Symbol
```

```
cat("Satz mit \\ literal") # Backslash selbst
## Satz mit \ literal
```

```
cat("Zeichenkette mit \U{0B00} Grad Symbol") # \Unicode
## Zeichenkette mit ° Grad Symbol
```

Warum `strsplit` eine Liste ausgibt

```
v <- c("ab-cdefg-hij-k-lmn", "opqrstuv-wxyz")
v
## [1] "ab-cdefg-hij-k-lmn" "opqrstuv-wxyz"
```

```
w <- strsplit(v, split="-")
w # Ein Element für jedes Element im Ursprungsvektor
## [[1]]
## [1] "ab"    "cdefg" "hij"   "k"     "lmn"
##
## [[2]]
## [1] "opqrstuv" "wxyz"
```

```
w[[1]]
## [1] "ab"    "cdefg" "hij"   "k"     "lmn"
```

Zeichenketten suchen II

```
worte
## [1] "Dies"     "ist"      "kein"     "einheitlicher" "Satz"
```

```
# für jedes Wort: an welcher Stelle 'ei' anfängt
c(regexpr("ei", worte)) # -1 wenn nicht drin
## [1] -1 -1  2  1 -1
 gregexpr("ei", worte)  # ditto: alle Stellen -> list
## -1       -1       2       1, 5       -1
```

regulärer Ausdruck (regex): Großschreibung, Anfang / Ende

grep: **g**lobal search for a **r**egular **e**xpression, **p**rint out matched lines

```
x <- c("abz", "Abz", "yzab", "abyz", "nichts")
```

```
grep("ab", x,v=T) # value=TRUE abgekürzt für kurze Folien
## [1] "abz"  "yzab" "abyz"
```

```
grep("ab", x,v=T, ignore.case=TRUE) # Großschreibung egal
## [1] "abz"  "Abz"  "yzab" "abyz"
```

```
grep("^ab", x,v=T) # caret: Muss anfangen mit
## [1] "abz"  "abyz"
```

```
grep("yz", x,v=T)
## [1] "yzab" "abyz"
```

```
grep("yz$", x,v=T) # dollar: Muss enden mit
## [1] "abyz"
```

Siehe auch: `startsWith` und `endsWith`

regulärer Ausdruck (regex): Wildcards

```
x <- c("cfu", "cfgu", "cfghu", "cnu", "cmu")
```

```
grep("c.u", x,v=T) # .: irgendein beliebiges Zeichen
## [1] "cfu" "cnu" "cmu"
```

```
grep("c.*u", x,v=T) # .*: egal wieviele beliebige Zeichen
## [1] "cfu"   "cfgu"  "cfghu" "cnu"   "cmu"
```

```
grep("c.{2}u", x,v=T) # .{2}: genau 2 beliebige Zeichen
## [1] "cfgu"
```

```
grep("c(f|n)u", x,v=T) # (x|y): x oder y
## [1] "cfu" "cnu"
```

```
grep("c[kmf]u", x,v=T) # [xyz]: irgendeins dieser Zeichen
## [1] "cfu" "cmu"
```

```
grep("c[^km]u", x,v=T) # [^xyz]: nicht diese Zeichen (normal ^anfang)
## [1] "cfu" "cnu"
```

```
grep("c[k-o]u", x,v=T) # [a-xA-X]: zwischen a und X
## [1] "cnu" "cmu"
```

regulärer Ausdruck (regex): Wiederholungen

repetition quantifiers für Häufigkeit des vorangehenden items:

```
x <- c("cd","cxd","cxxd","cxxxd","cxxxxd")
```

```
grep("cxd", x,v=T)
## [1] "cxd"
```

```
grep("cx?d"    , x,v=T) # ?: 0 oder 1 mal
## [1] "cd"  "cxd"
```

```
grep("cx*d"    , x,v=T) # *: 0 oder mehrfach
## [1] "cd"    "cxd"   "cxxd"   "cxxxd"  "cxxxxd"
```

```
grep("cx+d"    , x,v=T) # +: einmal oder öfter
## [1] "cxd"   "cxxd"   "cxxxd"  "cxxxxd"
```

```
grep("cx{2}d"  , x,v=T) # {n}: n mal
## [1] "cxxd"
```

```
grep("cx{2,}d" , x,v=T) # {n,}: n mal oder öfter
## [1] "cxxd"   "cxxxd"  "cxxxxd"
```

```
grep("cx{2,3}d", x,v=T) # {n,m}: n bis m mal
## [1] "cxxd"  "cxxxd"
```

regulärer Ausdruck (regex): Regex Operatoren ignorieren

```
". \ | ( ) [ { ^ $ * + ?" # regex metacharacters
```

```
x <- c("ab.de", "abde", "a^bcde", "bcde")
```

```
grep("^bc",   x, value=TRUE)
## [1] "bcde"
```

```
grep("^bc",   x, value=TRUE, fixed=TRUE) # ohne regex
## [1] "a^bcde"
```

```
grep(".de",   x, value=TRUE)
## [1] "ab.de"  "abde"   "a^bcde" "bcde"
```

```
grep("\\.de", x, value=TRUE) # echter Punkt (mit regex)
## [1] "ab.de"
```

Fertige Sammlungen von Zeichenketten

```
grep("[UV[:digit:]WX]", c("ab3d","abUd","abcd"), v=T)
## [1] "ab3d" "abUd"
```