R-programmering VT2023

Föreläsning 3

Josef Wilzén

2023-02-06

Linköpings Universitet

Föreläsning 3:

Seminarium på torsdag $\rightarrow$ behöver era frågor/feedback

Studieteknik $\rightarrow$ funkar det bra?

- Sammanfattning Föreläsning 2
- Villkorssatser
- Loopar
- Funktionsmeddelanden
- Debugging

Sammanfattning föreläsning 2

Matriser

- Två dimensionell vektor, alla element av samma typ
- Välj index med `[ "rad" , "kolumn" ]`
  - Saknas `rad` eller `kolumn` väljs hela.
  - Om man väljer ut bara är en rad eller kolumn görs resultatet om till en vektor.
    - För att behålla matrisstrukturen använd `drop = FALSE`
- `length` ger antal element
- `dim` ger dimensionerna

data.frame

- Dataset, där varje variabel (kolumn) kan ha olika typer.
- Varje variabel har ett namn.
  - Komma åt variabler via namnet eller ordningen.
- Varje rad innehåller ett värde per kolumn.
- Lätt att lägga till och ta bort kolumner.

Listor

- Tänk vektor där varje element kan vara vad som helst.
- Möjligt att namnge element.
- `[ ]` tar fram en del av listan
- `[[ ]]` tar fram elementet

Programkontroll

Programkontroll

- Kontrollera körningen av program eller funktioner
- Olika typer av kontroller vi kan göra:
  - Köra en annan del av koden
  - Vilkorsstyra kod
  - Kör kod upprepade antal gånger
  - Köra kod tills ett villkor är uppfyllt
  - Avbryta ett program i förtid

Villkorssatser

Vilkorssatser (if-else)

- Välja att utföra något baserat på logiskt villkor

```
if ( Villkor ) {
  Kod om Villkor == TRUE
} else {
  Kod om Villkor == FALSE
}
```

- Villkor ska vara någon logik
  - Tal större än noll är vanligtvis sanna.
- Om villkor är en vektor kommer den bara kolla på **första** elementet

If - else if - else

- Det är inte alltid man bara vill ha två olika utfall

```
if ( Villkor1 ) {
  Kod om Villkor1 == TRUE
} else if ( Villkor2 ) {
  Kod om Villkor2 == TRUE
} else if ( Villkor3 ) {
  Kod om Villkor3 == TRUE
} else {
  Kod om Villkor1 och Villkor2 och Villkor3 == FALSE
}
```

- Kommer välja den **första** som är sann

Villkorssatser - Exempel

```
prgm <- "R"
if ( prgm == "R") { cat("Kul med",prgm)}
```

Villkorssatser - Exempel

```
prgm <- "R"
if ( prgm == "R") { cat("Kul med",prgm)}
```

```
## Kul med R
```

Villkorssatser - Exempel

```
prgm <- "R"
if ( prgm == "R") { cat("Kul med",prgm)}
```

```
## Kul med R
```

```
prgm <- "Excel"
if ( prgm == "R") { cat("Kul med",prgm)}
```

Villkorssatser - Exempel II

```
prgm <- "Excel"
if ( prgm == "R") {
  cat("Kul med",prgm)
} else if ( prgm == "Python" ) {
  print("Okej")
} else {
  print("Hmm...")
}
```

Villkorssatser - Exempel II

```
prgm <- "Excel"
if ( prgm == "R") {
  cat("Kul med",prgm)
} else if ( prgm == "Python" ) {
  print("Okej")
} else {
  print("Hmm...")
}
```

```
## [1] "Hmm..."
```

Loopar (for-loop)

For-loop

- Upprepningar av kod
- I R används `for` för loopar över vektor/lista

```
for ( elem in vektor ) {
  # Kod som anropas en gång per element
  # elem är ett element i vektorn
}
```

- Vilken vektor eller lista kan användas
- Viktigt koncept: Loopens index (`elem`) är det ENDA som ändras i loopen.

For-loop - Exempel

```
vektor <- 3:5
for ( element in vektor ) {
  print( element*2 )
}
```

For-loop - Exempel

```
vektor <- 3:5
for ( element in vektor ) {
  print( element*2 )
}
```

```
## [1] 6
## [1] 8
## [1] 10
```

For-loop - Exempel II

```
vektor <- c("a","b","c","d")
for ( element in vektor ) {
  print( element )
}
```

For-loop - Exempel II

```
vektor <- c("a","b","c","d")
for ( element in vektor ) {
  print( element )
}
```

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

For-loop - Exempel III

- Kan loopa på flera olika sätt

```
for ( element in vektor ) {
  print( element )
}
```

```
for ( index in seq_along(vektor) ) {
  print( vektor[index] )
}
```

```
for ( index in 1:length(vektor) ) {
  print( vektor[index] )
}
```

Nästlade loopar

Nästlade loopar

- Om vi vill loopa i flera nivåer
- Delas ofta upp i yttre och inre loop
- Tänk som en klocka
  - Timmar: Yttersta loopen
  - Minuter: Inre loop, gör 60 iterationer / timme
  - Sekunder: Innersta loopen, gör 60 iteration / minut

Nästlade loopar - Exmepel

```
A <- matrix(1:6, nrow = 2)
for ( i in 1:2 ) {
  for ( j in 1:3 ) {
    text <- paste("rad:",i," kolumn:",j,
                  "värde:",A[i,j])
    print(text)
  }
}
```

Nästlade loopar - Exmepel

```
A <- matrix(1:6, nrow = 2)
for ( i in 1:2 ) {
  for ( j in 1:3 ) {
    text <- paste("rad:",i," kolumn:",j,
                  "värde:",A[i,j])
    print(text)
  }
}
```

```
## [1] "rad: 1  kolumn: 1 värde: 1"
## [1] "rad: 1  kolumn: 2 värde: 3"
## [1] "rad: 1  kolumn: 3 värde: 5"
## [1] "rad: 2  kolumn: 1 värde: 2"
## [1] "rad: 2  kolumn: 2 värde: 4"
## [1] "rad: 2  kolumn: 3 värde: 6"
```

while-loop

while-loop

- Om vi inte vet antalet iterationer på förhand

```
while ( Villkor ) {
  # Kod som anropas så länge Villkor == TRUE
}
```

- **Varning!** Kan fortsätta hur länge som helst
- **Obs!** Villkor måste kunna utvärderas innan loopen startar
- `repeat` repeterar kod till `break`

while-loop - Exempel

```
i <- 1 # Obs!
while ( i < 5 ) {
  print(i)
  i <- i + 1
}
```

while-loop - Exempel

```
i <- 1 # Obs!
while ( i < 5 ) {
  print(i)
  i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

Kontrollstrukturer för loopar

Kontrollstrukturer för loopar

- Vill ofta kontrollera hur looparna arbetar
- Finns följande kontrollstrukturer:

Kontrollstruktur	**Effet**
`next()`	Börja med nästa iteration direkt
`break()`	Avbryt den aktuella / innersta loopen
`stop()`	Avbryter allt och genererar ett felmeddelande

next() - Exempel

```
i <- 0
while ( i < 11 ) {
  i <- i + 1
  if ( i %% 2 == 0) { next() }
  print(i)
}
```

next() - Exempel

```
i <- 0
while ( i < 11 ) {
  i <- i + 1
  if ( i %% 2 == 0) { next() }
  print(i)
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
## [1] 11
```

break() - exempel

```
for ( i in 1:3 ) {
  for ( letter in c("a","b","c") ) {
    if ( letter == "b" ) { break() }
    print(letter)
  }
}
```

break() - exempel

```
for ( i in 1:3 ) {
  for ( letter in c("a","b","c") ) {
    if ( letter == "b" ) { break() }
    print(letter)
  }
}
```

```
## [1] "a"
## [1] "a"
## [1] "a"
```

stop() - Exempel

```
for ( i in 1:3 ) {
  for ( letter in c("a","b","c") ) {
    if ( letter == "b" ) { stop("Det blev fel!") }
    print(letter)
  }
}
```

stop() - Exempel

```
for ( i in 1:3 ) {
  for ( letter in c("a","b","c") ) {
    if ( letter == "b" ) { stop("Det blev fel!") }
    print(letter)
  }
}
```

```
## [1] "a"
```

```
## Error: Det blev fel!
```

Varningsmeddelanden och debugg

Varningsmeddelanden

- `stop()` avbryter funktioner/loopar och meddelar ett fel
- `warning()` skapar en varning, som inte avbryter
- Varningar sparas och skrivs ut sist
- `warnings()` skriver ut tidigare varningar

Varningsmeddelanden - Exempel

```
for ( chr in c("a","b") ) {
  print(chr)
  warning( paste("Farligt värde",chr) )
}
```

Varningsmeddelanden - Exempel

```
for ( chr in c("a","b") ) {
  print(chr)
  warning( paste("Farligt värde",chr) )
}
```

```
"a"
"b"
Warning messages:
1: Farligt värde a
2: Farligt värde b
```

Debugging

- Det uppstår ofta fel vid programmering
- Debugging handlar om att hitta orsaken
- Olika typer av fel:
  - Syntaktiska fel: Felaktig syntax i koden
  - Semantiska fel: Olämplig användning av objekt/funktioner
  - Logiska fel: Programmet löser inte det tänkta problemet

Debugging II

- Använd `cat()`, `print()` eller `message()` för att skriva ut värden under körning.
  - Använd `paste()` för att kombinera text och variabler till en sträng.
- `browser()` Hoppar in i funktionen
  - `n`: kör nästa rad
  - `c`: kör allt i funktion / loop
  - `Q`: avsluta
- `debugg()` Hoppa in i funktionen från början